



AFRL-RI-RS-TR-2015-253

MINIMIZING OVERHEAD FOR SECURE COMPUTATION AND FULLY HOMOMORPHIC ENCRYPTION: OVERHEAD

UNIVERSITY OF VIRGINIA

NOVEMBER 2015

FINAL TECHNICAL REPORT

APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED

STINFO COPY

**AIR FORCE RESEARCH LABORATORY
INFORMATION DIRECTORATE**

NOTICE AND SIGNATURE PAGE

Using Government drawings, specifications, or other data included in this document for any purpose other than Government procurement does not in any way obligate the U.S. Government. The fact that the Government formulated or supplied the drawings, specifications, or other data does not license the holder or any other person or corporation; or convey any rights or permission to manufacture, use, or sell any patented invention that may relate to them.

This report is the result of contracted fundamental research deemed exempt from public affairs security and policy review in accordance with SAF/AQR memorandum dated 10 Dec 08 and AFRL/CA policy clarification memorandum dated 16 Jan 09. This report is available to the general public, including foreign nationals. Copies may be obtained from the Defense Technical Information Center (DTIC) (<http://www.dtic.mil>).

AFRL-RI-RS-TR-2015-253 HAS BEEN REVIEWED AND IS APPROVED FOR PUBLICATION IN ACCORDANCE WITH ASSIGNED DISTRIBUTION STATEMENT.

FOR THE DIRECTOR:

/ S /

CARL THOMAS
Work Unit Manager

/ S /

MARK LINDERMAN
Technical Advisor, Computing
& Communications Division
Information Directorate

This report is published in the interest of scientific and technical information exchange, and its publication does not constitute the Government's approval or disapproval of its ideas or findings.

REPORT DOCUMENTATION PAGE				Form Approved OMB No. 0704-0188	
The public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Department of Defense, Washington Headquarters Services, Directorate for Information Operations and Reports (0704-0188), 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302. Respondents should be aware that notwithstanding any other provision of law, no person shall be subject to any penalty for failing to comply with a collection of information if it does not display a currently valid OMB control number. PLEASE DO NOT RETURN YOUR FORM TO THE ABOVE ADDRESS.					
1. REPORT DATE (DD-MM-YYYY) NOVEMBER 2015		2. REPORT TYPE FINAL TECHNICAL REPORT		3. DATES COVERED (From - To) OCT 2010 – JUN 2015	
4. TITLE AND SUBTITLE MINIMIZING OVERHEAD FOR SECURE COMPUTATION AND FULLY HOMOMORPHIC ENCRYPTION: OVERHEAD				5a. CONTRACT NUMBER FA8750-11-C-0080	
				5b. GRANT NUMBER N/A	
				5c. PROGRAM ELEMENT NUMBER 62303E	
6. AUTHOR(S) Abhi Shelat, Susan Hohenberger, Steven Myers, and Rafael Pass				5d. PROJECT NUMBER BL12	
				5e. TASK NUMBER 0U	
				5f. WORK UNIT NUMBER VA	
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) University of Virginia 1001E. Emmet St. Charlottesville, VA 22903-4833				8. PERFORMING ORGANIZATION REPORT NUMBER	
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES) Air Force Research Laboratory/RITA 525 Brooks Road Rome NY 13441-4505				10. SPONSOR/MONITOR'S ACRONYM(S) AFRL/RI	
				11. SPONSOR/MONITOR'S REPORT NUMBER AFRL-RI-RS-TR-2015-253	
12. DISTRIBUTION AVAILABILITY STATEMENT Approved for Public Release; Distribution Unlimited. This report is the result of contracted fundamental research deemed exempt from public affairs security and policy review in accordance with SAF/AQR memorandum dated 10 Dec 08 and AFRL/CA policy clarification memorandum dated 16 Jan 09.					
13. SUPPLEMENTARY NOTES					
14. ABSTRACT Investigating the overhead associated with Fully Homomorphic Encryption, this project examined a wide range of topics, including; Improving the efficiency of generating Yao's Garbled Circuits, Secure 2 Party Computation, Non-Maleable and Zero-Proof Secure Computation, Digital Signatures, Fully Homomorphic Encryption and Somewhat Homomorphic Encryption, Program Obfuscation and a secure Bitcoin implementation. Many papers were written (see appendices) and much of this work has been presented in premier Security, Privacy and Cryptography Conferences.					
15. SUBJECT TERMS Fully Homomorphic Encryption, Yao's Garbled Circuits, Secure Computation, Secure Function Evaluation, Threshold Fully Homomorphic Encryption, Zero-Knowledge Proof					
16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT UU	18. NUMBER OF PAGES 939	19a. NAME OF RESPONSIBLE PERSON CARL R. THOMAS
a. REPORT U	b. ABSTRACT U	c. THIS PAGE U			19b. TELEPHONE NUMBER (Include area code) N/A

CONTENTS

List of Figures	iv
List of Tables	iv
1 Summary	1
2 Introduction	2
3 Methods, Assumptions and Procedures	3
3.1 Yao's Garbled Circuits	3
3.1.1 Malicious Garbled Circuits on Compute Clusters	3
3.1.2 Semi-Honest Garbled Circuits on GPUs	3
3.2 General Theoretical Work	4
4 Results and Discussion	5
4.1 Secure 2-party Computation	5
4.1.1 Malicious Garbled Circuits on Compute Clusters	5
4.1.2 Parallelism	5
4.1.3 Semi-Honest Garbled Circuits on GPUs	7
4.1.4 PCF Compiler	10
4.1.5 Oblivious RAM Results	11
4.2 Theory of Secure Computation	12
4.2.1 Non-Malleable Commitments	13
4.2.2 Zero-Knowledge	13
4.3 Digital Signatures	14
4.3.1 Computing on Authenticated Data	14
4.3.2 Full Domain Hash from Multilinear Maps	15
4.3.3 Universal Aggregate Signatures	16
4.4 Encryption	16
4.4.1 New Approach for Chosen-Ciphertext Security	16
4.4.2 Results on Construction of Other Strong Forms of Encryption	17
4.4.3 Chosen-Ciphertext Security does not imply Circular Security	17
4.4.4 Online/Offline Attribute-Based Encryption	18
4.4.5 Blackbox proofs of knowledge of plaintext	18
4.4.6 Efficient Prototyped Bootstrapping FHE with SAGE	21
4.4.7 Tamper-resilient Cryptography	21
4.5 New Software Tools	22
4.5.1 Charm: A Toolkit for Rapid Prototyping of Cryptographic Systems	22
4.5.2 The AutoTools Suite	23
4.6 Protocols for Bitcoin	24
4.7 Program Obfuscation	25

4.7.1	Understanding Secure Computation in the Context of Complex Systems	25
5	Conclusions	26
6	References	28
	APPENDICES	34
A	List of Papers Resulting from Project	34
A.1	In preparation or submission	34
A.2	In print or to appear	34
A.3	CLP15:Constant-round Concurrent Zero-knowledge from Indistinguishability Obfuscation	38
A.4	BCP15:Large-Scale MPC	57
A.5	LP15:Succinct Garbling	77
A.6	LP16:Constant-round Non-malleable Commitments From Any One-way Function	127
A.7	HKW15: Adaptively Secure Puncturable Pseudorandom Functions in the Standard Model	154
A.8	APGR13: CloudSourcing Cryptography: Automating the Generation of Outsourced Cryptographic Algorithms	183
A.9	ABCHSW15: Computing on Authenticated Data	195
A.10	HKW15: Universal Signature Aggregators	236
A.11	AGHP14: Machine-Generated Algorithms and Proofs and Software	298
A.12	BCGGMTV14: Zerocash: Practical Decentralized Anonymous E-Cash from Bitcoin	338
A.13	GGMR14: Rational Zero: Economic Security from Zerocoin with Everlasting Security	354
A.14	HSW14: Replacing a Random Oracle	369
A.15	CP14:Parallel Repetition for Interactive Arguments	391
A.16	HW14: Online/Offline Attribute-Based Encryption	405
A.17	GGM14: Decentralized Anonymous Credentials	423
A.18	WHCSS14: SCORAM: Oblivious RAM for Secure Computation	435
A.19	AGH14: Using SMT Solvers to Automate Design Tasks for Encryption and Signature Schemes	448
A.20	HMSG13: GPU and CPU Parallelization of Honest-but-Curious Secure Two-Party Computation	460
A.21	MSS13: Blackbox Construction of A More Than Non-Malleable CCA1 Encryption Scheme from Plaintext Awareness	470
A.22	CLP13: Constant-Round Concurrent Zero Knowledge From P-Certificates	504
A.23	P13:Unprovable Security of NIZM and Non-malleable Commitments	543

A.24 MSS13: Black-Box Proof of Knowledge of Plaintext and Multiparty Computation with Low Communication Overhead	570
A.25 LP13:Black-box Constructions of Composable Protocols	591
A.26 CPT13: Interactive Coding Revisited	639
A.27 CPS13: Non-Black-Box Simulation from One-Way Functions And Applications to Resettable Security	673
A.28 SS13: Fast Two-Party Secure Computation with Minimal Assumptions . . .	684
A.29 HSW13: Full Domain Hash from (Leveled) Multilinear Maps and Identity-Based Aggregate Signatures	697
A.30 MGGR13: Zerocoin: Anonymous Distributed e-Cash from Bitcoin	730
A.31 KMBS13: PCF: A Portable Circuit Format For Scalable Two-Party Secure Computation	745
A.32 AGHP12: Machine-Generated Algorithms and Proofs and Software	761
A.33 HLW12: Detecting Dangerous Queries	776
A.34 CGH12: New Definitions and Separations for Circular Security	800
A.35 KSS12: Billion-Gate Secure Computation with Malicious Adversaries . . .	821
A.36 CPT12: The Knowledge Tightness of Parallel Zero-Knowledge	837
A.37 ABCHSW12: Computing on Authenticated Data	855
A.38 CHP11: Batch Verification of Short Signatures	890
A.39 GHW11: Outsourcing the Decryption of ABE Ciphertexts	915
Glossary of Terms	931

LIST OF FIGURES

1	1a) Circuit gate generation rates of [18] vs. our technique using fully parallelizable circuit generation. 1b) Gates generation rate per multi-processor on differing circuit sizes.	9
2	Gate Generation Times comparing to Kreuter et al.[37].	10

LIST OF TABLES

1	The time (in seconds) of running AES circuit with security parameter $k=80$ and $s=256$. The number of nodes represents the degree of parallelism on each side. “–” means that the time is smaller than 0.05 second and thus omitted.	6
2	The performance of our main protocol with $k = 80$ and $\sigma = 256$. All numbers in “time” column come from an average of 30 data points and have the 95% confidence interval $< 1\%$	6
3	The 95% two-sided confidence intervals for computation and communication time for each stage in the 1024-AES128 experiment	7
4	Benchmark system descriptions. EC2 runs a Xen virtual machine.	8

1 SUMMARY

The underlying theme of the PROCEED OVERHEAD project was to investigate the *fundamental limits that govern the construction of secure multi-party protocols*. The current approach to secure protocol design involves three steps:

1. devise a *bare* protocol that completes the collaborate task with no security—e.g., design a circuit or Turing machine that computes a function,
2. transform the bare protocol into one that is secure in the presence of *honest-but-curious* (HBC) adversaries, and
3. transform the HBC protocol into one that secure against arbitrarily malicious parties.

Over the 5 year project period, we have studied the choices and methods involved at each stage and have made significant progress on understanding fundamental overheads in each case. These overheads consist of (a) additional rounds or exchanges between the players, (b) additional communication (i.e., longer messages between players), (c) additional computation by all parties, (d) additional requirements on the types of players involved in the collaboration (e.g., an honest majority) needed to achieve security, and (e) additional complexity in concretely implementing the protocol.

Our results from this project analyze each type of overhead through either formal analytical measurement, characterization, or direct experimental measurement. Furthermore, we have developed several systems that put into practice state-of-the-art techniques for secure computation that have emerged from our analysis. As an example, our system for two-party secure computation against malicious adversaries that has been developed during PROCEED has improved the state-of-the-art performance by a factor of 10^4 and has increased the size of the functions that can be securely computed by a factor of 10^6 .

The results of these efforts appeared in 44 published papers, including the top theoretical venues such as STOC and FOCS, to cryptography flagship conferences including CRYPTO and Eurocrypt, to top applied venues such as IEEE Security and Privacy, USENIX Security and ACM Computer and Communications Security, and journals such as JACM. Six papers were invited to special issue on *best papers* from STOC, TCC, CRYPTO, SCN, and AsiaCrypt. The project also supported the development of open source code, including the Charm library, which has been downloaded thousands of times.

Our results are organized into a few broad categories. We have major results on *practical secure two-party computation*, as well as *theoretical characterizations* of the performance overheads for multi-party secure computation. Our project has also studied basic cryptographic primitives—*encryption algorithms* and *signature schemes*—that are used widely in all types of secure computation. These results either clarify performance issues with these primitives, develop new techniques for proving the security of efficient constructions, or clarify the relationship between security notions for encryption. Finally, we study other far-reaching topics related to secure computation, such as program obfuscation (an advanced technique in cryptography that could result in much simpler and more secure implementations of secure computation), and secure computations protocols for bitcoin.

2 INTRODUCTION

The US government and private industries are in great need of better ways to secure their data. The focus of this four-university collaborative project (University of Virginia, Cornell, Indiana University, Johns Hopkins University) was to investigate sophisticated methods of cryptographically protecting data with a focus on “reducing the overhead”—that is making the cryptography as efficient and practical as possible.

Over the course of this multi-year project, substantial progress was made on both applied and theoretical fronts. The project was comprised of over a dozen sub-projects, which ranged in topic from secure multi-party computation to Fully Homomorphic Encryption to digital signatures to Bitcoin and more. The results of these efforts appeared in 44 published papers, including the top theoretical venues such as STOC and FOCS, to cryptography flagship conferences including CRYPTO and Eurocrypt, to top applied venues such as IEEE Security and Privacy, USENIX Security and ACM Computer and Communications Security, and journals such as JACM. Six papers were invited to special issue on *best papers* from STOC, TCC, CRYPTO, SCN, and AsiaCrypt. The project also supported the development of open source code, including the Charm library, which has been downloaded thousands of times.

Brief overview We have constructed some of the world’s fastest and most practical secure two-party computation systems implementing Yao’s Garbled Circuit Protocol on varied parallel architectures, and under differing security models. The parallel architectures varied from compute-cluster based super computers in malicious security models, to reasonably priced GPU based parallelism in the honest-but-curious model. In addition, key supportive technologies to Yao’s Garbled Circuit Protocol implementations have been designed and implemented, such as a new effective 2-party circuit compiler was developed that constructed the largest circuits known at the time, and was the first to incorporate a number of circuit optimization technologies.

Second, we resolved some central open problems in the theory of secure computations and zero-knowledge protocols. Most notably, we demonstrated the first *constant-round* secure computation protocols based on minimal hardness assumptions, the first black-box secure computation protocols that remain secure under *concurrent executions*, the first constant-round concurrent zero-knowledge constructions, and the first *resetably-secure* protocol based on minimal hardness assumptions. We also provided the first construction of program obfuscation which can be *proven* secure based on a natural hardness assumptions on multilinear maps.

Third, this project explored the authentication analog of fully homomorphic encryption. We captured and strengthened in one definition several disjoint notions of computing on authenticated data existing in the literature. We then provided generic constructions for all univariate and closed predicates, and specific efficient constructions for a broad class of natural predicates such as quoting, subsets, weighted sums, averages, and Fourier transforms.

3 METHODS, ASSUMPTIONS AND PROCEDURES

We subdivide the projects that were worked on into several areas, shadowing the concepts discussed in the summary and introduction. Within each area we discuss the basic methods of each project.

3.1 Yao's Garbled Circuits

3.1.1 Malicious Garbled Circuits on Compute Clusters. Yao's garbled circuits technique enables the creation of a two-party secure computation protocol. Prior to the PROCEED project, our work (Eurocrypt 2011) described a two-party protocol that is secure in the malicious model based on this technique. As a result of PROCEED, over a five year period, we have discovered several new techniques that have lead to a substantial improvement in the area of two-party secure computation. At a very high level, our protocols can now handle very large instances of secure computation problems (i.e. we can securely compute functions that require billions of boolean gates to represent) at a speed that is measured in millions of boolean gates per second.

To achieve these results, we have developed new protocol techniques for achieving security against malicious adversaries (specifically, new techniques for handling problems involving input consistency between different instances of a problem, output authenticity, garbling techniques, and methods for handling a general class of "selective failure" attacks). Additionally, we have focused on improving the parallel complexity of secure computation protocols, and thus benefited tremendously from the recent trend in computer architecture to add more "cores" to each processor, and more processors to each computer.

We have also developed new systems to translate high-level C programs into boolean circuits. Our PCF, or "portable circuit format" compiler can now handle circuits with nearly trillions of gates. In work prior to this project (and even in our first version of our compiler), methods to translate programs into boolean circuits could handle only very small toy examples.

3.1.2 Semi-Honest Garbled Circuits on GPUs. Previous work demonstrated the feasibility and practical use of secure two-party computation [5, 9, 15, 23]. In this work, we presented the first Graphical Processing Unit (GPU)-optimized implementation of an optimized Yao's garbled-circuit protocol for two-party secure computation in the honest-but-curious and 1-bit-leaked malicious models. We implemented nearly all of the modern protocol advancements, such as Free-XOR, Pipelining, and OT extension. Our implementation was the first allowing entire circuits to be generated concurrently, and makes use of a modification of the XOR technique so that circuit generation is optimized for implementation on SIMD architectures of GPUs. In our best cases we generated about 75 million gates per second and we exceeded the state-of-the-art performance metrics on modern CPU systems by a factor of about 200, and GPU systems by about a factor of 2.3. While many recent works on garbled circuits exploit the embarrassingly parallel nature of many tasks that are part of a secure computation protocol, we showed that there are still various forms and levels of paralleliza-

tion that may yet improve the performance of these protocols. In particular, we highlight that implementations on the SIMD architecture of modern GPUs require significantly different approaches than the general purpose MIMD architecture of multi-core CPUs, which again differ from the needs of parallelizing on compute clusters. Additionally, modifications to the security models for many common protocols have large effects on reasonable parallel architectures for implementation.

3.2 General Theoretical Work

Many of the results of this project explore theoretical questions concerning secure computation. For these questions, the method is to pose a question concerning the necessity of some aspect of overhead for secure computation and mathematically explore whether that aspect is essential or can be removed.

4 RESULTS AND DISCUSSION

In this section, we highlight the results from the main aspects of our project.

4.1 Secure 2-party Computation

4.1.1 Malicious Garbled Circuits on Compute Clusters. Yao’s garbled circuits technique enables the creation of a two-party secure computation protocol. Prior to the PROCEED project, PI Shelat and Shen (Eurocrypt 2011) described a two-party protocol that is secure in the malicious model based on this technique. As a result of PROCEED, over a five year period, we have discovered several new techniques that have lead to a substantial improvement in the area of two-party secure computation. At a very high level, our protocols can now handle very large instances of secure computation problems (i.e. we can securely compute functions that require billions of boolean gates to represent) at a speed that is measured in millions of boolean gates per second.

To achieve these results, we have developed new protocol techniques for achieving security against malicious adversaries (specifically, new techniques for handling problems involving input consistency between different instances of a problem, output authenticity, garbling techniques, and methods for handling a general class of “selective failure” attacks). Additionally, we have focused on improving the parallel complexity of secure computation protocols, and thus benefited tremendously from the recent trend in computer architecture to add more “cores” to each processor, and more processors to each computer.

The protocol was the most efficient in terms of both asymptotics and real-world execution times. In this project, we have optimized that protocol and implemented it on a cluster of machines. We have demonstrated a 100x factor of improvement, and continue to make adjustments to discover the true overhead rate for malicious security. The table below summarizes some of the performance results. In order to run experiments on large garbled circuits, we have built a compiler that produces optimized circuits from programs (the Fair-Play circuit compiler cannot handle circuits with more than one million gates without running into memory problems). In this case, the optimizations favor XOR gates over NAND gates. Our compiler has produced the smallest known garbled Yao circuit for AES.

4.1.2 Parallelism. In 2012, in [KSS12] we illustrated how to design highly parallelizable protocols for 2-party secure computation that are secure in the malicious model. As the following chart showed, the running time for evaluating the AES circuit (in which one party holds a key k , the other party holds an input x , and the goal is to compute $AES_k(x)$) decreases as expected as we run the protocol on up to 256 cores. In this same paper, we report 6B gate circuit evaluation at a rate of approximately 100-120k/gates per second. By November 2012, we had improved our protocol to run at $> 300k$ gates per second in our system. We are investigating ways to remove the algebraic operations from the protocol; this will result in large performance improvements when the circuit has many inputs. We also improved our compiler infrastructure to handle very large circuits in a more scalable way. In Jan’13, we employed the AESNI and SSE2 instruction sets in our system. While the former improves

Table 1: The time (in seconds) of running AES circuit with security parameter $k=80$ and $s=256$. The number of nodes represents the degree of parallelism on each side. “–” means that the time is smaller than 0.05 second and thus omitted.

core #	2		4		8		16		32		64		128		256	
	Gen	Evl	Gen	Evl	Gen	Evl	Gen	Evl	Gen	Evl	Gen	Evl	Gen	Evl	Gen	Evl
OT	79.1	16.8	39.5	8.4	19.8	4.2	9.9	2.1	4.9	1.1	2.5	0.6	1.3	0.3	0.6	0.2
Gen.	19.6	–	9.3	–	4.5	–	2.2	–	1.1	–	0.5	–	0.3	–	0.1	–
Inp. chk	–	0.9	–	0.4	–	0.2	–	0.1	–	–	–	–	–	–	–	–
Eval.	7.1	16.2	3.2	7.6	1.7	3.5	0.8	1.7	0.4	0.8	0.3	0.4	0.2	0.2	0.1	0.1
Inter-com	11.8	83.6	5.7	41.3	2.5	20.5	1.4	10.2	0.6	5.1	0.3	2.6	0.1	1.4	0.1	0.7
Intra-com	0.5	0.5	0.2	0.3	0.2	0.2	0.1	0.2	0.1	0.1	–	0.1	–	0.1	–	0.1
Total time	118.0	118.1	58.0	58.0	28.6	28.6	14.4	14.4	7.2	7.2	3.7	3.7	1.9	1.9	1.0	1.0

the generation of the doubly encrypted entries for garbled truth table, the latter speeds up the bit-wise XOR in a 128-bit primitive operation manner. Moreover, we have designed a new generator’s input consistency check that gets rid of the number theoretic assumption and group operation entirely (except for OTs).

Reducing hardness assumptions By April 2013, we improved our protocol so that it no longer relies on any specific hardness assumptions (our prior work required discrete log or special types of proof systems) and yet has better performance. We do this by describing new solutions to the three main problems for malicious Yao with cut-and-choose. We solve input consistency using a hashing approach, we solve selective failure using a coding approach, and we solve two-output authenticity using a commit/hash approach. We show in Figure 2 the overall execution time of our system securely evaluating circuits EDT-4095¹, RSA-256², and 1024-AES128. Overall, our system is able to handle 650,000+ (or $\sim 200,000$ non-XOR) gates per second. We also observe that for all three circuits that we evaluated, more than 60% of the execution time is spent on communicating the huge amount of data, the garbled circuits. If we consider only the circuit garbling, the rate that our system actually achieves could be as high as 1,600,000+ (or 500,000+ non-XOR) gates per second, with the help of various optimization techniques, including SSE2 and AESNI instruction sets, and the free-XOR technique. These results are eventually published in our [SS13] paper.

Table 2: The performance of our main protocol with $k = 80$ and $\sigma = 256$. All numbers in “time” column come from an average of 30 data points and have the 95% confidence interval $< 1\%$.

circuit	gates	(non-XOR)	time (sec)	comm.
EDT-4095	5.9B	(2.4B)	9,042	18 TB
RSA-256	0.93B	(0.33B)	1,437	3 TB
1024-AES128	32M	(9.3M)	49	74 GB

¹This circuit computes the edit distance of two 4,095-bit inputs.

²This circuit computes a 256-bit modular exponentiation.

Table 3: The 95% two-sided confidence intervals for computation and communication time for each stage in the 1024-AES128 experiment

		Gen (sec)		Eval (sec)	Comm (MB)
OT	comp	$0.4 \pm 0.09\%$		–	6
	comm	$0.1 \pm$	1%	$0.3 \pm 0.6\%$	
cut-& chk	comp	–		–	9
	comm	–		–	
Inp. Chk	comp	$0.8 \pm$	1%	$0.3 \pm 0.2\%$	2,008
	comm	$0.3 \pm$	1%	$0.9 \pm 1\%$	
Evl.	comp	$11.4 \pm$	0.6%	$28.0 \pm 0.4\%$	72,271
	comm	$9.2 \pm$	1%	$30.3 \pm 0.8\%$	
Total	comp	$12.6 \pm$	0.3%	$28.0 \pm 0.2\%$	74,294
	comm	$9.6 \pm$	1%	$31.5 \pm 0.4\%$	

In the fall of 2013, we developed demos for DARPA based on this work; the demos involve a mail regex parser for checking security designations, and a graph intersection demo to determine if two people have a path in a secret graph to one another. We have also continued to develop the PCF 2.0 framework to address various bugs and gaps in the implementation to support the various groups (Sharemind, Georgia Tech) who also use the library.

4.1.3 Semi-Honest Garbled Circuits on GPUs. Previous work demonstrated the feasibility and practical use of secure two-party computation [18, 29, 41, 64]. In this work [31], we presented the first Graphical Processing Unit (GPU)-optimized implementation of an optimized Yao’s garbled-circuit protocol for two-party secure computation in the honest-but-curious and 1-bit-leaked malicious models. We implemented nearly all of the modern protocol advancements, such as Free-XOR, Pipelining, and OT extension. Our implementation was the first allowing entire circuits to be generated concurrently, and makes use of a modification of the XOR technique so that circuit generation is optimized for implementation on SIMD architectures of GPUs. In our best cases we generated about 75 million gates per second and we exceeded the state-of-the-art performance metrics on modern CPU systems by a factor of about 200, and GPU systems by about a factor of 2.3. While many recent works on garbled circuits exploit the embarrassingly parallel nature of many tasks that are part of a secure computation protocol, we showed that there are still various forms and levels of parallelization that may yet improve the performance of these protocols. In particular, we highlight that implementations on the SIMD architecture of modern GPUs require significantly different approaches than the general purpose MIMD architecture of multi-core CPUs, which again differ from the needs of parallelizing on compute clusters. Additionally, modifications to the security models for many common protocols have large effects on

reasonable parallel architectures for implementation.

Our system [31] is currently the most efficient gate garbler, and definitely the case when one considers the cost per gate. Our implementation did key work to ensure that garbelling was very much optimized to the SIMD architecture of the GPU. Full details are given in the paper. We benchmarked our system in comparison to the others to show the benefit of our SIMD optimizations. A full analysis of the effectiveness is given in the full paper in the Appendix. Here, for the benefit of the reader we provide the key results from that section.

Most prior work in the area benchmarks the time it takes to generate and evaluate various circuits. This process indirectly benchmarks the number of gates generated or evaluated per second. However, this is often run on systems with varying numbers of cores, and to a lesser extent varying speeds. We report results on the average number of gates generated or evaluated per second per core. We note this metric seems relatively stable, and thus we use it for a near apples-to-apples comparison. Table 4 has details for the comparison systems. We note that even though EC2 has multiple GPUs, only one is used in the results presented. EC2 is run on Amazon’s elastic compute infrastructure, and is running under a Xen hypervisor. Since we do not have direct access to the bare metal, we cannot determine how much overhead the Xen hypervisor entails, but Xen project benchmarks suggest, assuming appropriate kernel patches have been applied, a 0-30% performance decrease [10].

Table 4: Benchmark system descriptions. EC2 runs a Xen virtual machine.

System	CPU	Core/ Thrd.	GHz	Ram (GB)	GPU
Kreuter et al. [41]	Xenon E5506	4	2.13	8	N/A
EC2	Xenon X5570	8/16	2.93	24	Tesla S2050
Tie	Xenon E5-2620	12/12	2	64	Tesla K20

GPU	Cores	SMs	GHz	Memory (GB)	Compute Capability
S2050 (EC2)	448	14	1.15	2.7	2.0
K20 (Tie)	2496	13	0.71	4.8	3.5

We ran circuit generation on the EC2 and Tie systems (cf. Table 4). We first compared our results to those of Frederiksen and Nielsen [18] in Fig. 1a. We remind the reader that we compared their circuit generation times from experiments where they have similar, but not identical circuits. This is due to the need to simulate the cut-and-choose malicious protocol in the other systems, which we do not support. Further, while we did have access to their circuit file, we could not execute it directly as we do not support their file description language in our system, and their binary file format was not conducive to easy translation. Thus, we

show in Fig. 1a that under similar workloads our scheme outperforms theirs on the same hardware using the metric of gates generated per second. Observe that we generate gates at about 2.3 times the rate on the Tie system compared to Frederiksen and Nielsen on the EC2 system. Observe that we generate gates at about 3 times the rate on the Tie system compared to Frederiksen and Nielsen. This is the benchmark system, as Frederiksen’ and Nielsen’s code is targeted at compute capability 3.X CUDA cards.

As the number of cores on systems can be highly variable, in Fig. 1b we calculate the average rate of gate generation per core for the two systems, to help with understanding performance on other GPU cards with varying numbers of cores. Note that in the benchmarks reported in Figs. 1a and 1b we have commented out any code in our system necessary to split large circuits into smaller sub-circuits so that they can fit onto the GPU, as Frederiksen and Nielsen have no such corresponding code as they simply assume the circuit will fit. Thus we are not penalized for computing overhead that the other system also does not compute.

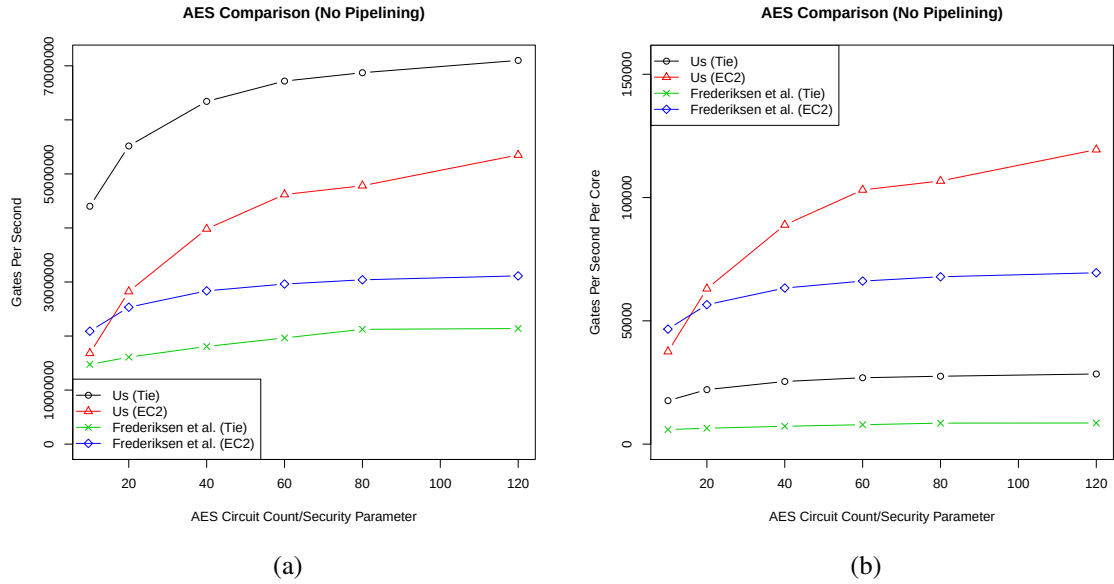


Figure 1: 1a) Circuit gate generation rates of [18] vs. our technique using fully parallelizable circuit generation. 1b) Gates generation rate per multi-processor on differing circuit sizes.

Next, we considered a number of different circuit sizes from both Kreuter et al.[37], and circuits that we have constructed. Given our support of PCF we can compare the same circuits as are tested by Kreuter et al.[37]. We see in Fig. 2, the absolute performance of our system versus that of Kreuter et al. in terms of Gates per sec, and then in Figs. 2a and 2b the relative performance per core. Note that performance per core is relatively stable across medium-to-large circuit sizes. Recall that our cores are substantially more abundant, and have lower cost and energy usage that those of Kreuter et al. Using the metric of gates per second we find our system, in the case of generation, provides significantly higher generation

rates: approximately three orders of magnitude. Our system tops out at around 75 million gates per second, while Kreuter et al tops out at 0.35 million gates per second. We note that their system is built for cluster computing, and so they pay a significant overhead to support it.

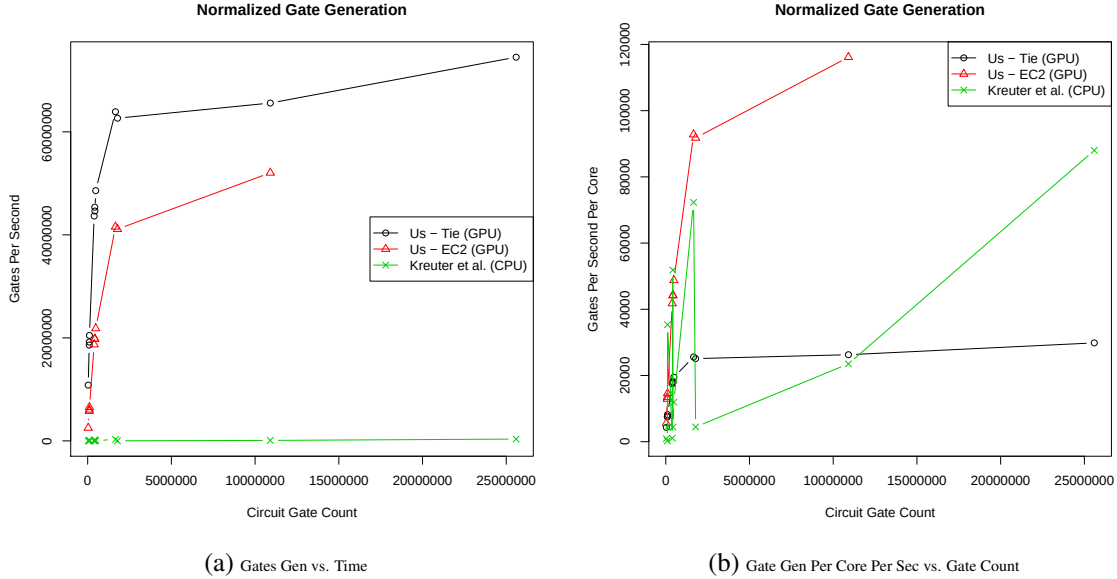


Figure 2: Gate Generation Times comparing to Kreuter et al.[37].

4.1.4 PCF Compiler. We have also developed new systems to translate high-level C programs into boolean circuits. Our PCF, or "portable circuit format" compiler can now handle circuits with nearly trillions of gates. In work prior to this project (and even in our first version of our compiler), methods to translate programs into boolean circuits could handle only very small toy examples.

The results of our PCFv1 compiler were described in [KMBS13] (USENIX Security'13). Previous approaches to compiling circuits have scaled poorly as the circuit size increases. Our approach is based on online circuit compression and lazy gate generation. We implemented an optimizing compiler for this new representation of circuits, and evaluated the use of this representation in two secure computation environments. Our evaluation demonstrates the utility of this approach, allowing us to scale secure computation beyond any previous system while requiring substantially less CPU time and disk space. In our largest test, we evaluate an RSA-1024 signature function with more than 42 billion gates, that was generated and optimized using our compiler. With our techniques, the bottleneck in secure computation lies with the cryptographic primitives, not the compilation or storage of circuits.

Since early 2014, we have been developing PCF2, which is an update to the original PCF work presented at Usenix'13. Improvements include better language support, more efficient

data structures, support for various methods to call special-purpose libraries, bindings to dozens of languages, and bug fixes throughout the pipeline.

4.1.5 Oblivious RAM Results. All known efficient constructions of generic secure two-party protocols require an *oblivious* representation of the function f to ensure that the control flow of the algorithm does not depend on its input and therefore leak partial information. The standard approach to creating an oblivious representation is to generate a boolean circuit from the description of f . This strategy is employed by dozens of prior works including all of our prior work in this project [50, 52, 4, 28, 40, 39, 47, 65] on secure computation.

When f is given as a RAM (Random Access Memory model) program, transforming f into a binary circuit may be problematic. A naive transformation replaces each indexed access to memory with a scan of the entire memory in order to keep the index hidden. To overcome this issue, Gordon et al [25] used an Oblivious RAM (ORAM) data structure proposed first by Goldreich [20] to compile RAM programs into secure computation protocols.

Intuitively, ORAM is a technique to transform a memory access (with *secret* index i) into a sequence of memory accesses (whose indices are revealed to the adversary but appear independent of the secret value i). ORAM techniques have been widely studied in other contexts [24, 61, 42, 23, 16, 6, 76, 74, 22, 20, 60, 63, 75, 12, 70, 67]. However, the goals of these prior works were (1) reducing the bandwidth overhead between the client and server; (2) reducing the *client storage*; and (3) reducing the server's overall memory overhead. Remarkably, state of the art approaches to ORAM design limit the overhead in all three aspects to various combinations of $O(\log^c(n))$ where $c \in \{0, 1, 2, 3\}$.

We embark on a comprehensive study of practical secure computation ORAM techniques. We do so via both theoretical analysis of several metrics used to judge ORAMs and experiments performed on optimized implementations of 4 state-of-the-art ORAM schemes. We report a new heuristic ORAM design that outperforms all other ORAMs we considered.

Our first observation is that traditional measures of ORAM schemes do not properly indicate the ORAM performance in secure computation setting. Previously, ORAM was primarily considered in storage outsourcing [69, 22, 68], and secure processor execution [49] settings. Thus, ORAM constructions were mainly evaluated by the *bandwidth overhead* (i.e., the number of data blocks retrieved per memory query).

Since the client-side computational logic needs to process secret values (e.g., the original index), their cost cannot be ignored as in traditional ORAMs designed merely for outsourcing storage. On the contrary, the overhead due to *securely* computing the client-side ORAM logic can easily dominate the overall cost in both bandwidth and CPU cycles. Therefore, the *circuit complexity* of the ORAM algorithm plays a critical role in evaluating the efficiency of ORAM schemes used in secure computation.

We find that, asymptotically speaking, the Binary Tree ORAM by Shi *et al.* [67] performs the same as a naively implemented Path ORAM [70], although Path ORAM is asymptotically faster in the data outsourcing scenario. In fact, due to the circuit size, Path ORAM is significantly slower for practical parameter settings. Next, we derive Path-SC ORAM, an optimized construction of Path ORAM that is $O(\log n)$ faster than its naive implementation.

However, because of the 3 oblivious sorts (resulting in a large constant factor in practice), its performance in practical parameter settings is still inferior to those theoretically slower schemes such as Binary Tree ORAM [67] and CLP ORAM [12].

We then present SCORAM, a heuristic *compact* ORAM design optimized for secure computation protocols. Our new design is almost 10x smaller in circuit size and also faster than all other designs we have tested for realistic settings (i.e., memory sizes between 4MB and 2GB, constrained by 2^{-80} failure probability). SCORAM makes it feasible to perform secure computations on gigabyte-sized data sets.

4.2 Theory of Secure Computation

Minimal Hardness Assumptions for $O(1)$ -round Secure Computation Joint with Lin and Venkatasubramanian, PI Pass investigated the minimal hardness assumptions needed for constant-round secure computation protocols. It is known that the existence of an “honest-but-curious” secure, so called, oblivious transfer (OT), protocol is both sufficient and necessary for construction of secure computation protocols. For constant-round secure computation, constant-round OT is necessary; the main open question is whether constant-round OT also suffices. (Our earlier results show that constant-round secure computation exists assuming the existence of enhanced trapdoor permutation; this assumption does not include, e.g., lattice-based assumption, etc). By relying on the above-mentioned non-malleable commitment scheme, we showed that indeed constant-round OT alone suffices to establish secure multi-party computation protocols for *any* function. This resolves a question left open since the inception of secure multi-party computation in 1987.

The paper appeared in Asiacrypt’12 and was invited to the special-issue on the best papers from the conferences.

Blackbox Concurrent Secure Computation PI Pass joint with Huijia Lin presented the first black-box construction of a secure multi-party computation protocol that satisfies a meaningful notion of concurrent security in the plain model (without any set-up, and without assuming an honest majority). Moreover, our protocol relies on the minimal assumption of the existence of a semi-honest OT protocol, and our security notion “UC with super-polynomial helpers” (Canetti et al, STOC’10) is closed under universal composition, and implies “super-polynomial-time simulation”.

The question of providing a black-box construction of a concurrently secure protocol was one of the key open problems in our original proposal. Our construction is currently still just a “feasibility” result, but it indicates that the tools for obtaining practical implementations of concurrently secure protocols are within reach. The paper appeared in *CRYPTO’12*.

Large Scale Secure Computation We are interested in secure computation protocols in settings where the number of parties is huge, and their data even larger. In this regime, the efficiency of existing solutions breaks down: either requiring resources linear in the circuit representation size of the function, or requiring parties to store and communicate information on the order of all parties’ combined inputs.

Assuming the existence of a single-use broadcast channel (per player), we demonstrate statistically secure n -party computation protocols for computing (multiple) arbitrary dynamic RAM programs over parties' inputs, handling $(1/3)$ fraction static corruptions, while preserving up to polylogarithmic factors the computation and memory complexities of the RAM program. Additionally, our protocol is load balanced across all parties, and achieves polylogarithmic communication locality (i.e., each party only ever needs to speak to $\text{poly-log}(n)$ other parties).

The paper was just accepted to *CRYPTO'15*.

4.2.1 Non-Malleable Commitments. One approach for achieving *round-efficient* secure multi-party computation protocols is to run many of its sub components in parallel. The problem with this approach is that the security of many standard cryptographic primitives does not necessarily remain intact when they are executed in parallel. For instance, a man-in-the-middle attacker participating in two simultaneous executions of a cryptographic protocol might use messages from one of the executions in order to violate the security of the second.

Non-malleable protocols block such attacks. The original paper by Dolev, Dwork and Naor from 1990 presented the first non-malleable protocols. Since then, non-malleable primitives have been extensively studied in the literature; the main focus of this study has been to improve the round-complexity of such protocols (indeed, if we want to use non-malleable protocols to improve the round-efficiency of secure computation protocols, it is essential that the non-malleable protocols themselves are round-efficient).

Constant-round Non-malleable Commitments Rafael Pass joint with his student Huijia Lin managed to completely resolve the round-complexity of non-malleable protocols, showing that constant-round protocols are possible using the minimal assumption of a one-way function; this had remained an open question since 1990 (this question was also explicitly mentioned in our DARPA proposal). A major application of this result is that *constant-round* secure multi-party computation protocols for any function are possible based on the existence of enhanced trapdoor permutations. This work resulted in a paper that is scheduled to appear in *Journal of the ACM* (the leading CS journal).

Non-interactive Non-maleable Commitments PI Pass investigated the possibility that non-interactive non-malleable commitment is possible. If such commitments were possible, it would dramatically decrease the round-complexity of secure computations protocols (our earlier results have established that a constant number of rounds suffice, but the precise constant is still rather high). In a paper appearing in *TCC'13*, Pass demonstrated that standard proof techniques cannot be used to prove security of such commitment schemes. The paper was invited to the "10-year anniversary of TCC" special issue in computational complexity, and also invited to the special-issue of best paper in TCC'13 in *Journal of Cryptology*. (A full version of the paper was just accepted for publication in the 10-year celebration issue.)

4.2.2 Zero-Knowledge.

Approved for Public Release; Distribution Unlimited.

Constant-round Concurrent Zero-knowledge from Falsifiable Assumption One of the main outstanding open problem in Concurrent Security is whether constant-round concurrent zero-knowledge protocols exists. We present a new, but falsifiable, hardness assumption under which a constant-round concurrent zero-knowledge protocol exists. The paper was accepted and presented at *FOCS'13*.

New Technique for Non-Black-Box Simulations: Resetable Security from One-way Functions The simulation paradigm, introduced by Goldwasser, Micali and Rackoff, is of fundamental importance to modern cryptography. In a breakthrough work from 2001, Barak (FOCS'01) introduced a novel non-black-box simulation technique. This technique enabled the construction of new cryptographic primitives, such as resettable-sound zero-knowledge arguments, that cannot be proven secure using just black-box simulation techniques. The work of Barak and its follow-ups, however, all require stronger cryptographic hardness assumptions than the minimal assumption of one-way functions: the work of Barak requires the existence of collision-resistant hash functions, and a very recent result by Bitansky and Paneth (FOCS'12) instead requires the existence of an Oblivious Transfer protocol.

We show how to perform non-black-box simulation assuming just the existence of one-way functions. In particular, we demonstrate the existence of a constant-round resettable-sound zero-knowledge argument based only on the existence of one-way functions. Using this technique, we determine necessary and sufficient assumptions for several other notions of resettable security of zero-knowledge proofs. An additional benefit of our approach is that it seemingly makes practical implementations of non-black-box zero-knowledge viable.

The paper was accepted in STOC'13 and invited to the special-issue in Siam JOC on best papers from *STOC'13*. We additionally wrote a follow-up paper on “simultaneous resettable security” that appeared in *FOCS'13*.

4.3 Digital Signatures

We now discuss functional and foundational advances in realizing practical digital signatures.

4.3.1 Computing on Authenticated Data. This project represents joint work involving four PROCEED members: Dan Boneh, Susan Hohenberger, Abhi Shelat and Brent Waters. It explores the authentication analog of fully homomorphic encryption. We capture and strengthen in one definition several disjoint notions of computing on authenticated data existing in the literature. We then provide generic constructions for all univariate and closed predicates, and specific efficient constructions for a broad class of natural predicates such as quoting, subsets, weighted sums, averages, and Fourier transforms. The original work appeared in TCC 2012 and a full version appeared in the Journal of Cryptology in 2015.

Brent Waters and Susan Hohenberger also explored expanding this notion to computing on authenticated graphs. We developed a candidate approach, which generalizes the “untraceable linking” techniques used in the substring construction of the prior TCC paper, but were unable to completely analyze it. That is, we were still exploring if the mechanisms

Approved for Public Release; Distribution Unlimited.

that connected two adjoining letters in a text document can be generalized to connect any two nodes in a graph. This subproject was downgraded as a priority when the breakthrough result of realizing multilinear maps appeared by Garg, Gentry and Halevi. Armed with this new mathematical tool, our focus shifted to more foundational authentication problems, which we now describe in Section 4.3.2.

4.3.2 Full Domain Hash from Multilinear Maps. Applying a full domain hash is a common technique in cryptography where a hash function, modeled as a random oracle, is used to hash a string into a set. Originally, the concept referred to a signature scheme where one hashed into the range of a trapdoor permutation (Bellare-Rogaway 1993). Subsequently, full domain hash has been treated as a more general concept. Pairing-based applications of Full Domain Hash include: the original Boneh-Franklin identity-based encryption schemes, short and aggregate signatures, Hierarchical Identity-Based Encryption, and decentralized Attribute-Based Encryption. Typically, proofs of such schemes will use the random oracle heuristic to “program” the output of the hash function in a certain way for which there is no known standard model equivalent.

Given that there are well-known issues with random oracle instantiability in general and problems with Full Domain Hash in particular, there has been a push to find standard model realizations of these applications.

In this project we explore building constructions with full domain hash structure, but with standard model proofs that do not employ the random oracle heuristic. The launching point for our results will be the utilization of a “leveled” *multilinear map* setting for which Garg, Gentry, and Halevi (GGH) gave an approximate candidate. Our first step is the creation of a standard model signature scheme that exhibits the structure of the Boneh, Lynn and Shacham signatures. In particular, this gives us a signature that admits unrestricted aggregation.

We build on this result to offer an *identity-based* aggregate signature scheme that admits unrestricted aggregation. In our construction, an arbitrary-sized set of signatures on identity/message pairs can be aggregated into a single group element, which authenticates the entire set. The identity-based setting has important advantages over regular aggregate signatures in that it eliminates the considerable burden of having to store, retrieve or verify a set of verification keys, and minimizes the total cryptographic overhead that must be attached to a set of signer/message pairs. While identity-based signatures are trivial to achieve, their aggregate counterparts are not. To the best of our knowledge, no prior candidate for realizing unrestricted identity-based aggregate signatures exists in either the standard or random oracle models.

A key technical idea underlying these results is the realization of a hash function with a Naor-Reingold-type structure that is publicly computable using repeated application of the multilinear map. We expect this to have wider applications. We present our results in a generic “leveled” multilinear map setting and then show how they can be translated to the GGH graded algebras analogue of multilinear maps.

This work was performed by Susan Hohenberger, Amit Sahai and Brent Waters and accepted to CRYPTO 2013.

Next, we improved our prior results for full domain hash to provide a method to instantiate the random oracle with a concrete hash function in these applications. This has been an open problem since random oracles were proposed by Bellare and Rogaway in 1993, and full domain hash applications now encompass a broader range of notable cryptographic schemes including the Boneh-Franklin IBE scheme and Boneh-Lynn-Shacham (BLS) signatures. All of the above described schemes required a hash function that had to be modeled as a random oracle to prove security. Our work utilizes recent advances in indistinguishability obfuscation to construct specific hash functions for use in these schemes. We then prove security of the *original* cryptosystems when instantiated with our specific hash function.

This work was performed by Susan Hohenberger, Amit Sahai and Brent Waters and accepted to Eurocrypt 2014.

4.3.3 Universal Aggregate Signatures. We introduce the concept of universal signature aggregators. In a universal signature aggregator system, a third party, using a set of common reference parameters, can aggregate a collection of signatures produced from *any* set of signing algorithms (subject to a chosen length constraint) into one short signature whose length is independent of the number of signatures aggregated. In prior aggregation works, signatures can only be aggregated if all signers use the same signing algorithm (e.g., BLS) and shared parameters. A universal aggregator can aggregate across schemes even in various algebraic settings (e.g., BLS, RSA, ECDSA), thus creating novel opportunities for compressing authentication overhead. It is especially compelling that *existing* public key infrastructures can be used and that the signers do not have to alter their behavior to enable aggregation of their signatures.

We provide multiple constructions and proofs of universal signature aggregators based on indistinguishability obfuscation and other supporting primitives. We detail our techniques as well as the tradeoffs in features and security of our solutions.

This work was performed by Susan Hohenberger, Venkata Koppula and Brent Waters and appeared in Eurocrypt 2015.

4.4 Encryption

We now discuss advances in security and efficiency for encryption.

4.4.1 New Approach for Chosen-Ciphertext Security. Chosen-ciphertext security is the deployable standard for encryption, but yet can sometimes be difficult to realize.

In this project, we presented a new approach for creating chosen-ciphertext secure encryption. The focal point of our work is a new abstraction that we call *Detectable Chosen-Ciphertext Security* (DCCA). Intuitively, this notion is meant to capture systems that are not necessarily chosen ciphertext attack (CCA) secure, but where we can detect whether a certain query CT can be useful for decrypting (or distinguishing) a challenge ciphertext CT^* .

We show how to build chosen ciphertext secure systems from DCCA security. We motivate our techniques by describing multiple examples of DCCA systems including creating

them from 1-bit CCA secure encryption — capturing the Myers-shelat result (FOCS 2009). Our work identifies DCCA as a new target for building CCA secure systems.

This is joint work by Susan Hohenberger, Allison Lewko and Brent Waters that appeared in Eurocrypt 2012.

4.4.2 Results on Construction of Other Strong Forms of Encryption. We construct [56] a Non-Malleable Chosen Ciphertext Attack (NM-CCA1) encryption scheme from any encryption scheme that is also plaintext aware and weakly simulatable. We believe this is the first construction of a NM-CCA1 scheme that follows strictly from encryption schemes with seemingly weaker or incomparable security definitions to NM-CCA1. Previously, the statistical Plaintext Awareness #1 (PA1) notion of security for encryption was only known to imply CCA1. Our result is therefore novel because unlike the case of Chosen Plaintext Attack (CPA) and Chosen Ciphertext Attack (CCA2), it is unknown whether a CCA1 scheme can be transformed into an NM-CCA1 scheme. Additionally, we show both the Damgård Elgamal Scheme (DEG) and the Cramer-Shoup Lite Scheme (CS-Lite) are weakly simulatable under the DDH assumption. Since both are known to be statistical Plaintext Aware 1 (PA1) under the Diffie-Hellman Knowledge (DHK) assumption, they instantiate our scheme securely.

Furthermore, in response to a question posed by Matsuda and Matsuura, we define cNM-CCA1-security in which an NM-CCA-adversary is permitted to ask a $c \geq 1$ number of parallel queries after receiving the challenge ciphertext. We extend our construction to yield a cNM-CCA1 scheme for any constant c . All of our constructions are black-box. This work was completed by Mona Sergi (a graduate student whose PhD was partially funded by this project), abhi shelat, and Steven Myers.

4.4.3 Chosen-Ciphertext Security does not imply Circular Security. Traditional definitions of encryption security guarantee secrecy for any plaintext that can be computed by an outside adversary. In some settings, such as anonymous credential or disk encryption systems, this is not enough, because these applications encrypt messages that depend on the secret key. A natural question to ask is do standard definitions capture these scenarios? One area of interest is *n-circular security* where the ciphertexts $E(pk_1, sk_2), E(pk_2, sk_3), \dots, E(pk_{n-1}, sk_n), E(pk_n, sk_1)$ must be indistinguishable from encryptions of zero. Acar et al. (Eurocrypt 2010) provided a CPA-secure public key cryptosystem that is not 2-circular secure due to a distinguishing attack.

In this work, we consider a natural relaxation of this definition. Informally, a cryptosystem is *n-weak circular secure* if an adversary given the cycle $E(pk_1, sk_2), E(pk_2, sk_3), \dots, E(pk_{n-1}, sk_n), E(pk_n, sk_1)$ has no significant advantage in the regular security game, (e.g., CPA or CCA) where ciphertexts of chosen messages must be distinguished from ciphertexts of zero. Since this definition is sufficient for some practical applications and the Acar et al. counterexample no longer applies, the hope is that it would be easier to realize, or perhaps even implied by standard definitions. We show that this is unfortunately not the case: even this weaker notion is not implied by standard definitions. Specifically, we show:

- For symmetric encryption, under the minimal assumption that one-way functions exist, n -weak circular (CPA) security is not implied by CCA security, for any n . In fact, it is not even implied by authenticated encryption security, where ciphertext integrity is guaranteed.
- For public-key encryption, under a number-theoretic assumption, 2-weak circular security is not implied by CCA security.

In both of these results, which also apply to the stronger circular security definition, *we actually show for the first time an attack in which the adversary can recover the secret key of an otherwise-secure encryption scheme after an encrypted key cycle is published*. These negative results are an important step in answering deep questions about which attacks are prevented by commonly-used definitions and systems of encryption. They say to practitioners: if key cycles may arise in your system, then even if you use CCA-secure encryption, your system may break catastrophically; that is, a passive adversary might be able to recover your secret keys.

This is joint work with David Cash, Matthew Green and Susan Hohenberger that appeared in PKC 2012.

4.4.4 Online/Offline Attribute-Based Encryption. Attribute-based encryption (ABE) is a type of public key encryption that allows users to encrypt and decrypt messages based on user attributes. For instance, one can encrypt a message to any user satisfying the boolean formula (“crypto conference attendee” AND “PhD student”) OR “IACR member”. One drawback is that encryption and key generation computational costs scale with the complexity of the access policy or number of attributes. In practice, this makes encryption and user key generation a possible bottleneck for some applications.

To address this problem, we developed new techniques for ABE that split the computation for these algorithms into two phases: a preparation phase that does the vast majority of the work to encrypt a message or create a secret key *before* it knows the message or the attribute list/access control policy that will be used (or even the size of the list or policy). A second phase can then rapidly assemble an ABE ciphertext or key when the specifics become known. This concept is sometimes called “online/offline” encryption when only the message is unknown during the preparation phase; we note that the addition of unknown attribute lists and access policies makes ABE significantly more challenging.

One motivating application for this technology is mobile devices: the preparation work can be performed while the phone is plugged into a power source, then it can later rapidly perform ABE operations on the move without significantly draining the battery.

Our performance estimates showed that over 99% of the computational work could be moved to the offline phase in many scenarios.

This is joint work with Susan Hohenberger and Brent Waters that appeared in PKC 2014.

4.4.5 Blackbox proofs of knowledge of plaintext. We develop [57, 56] a novel proof of knowledge of a plaintext protocol and show how to use it in the construction of a fully

black-box multi-party computation protocol with low communication overhead. We briefly describe the motivation behind our work.

Secure computation with an honest majority can be accomplished without any cryptographic assumptions, but the best such protocol requires the parties to communicate $|f| \log |f| + d^2 \cdot \text{poly}(n, \log |f|)$ bits [15] and at least d rounds. Here $|f|$ is the size of the function being computed and d is the circuit depth of f , and thus the communication of the protocol is super-linearly related to the number of gates in f . Until recently, even the use of cryptographic assumptions for secure computation required $\text{polylog}(\lambda)$ communication overhead per gate [15] where λ is a security parameter.

Gentry [19] circumvents per-gate overhead as follows: the honest-but-curious parties use secure multi-party computation to generate an FHE key, each party encrypts its input, and sends the resulting ciphertext and proof to other parties. Once all parties have encryptions of everyone's inputs, they compute the function of interest locally using the evaluation procedure of the FHE. Finally, to use the resulting ciphertexts as inputs to a secure multi-party computation which computes the decryption of the majority input. In order to be secure against malicious adversaries, the Naor and Nissim compiler [59], which makes use of the PCP theorem, can be applied. The use of the PCP theorem in the SMC steps makes the approach impractical, even when presented with a practical FHE scheme.

The motivation behind our work is to remove any use white-box techniques, such as the PCP theorem or generic ZK or NIZK, from the above framework for constructing communication-efficient secure protocols. These techniques have historically been inefficient. In other words, we seek a black-box transformation from TFHE to secure computation.

First Contribution The main technical hurdle in devising a black-box transformation from TFHE to secure computation is to implement the requirement for each player to prove that they “know the plaintext” corresponding to the encrypted input that they have broadcast. This step is essential because it prevents one player from copying (or mauling via the homomorphism) the input of a player who has acted earlier. To handle this step, we show how to construct a two-round black-box proof of knowledge of an encrypted bit for any circuit private FHE scheme using only the encryption scheme. Since our protocol is only two rounds, it is not zero-knowledge (cf. [21]), but can provably keep the encrypted bit hidden. Our POK requires that the public-key contain a labeled encryption of 0 and 1, which given all known FHE schemes seems to be a natural modification.³ For traditional FHE schemes, the POK can be used completely black-box, without even the need for the modification.

The basic idea of our proof of knowledge protocol is to first modify the encryption scheme so that the message is encoded using an error-correcting code (ECC) based verifiable secret sharing (VSS) scheme. To encrypt a message we first generate its secret shares, and encrypt them independently using fresh randomness. A verifier now requests the Prover to reveal the randomness used to encrypt a sub-threshold number of the shares. The verifier

³Since all current schemes contain bit-wise encryptions of their own secret-keys which are random bit strings, and a natural extension of any protocol that provides encryptions of one's own secret-key can be used to derive a labeled encryption of 0 and 1 which we describe.

then does a consistency check, based on the ECC underlying the scheme, to ensure that the shares were encoded properly. In particular, the error-correcting code we choose offers a property that allows one to check whether local parts of the codeword are error-free. The verifier accepts if everything appears to be properly coded. Since the number of shares revealed is less than the threshold, it does not leak any information about the original message. To show a proof of knowledge property, we argue that an extractor can rewind the Prover and ask for another set of shares to be opened. With high probability, this second transcript provides enough new shares to run the VSS recover algorithm, and recover the original message. The one issue with this approach is that the Prover must reveal the randomness used to encrypt some of the shares. The semantic security of an encryption scheme does not guarantee any security when these random bits are revealed—in particular, the security of the rest of the unopened encryptions are not guaranteed. Instead, we require the encryption scheme to be secure against a selective opening attack (SOA). Fortunately, a result of Hemenway et al. [27] can be generalized to show that any circuit private homomorphic encryption scheme can be made into an SOA-secure one.

We point out that our proof of knowledge requires the encryption scheme to be homomorphic and circuit-private. Recently, Damgård et al. [17] demonstrates a three-round Σ -protocol for knowledge of plaintext, but their protocol *requires* the underlying encryption scheme to also be homomorphic on the random coins used to encrypt. Although many FHE schemes support this property on their random coins, it is certainly not specified in the definition of FHE. In contrast, circuit privacy has been independently defined and seems to be a naturally weaker property.⁴ Moreover, their scheme requires the message space for the FHE to be over $\mathbb{Z}_N$ for N related to the security parameter. While in general, single-bit FHE implies many-bit FHE, we are not aware of any such transformation that *also* preserves the homomorphism over the random coins as required by their protocol. Thus, the requirement for large message space *and* homomorphism over the random coins seem to be extra assumption which our work can avoid (our protocol also works on single-bit FHE). Finally, the Σ -protocol from [17] must be compiled into a full zero-knowledge protocol using standard techniques which add round complexity and/or setup assumptions; we show that our two-round protocol with its hidden-bit property suffices for our secure computation protocol.

Second Contribution By combining our result with almost any TFHE scheme, we construct a secure multi-party protocol that avoids *both* per-gate communication complexity *and* white-box techniques such as the PCP theorem or Zero-Knowledge. The communication complexity of our protocol is $O(\lambda^c \cdot n^2)$ where λ is a security parameter and c is a small constant for the TFHE scheme and is thus independent of $|f|$. Our black-box transformation is particularly important because if practical FHE (and TFHE) can be constructed, our transformations will result in practical SFE. Our work is in the standard model and does not require trust assumptions such as the common reference string, a random oracle or public-key setup.

⁴Even though current schemes achieve circuit privacy via randomness homomorphisms, it is certainly plausible for future constructions to achieve circuit privacy in other ways. Moreover, there do not seem to be any natural ways to transform a circuit private scheme to one with a randomness homomorphism, and thus we feel it is a weaker notion.

Final Contribution For completeness, we also construct a *threshold* fully homomorphic public-key encryption scheme (TFHE) based on the Approximate GCD problem and the fully homomorphic encryption scheme presented by van Dijk et al. [72], and our result was the first to demonstrate the feasibility of directly achieving this threshold primitive for FHE. Since our original eprint submission, [3] and [48] present more efficient TFHE constructions based on LWE-style assumptions. The point of this construction is to demonstrate feasibility of TFHE under different complexity assumptions.⁵

4.4.6 Efficient Prototyped Bootstrapping FHE with SAGE. We provided several implementations of FHE that incorporate several different optimizations. All implementations were done in SAGE[71], which is an open-source advanced mathematics package that has an incorporated programming language based on Python. Our first implementation was one that implemented bootstrapping working on the RLWE based scheme of Brakerski and Vaikuntanathan [9], with improvements as suggested in Lauter, Naehrig and Vaikuntanathan [58]. We benchmarked our implementation in Sage of the non-bootstrapping scheme against the implementation in Magma of the Lauter, Naehrig and Vaikuntanathan system [58] and get slightly faster results. (Note, Lauter et al. did not implement bootstrapping, so there is no possibility of providing a comparison here). This suggests that there is no preference in choosing the Magma implementation system which is costly and proprietary, over SAGE. The second implementation was Brakerski's new algorithm, We did not implement bootstrapping, but did implement more advanced error handling mechanisms, specifically the modular reduction technique.

The SAGE system, while excellent for prototyping code was too inefficient to handle large realistic security parameters. Therefore, we looked into the possibility of augmenting the SAGE system with a backend that could handle polynomial algebra on the GPU. We were successful in writing small programs that access GPGPU through SAGE, going through pycuda. However, after some experimentation and implementation of basic mathematics on the GPU, we realized that the architecture was not promising. The reason was that the bus is too slow to have individual polynomial multiplication and addition operations pushed across it, to have them performed on the GPU, but this is the only method that SAGE seems to support the outsourcing of such operations. Therefore, while we believe there is much promise in using the GPU to perform FHE operations, accessing it through SAGE is not a promising approach.

4.4.7 Tamper-resilient Cryptography.

On the Impossibility of Tamper-resilient Cryptography We initiate a study of the security of cryptographic primitives in the presence of efficient tampering attacks to the randomness of honest parties. More precisely, we consider p-tampering attackers that may tamper

⁵We note that historically, threshold encryption has been presented where the key-generation algorithm and decryption algorithms are single algorithms, or they are multi-party protocols. We present multi-party protocols.

with each bit of the honest parties' random tape with probability p , but have to do so in an "online" fashion. We present both positive and negative results:

- Any secure encryption scheme, bit commitment scheme, or zero-knowledge protocol can be broken with probability p by a p -tampering attacker. The core of this result is a new Fourier analytic technique for biasing the output of bounded-value functions, which may be of independent interest (and provides an alternative, and in our eyes simpler, proof of the classic Santha-Vazirani theorem).
- Assuming the existence of one-way functions, cryptographic primitives such as signatures, identification protocols can be made resilient to p -tampering attacks for any $p = 1/n^\alpha$, where $\alpha > 0$ and n is the security parameter.

The paper is appeared in *CRYPTO'14* and was invited to the special issues of the best papers in the conference.

4.5 New Software Tools

In this section, we describe open source software that was used by or partially developed with the support of this DARPA funding. This work formed the basis for a 2013 PhD dissertation for JHU graduate student Joseph Ayo Akinyele, although he was not funded on this grant.

4.5.1 Charm: A Toolkit for Rapid Prototyping of Cryptographic Systems. Matt Green led the development of Charm, an extensible framework designed for rapid prototyping of cryptographic systems that utilize the latest advances in cryptography, such as fully-homomorphic encryption and SFE, as well as the traditional cryptographic functionalities. Charm is designed to minimize code complexity, promote code re-use, and to automate interoperability, while not compromising on efficiency. It was first publicly presented in NDSS 2012. The work to build Charm was not part of PROCEED, but Charm was an available foundation for some of the implementation tasks we proposed. The Charm team interacted during the project with Galois and UVA on making this code available and easy to use by the PROCEED team. It was also used for the development of further tools as noted in Section 4.5.2. A dedicated website for Charm was launched at

<http://www.charm-crypto.com/>

According to Github records, the library has been downloaded thousands of times worldwide.

The library continues to be actively maintained and a paper on it was published with the following information: Joseph A. Akinyele, Christina Garman, Ian Miers, Matthew W. Pagano, Michael Rushanan, Matthew Green, Aviel D. Rubin: Charm: a framework for rapidly prototyping cryptosystems. *J. Cryptographic Engineering* 3(2): 111-128 (2013).

4.5.2 The AutoTools Suite. Cryptographic design tasks are primarily performed by hand today. Shifting more of this burden to computers could make the design process faster, more accurate and less expensive. In this subproject, we investigated tools for programmatically altering existing cryptographic constructions to reflect particular design goals. Our techniques enhance both security and efficiency with the assistance of advanced tools including Satisfiability Modulo Theories (SMT) solvers. A dedicated website for these automation tools was launched at

<https://github.com/JHUISI/auto-tools>

Specifically, we developed the following suite of tools:

AutoBatch AutoBatch is an automated tool for generating batch verification code in either Python or C++ from a high level representation of a signature scheme. AutoBatch outputs both software and, for transparency, a LaTeX file describing the batching algorithm and arguing that it preserves the unforgeability of the original scheme.

We tested AutoBatch on over a dozen pairing-based schemes to demonstrate that a computer could find competitive batching solutions in a reasonable amount of time. In particular, it found an algorithm that is faster than a batching algorithm from Eurocrypt 2010. Another novel contribution is that it handles *cross-scheme* batching, where it searches for a common algebraic structure between two distinct schemes and attempts to batch them together.

We also published a journal version that expanded on the ACM CCS 2012 original work in a number of ways. We added a new loop-unrolling technique and showed that it helps cut the batch verification cost of one scheme by roughly half. We described our pruning and search algorithms in greater detail, including pseudocode and diagrams. All experiments were also re-run using the RELIC pairing library. We compared those results to our earlier results using the MIRACL library, and discuss why RELIC outperforms MIRACL in all but two cases. Automated proofs of several new batching algorithms are also included.

AutoGroup AutoGroup converts a pairing-based encryption or signature scheme written in (simple) symmetric group notation into a specific instantiation in the more efficient, asymmetric setting. Some existing symmetric schemes have hundreds of possible asymmetric translations, and this tool allows the user to optimize the construction according to a variety of metrics, such as ciphertext size, key size or computation time.

AutoStrong The AutoStrong tool focuses on the security of digital signature schemes by automatically converting an existentially unforgeable signature scheme into a *strongly* unforgeable one. The main technical challenge here is to automate the “partitioned” check, which allows a highly-efficient transformation.

CloudSource CloudSource is an automated tool for developing “outsourcing-ready” decryption of cryptographic schemes. CloudSource is designed to programmatically analyze existing pairing-based encryption schemes, and to derive new algorithms that allow users to outsource portions of the decryption routine to an untrusted server.

The CloudSource tool addresses a growing need, as we increasingly deploy cryptography in environments where users employ limited mobile devices and yet have ready access to cloud-based computing resources. The techniques we propose form part of a growing line of work aimed towards the automated analysis and engineering of cryptographic protocols, and will help to reduce the need for manual optimization of these constructions.

Our experiments demonstrate that these design tasks studied can be performed automatically in (usually) a matter of seconds.

This work includes collaborations by Susan Hohenberger, Matthew Green, Joseph Ayo Akinyele, Matthew Pagano, and Avi Rubin. Multiple papers appeared in ACM CCS and the Journal of Computer Security.

4.6 Protocols for Bitcoin

Bitcoin is the first e-cash system to see widespread adoption. While Bitcoin offers the potential for new types of financial interaction, it has significant limitations regarding privacy. Specifically, because the Bitcoin transaction log is completely public, users' privacy is protected only through the use of pseudonyms. In this project, we propose Zerocoin, a cryptographic extension to Bitcoin that augments the protocol to allow for fully anonymous currency transactions. Our system uses standard cryptographic assumptions and does not introduce new trusted parties or otherwise change the security model of Bitcoin. We detail Zerocoin's cryptographic construction, its integration into Bitcoin, and examine its performance both in terms of computation and impact on the Bitcoin protocol. This work involved Ian Miers, Christina Garman, Matthew Green and Avi Rubin, and appeared in IEEE Security and Privacy 2013.

Although payments are conducted between pseudonyms, Bitcoin cannot offer strong privacy guarantees: payment transactions are recorded in a public decentralized ledger, from which much information can be deduced. Zerocoin (Miers et al., IEEE S&P 2013) tackles some of these privacy issues by unlinking transactions from the payments origin. Yet it still reveals payment destinations and amounts, and is limited in functionality. In this project, we construct a full-fledged ledger-based digital currency with strong privacy guarantees. Our results leverage recent advances in zero-knowledge Succinct Non-interactive ARguments of Knowledge (zk-SNARKs). We formulate and construct decentralized anonymous payment schemes (DAP schemes). A DAP scheme lets users pay each other directly and privately: the corresponding transaction hides the payments origin, destination, and amount. We provide formal definitions and proofs of the constructions security. We then build Zerocash, a practical instantiation of our DAP scheme construction. In Zerocash, transactions are less than 1 kB and take under 6 ms to verify orders of magnitude more efficient than the less-anonymous Zerocoin and competitive with plain Bitcoin. This work involved an MIT-JHU collaboration, including Eli Ben-Sasson, Alessandro Chiesa, Christina Garman, Matthew Green, Ian Miers, Eran Tromer and Madars Virza, and appeared in IEEE Security and Privacy 2014.

4.7 Program Obfuscation

Program obfuscation serves to "scramble" a computer program, hiding its implementation details while preserving its functionality. Unfortunately, the "dream" notion of security, guaranteeing that obfuscated code does not reveal information beyond black-box access to the original program, has historically run into strong impossibility results, and is known to be impossible to achieve for general programs.

Recently, the first plausible candidate of general-purpose obfuscation was presented (Garg et al FOCS 2013) for a relaxed notion of security, known as *indistinguishability* obfuscation, which requires only that obfuscations of functionally equivalent programs are indistinguishable.

We initiated the study of the stronger notion of *extractability* or "differing-inputs" obfuscation: An extractability obfuscator for a class of algorithms M guarantees that if an efficient attacker A can distinguish between obfuscations of two algorithms $M_1, M_2 \in M$, then A can efficiently recover (given M_1 and M_2) an input on which M_1 and M_2 provide different outputs (Barak et al JACM 2012). We demonstrate that extractability obfuscation provides several new applications, including obfuscation of Turing machines, indistinguishability-secure functional encryption for an unbounded number of key queries and unbounded message spaces, and a new notion of *functional witness encryption*. We also explore the relation between extractability obfuscation and other cryptographic notions and show that in special cases, extractability obfuscation is in fact implied by indistinguishability obfuscation. This paper was accepted to TCC'14, and has since been quite influential.

In a different vein, we investigated constructions of obfuscations that can provably be based on some cryptographic hardness assumptions. This was a wide-open problem. In our work, we stipulated new hardness assumptions for multilinear maps and provided the first provably-secure construction of obfuscation based on a succinct hardness assumption. The paper appeared in *CRYPTO'14*.

In an orthogonal approach we investigate whether obfuscation implies other crypto-graphic hardness assumption. We show that indistinguishability obfuscation plus a slight variant of the assumption that $NP \not\subseteq BPP$ implies the existence of one-way functions. This result appeared in FOCS'14.

4.7.1 Understanding Secure Computation in the Context of Complex Systems. In some secure computation schemes, such as bitcoin, there is the need for all parties to engage in the system and trust it for it to be of use and produce value. Other large scale secure computation schemes share this property. In [30], we consider how complex systems effect security in a number of settings producing value (e.g., SMC) and harm. We consider ways this can be taxonomized and studied.

5 CONCLUSIONS

This project resulted in several high-quality publications that appeared at top venues, in 3 high-quality implementations of secure computation protocols and the tools needed to construct such protocols, and in several software artifacts that were demonstrated to DARPA officials during the demonstration days held annually throughout the performance period.

Two-party Secure Computation In the project area of secure two-party computation, published papers include [73, 26, 31, 36, 66, 38]. These papers correspond to a number of high-performance implementations of Yao’s Garbled Circuit protocols, and related machinery. In [31] there was a SIMD implementation of Yao’s circuits implemented on GPGPU architectures that showed that even in the honest-but-curious model, the protocol could be made so computationally efficient as to have the communication overhead dominate the computation.

Zero-Knowledge Our progress on Interactive Proofs and Zero-knowledge can be found in [44, 35, 54, 34, 32, 13, 62, 33, 14]. Most notable, we presented the first concurrent zero-knowledge protocol, new parallel repetition theorems, and our journal paper on non-malleability (which appeared in JACM) gives the first constant-round non-malleable commitment and zero-knowledge protocols based on one-way functions.

Encryption and Authentication In the area of encryption and authentication research, published papers include [2, 57, 1]. Understanding security requirements for encryption schemes and their relative power is essential for understanding the effectiveness of cryptographic primitives used in different settings. In [55, 56] we show the first construction to achieve more than CCA1 security from a primitive that has only a weaker form of plaintext awareness. We initiated the study of “randomness-dependent” encryption schemes in [5]. We present several new lower bounds on the efficiency of cryptographic primitives in [53, 51].

Multi-Party Computation In the project we developed a protocol that introduced the first threshold decryption scheme for a leveled fully homomorphic encryption scheme [57] and showed how to use it to achieve secure multi-party computation which asymptotically met known lower bounds. The SMC scheme itself was black-box and only relied on threshold FHE as a building block. More of our work on Secure Computation can be found in [8, 11, 45, 43]; most notably, we gave the first black-box constructions of concurrently secure protocols.

Obfuscation Our work on Obfuscation can be found in [46, 7]; most notably, we gave the first applications of differing-input obfuscation, and provided new constructions of indistinguishability obfuscation.

Complex Systems We did some work [30] in the direction of considering how to construct a taxonomy of complex systems as they relate to secure computing and networking systems. Understanding how large systems can become secure and trusted or insecure, due to individual actors. Large SMC systems such as crypto-currencies rely on such properties.

6 REFERENCES

REFERENCES

- [1] Jae Hyun Ahn, Dan Boneh, Jan Camenisch, Susan Hohenberger, Abhi Shelat, and Brent Waters. Computing on authenticated data. In *Theory of Cryptography - 9th Theory of Cryptography Conference, TCC 2012, Taormina, Sicily, Italy, March 19-21, 2012. Proceedings*, pages 1–20, 2012. 26
- [2] Jae Hyun Ahn, Dan Boneh, Jan Camenisch, Susan Hohenberger, Abhi Shelat, and Brent Waters. Computing on authenticated data. *J. Cryptology*, 28(2):351–395, 2015. 26
- [3] Gilad Asharov, Abhishek Jain, and Daniel Wichs. Multiparty computation with low communication, computation and interaction via threshold FHE. *IACR Cryptology ePrint Archive*, 2011:613, 2011. 21
- [4] Assaf Ben-David, Noam Nisan, and Benny Pinkas. FairplayMP: A System for Secure Multi-party Computation. In *ACM Conference on Computer and Communications Security*, 2008. 11
- [5] Eleanor Birrell, Kai min Chung, Rafael Pass, and Sidharth Telang. Randomness-dependent message security. In *TCC12*, 2012. 26
- [6] Dan Boneh, David Mazieres, and Raluca Ada Popa. Remote oblivious storage: Making oblivious RAM practical. <http://dspace.mit.edu/bitstream/handle/1721.1/62006/MIT-CSAIL-TR-2011-018.pdf>, 2011. 11
- [7] Elette Boyle, Kai min Chung, and Rafael Pass. On extractability obfuscation. In *TCC 2014*, 2014. 26
- [8] Elette Boyle, Kai min Chung, and Rafael Pass. Large-scale multi-party computation. In *Crypto 2015*, 2015. 26
- [9] Zvika Brakerski and Vinod Vaikuntanathan. Fully homomorphic encryption from ring-lwe and security for key dependent messages. In *CRYPTO*, pages 505–524, 2011. 21
- [10] Ian Campbell. Baremetal vs. xen vs. kvm — redux. <http://blog.xen.org/index.php/2011/11/29/baremetal-vs-xen-vs-kvm-redux/>, Nov 2011. 8
- [11] Ran Canetti, Huijia Lin, and Rafael Pass. From unprovability to composable security. In *FOCS 2013*, 2013. 26
- [12] Kai-Min Chung, Zhenming Liu, and Rafael Pass. Statistically-secure oram with $\tilde{O}(\log 2n)$ overhead, 2013. 11, 12

- [13] Kai-Min Chung, Rafail Ostrovsky, Rafael Pass, and Ivan Visconti. Simultaneous resetability from one-way functions. 2013. 26
- [14] Kai-Min Chung and Rafael Pass. The randomness complexity of parallel repetition. In *FOCS*, 2011. 26
- [15] Ivan Damgård, Yuval Ishai, and Mikkel Krøigaard. Perfectly secure multiparty computation and the computational overhead of cryptography. In *EUROCRYPT*, pages 445–465, 2010. 19
- [16] Ivan Damgård, Sigurd Meldgaard, and Jesper Buus Nielsen. Perfectly secure oblivious RAM without random oracles. In *TCC*, 2011. 11
- [17] Ivan Damgård, Valerio Pastro, Nigel P. Smart, and Sarah Zakarias. Multiparty computation from somewhat homomorphic encryption. *IACR Cryptology ePrint Archive*, 2011:535, 2011. 20
- [18] Tore Kasper Frederiksen and Jesper Buus Nielsen. Fast and maliciously secure two-party computation using the gpu. Technical report, Cryptology ePrint Archive, Report 2013/046, 2013. <http://eprint.iacr.org>, 2012. iv, 7, 8, 9
- [19] Craig Gentry. *A fully homomorphic encryption scheme*. PhD thesis, Stanford University, 2009. crypto.stanford.edu/craig. 19
- [20] O. Goldreich. Towards a theory of software protection and simulation by oblivious RAMs. In *STOC*, 1987. 11
- [21] Oded Goldreich and Yair Oren. Definitions and properties of zero-knowledge proof systems. *J. Cryptology*, 7(1):1–32, 1994. 19
- [22] Michael T. Goodrich and Michael Mitzenmacher. Privacy-preserving access of outsourced data via oblivious RAM simulation. In *ICALP*, 2011. 11
- [23] Michael T. Goodrich, Michael Mitzenmacher, Olga Ohrimenko, and Roberto Tamassia. Oblivious RAM simulation with efficient worst-case access overhead. In *CCSW*, 2011. 11
- [24] Michael T. Goodrich, Michael Mitzenmacher, Olga Ohrimenko, and Roberto Tamassia. Privacy-preserving group data access via stateless oblivious RAM simulation. In *SODA*, 2012. 11
- [25] S. Dov Gordon, Jonathan Katz, Vladimir Kolesnikov, Fernando Krell, Tal Malkin, Mariana Raykova, and Yevgeniy Vahlis. Secure two-party computation in sublinear (amortized) time. In *CCS*, pages 513–524, 2012. 11
- [26] Chih hao Shen and Abhi Shelat. Fast two-party secure computation with minimal assumptions. In *ACM CCS 2013*, 2012. 26

- [27] Brett Hemenway, Benoit Libert, Rafail Ostrovsky, and Damien Vergnaud. Lossy encryption: Constructions from general assumptions and efficient selective opening chosen ciphertext security. Technical Report 2009/088, eprint.iacr.org, 2009. Cryptology ePrint Archive. 20
- [28] Yan Huang, David Evans, Jonathan Katz, and Lior Malka. Faster Secure Two-Party Computation Using Garbled Circuits. In *USENIX Security Symposium*, 2011. 11
- [29] Yan Huang, Jonathan Katz, and David Evans. Quid-pro-quo-tocols: Strengthening semi-honest protocols with dual execution. In *Security and Privacy (SP), 2012 IEEE Symposium on*, pages 272–284. IEEE, 2012. 7
- [30] Nathaniel Husted and Steven Myers. Emergent properties & security: The complexity of security as a science. In Konstantin Beznosov, Anil Somayaji, Tom Longstaff, and Paul C. van Oorschot, editors, *Proceedings of the 2014 workshop on New Security Paradigms Workshop, Victoria, BC, Canada, September 15-18, 2014*, pages 1–14. ACM, 2014. 25, 27
- [31] Nathaniel Husted, Steven Myers, Abhi Shelat, and Paul Grubbs. Gpu and cpu parallelization of honest-but-curious secure two-party computation. In *ACSAC'13*, pages 169–178, 2013. 7, 8, 26
- [32] R. Pass K. Chung and S. Telang. Knowledge-preserving interactive coding. In *FOCS 2013*, 2013. 26
- [33] Dustin Tseng Kai-min Chung, Rafael Pass. The knowledge tightness of parallel zero-knowledge. In *TCC 2012*, 2012. 26
- [34] Rafael Pass Kai-min Chung, Huijia Lin. Constant-round concurrent zero-knowledge from p-certificates. In *FOCS 2013*, 2013. 26
- [35] Rafael Pass Kai-min Chung, Huijia Lin. Constant-round concurrent zero-knowledge from indistinguishability obfuscation. In *Crypto 2015*, 2015. 26
- [36] Ben Kreuter, Chih hao Shen, and Abhi Shelat. Billion-gate secure computation with malicious adversaries. In *USENIX Security 2012*, 2012. 26
- [37] Ben Kreuter, Ben Mood, abhi shelat, and Kevin Butler. Pcf: A portable circuit format for scalable two-party secure computation. In *To Appear in USENIX Security 2013*, 2013. iv, 9, 10
- [38] Benjamin Kreuter, Benjamin Mood, Kevin Butler, and abhi shelat. Two-output secure computation with malicious adversaries. In *Usenix Security 2013*, 2013. 26
- [39] Benjamin Kreuter, Benjamin Mood, Abhi Shelat, and Kevin Butler. PCF: A Portable Circuit Format for Scalable Two-Party Secure Computation. In *USENIX Security Symposium*, 2013. 11

- [40] Benjamin Kreuter, Abhi Shelat, and Chih hao Shen. Billion-Gate Secure Computation with Malicious Adversaries. In *USENIX Security Symposium*, 2012. 11
- [41] Benjamin Kreuter, Abhi Shelat, and Chih-Hao Shen. Billion-gate secure computation with malicious adversaries. In *Proceedings of the 21st USENIX conference on Security symposium, Security*, volume 12, pages 14–14, 2012. 7, 8
- [42] Eyal Kushilevitz, Steve Lu, and Rafail Ostrovsky. On the (in)security of hash-based oblivious RAM and a new balancing scheme. In *SODA*, 2012. 11
- [43] Huija Lin, Rafael Pass, and Muthu Venkitasubramaniam. Uc from ot. In *AsiaCrypt’12*, 2012. 26
- [44] Huijia Lin and Rafael Pass. Constant-round non-malleable commitments from any one-way function. In *STOC*, pages 705–714, 2011. 26
- [45] Huijia Lin and Rafael Pass. Black-box constructions of composable protocols. In *CRYPTO 2013*, 2013. 26
- [46] Huijia Lin and Rafael Pass. Succinct garbling schemes and application. In *To appear in STOC 2015 in a merged version*, 2015. 26
- [47] Chang Liu, Yan Huang, Elaine Shi, Jonathan Katz, and Michael Hicks. Automating efficient ram-model secure computation. *IEEE S & P*, 2014. 11
- [48] Adriana Lopez-Alt, Eran Tromer, and Vinod Vaikuntanathan. Cloud-assisted multiparty computation from fully homomorphic encryption. *IACR Cryptology ePrint Archive*, 2011:663, 2011. 21
- [49] Martin Maas, Eric Love, Emil Stefanov, Mohit Tiwari, Elaine Shi, Kriste Asanovic, John Kubiatowicz, and Dawn Song. Phantom: Practical oblivious computation in a secure processor. In *CCS*, 2013. 11
- [50] Philip MacKenzie, Alina Oprea, and Michael Reiter. Automatic Generation of Two-party Computations. In *ACM Conference on Computer and Communications Security*, 2003. 11
- [51] Mohammad Mahmoody and Rafael Pass. The curious case of non-interactive commitments. In *CRYPTO’12*, 2012. 26
- [52] Dahlia Malkhi, Noam Nisan, Benny Pinkas, and Yaron Sella. Fairplay: A secure two-party computation system. In *USENIX Security*, 2004. 11
- [53] Kai min Chung, Huijia Lin, Mohammad Mahmoody, and Rafael Pass. On the power of non-uniform proofs of security. In *ITCS’13*, 2013. 26
- [54] Kai min Chung andi Rafaeli Pass. Parallel repetition for interactive arguments. In *SIGACT News 2014*, 2014. 26

- [55] Steven Myers, Mona Sergi, and Abhi Shelat. Blackbox construction of a more than non-malleable CCA1 encryption scheme from plaintext awareness. In Ivan Visconti and Roberto De Prisco, editors, *Security and Cryptography for Networks - 8th International Conference, SCN 2012, Amalfi, Italy, September 5-7, 2012. Proceedings*, volume 7485 of *Lecture Notes in Computer Science*, pages 149–165. Springer, 2012. 26
- [56] Steven Myers, Mona Sergi, and Abhi Shelat. Black-box construction of a more than non-malleable CCA1 encryption scheme from plaintext awareness. *Journal of Computer Security*, 21(5):721–748, 2013. 17, 18, 26
- [57] Steven Myers, Mona Sergi, and Abhi Shelat. Black-box proof of knowledge of plaintext and multiparty computation with low communication overhead. In *TCC*, pages 397–417, 2013. 18, 26
- [58] Michael Naehrig, Kristin Lauter, and Vinod Vaikuntanathan. Can homomorphic encryption be practical? In *CCSW*, pages 113–124, 2011. 21
- [59] Moni Naor and Kobbi Nissim. Communication preserving protocols for secure function evaluation. In *Proceedings of the thirty-third annual ACM symposium on Theory of computing*, STOC '01, pages 590–599, New York, NY, USA, 2001. ACM. 19
- [60] R. Ostrovsky. Efficient computation on oblivious RAMs. In *STOC*, 1990. 11
- [61] Rafail Ostrovsky and Victor Shoup. Private information storage (extended abstract). In *STOC*, 1997. 11
- [62] Rafael Pass Per Austrin, Johan Hastad. On the power of many 1-bit provers. In *Innovations in Theoretical Computer Science Conference 13*, 2013. 26
- [63] Benny Pinkas and Tzachy Reinman. Oblivious RAM revisited. In *CRYPTO*, 2010. 11
- [64] Shi Pu, Pu Duan, and Jyh-Charn Liu. Fastplay—a parallelization model and implementation of smc on cuda based gpu cluster architecture. Technical report, Cryptology ePrint Archive, Report 2011/097, 2011. <http://eprint.iacr.org>, 2011. 7
- [65] Aseem Rastogi, Matthew A. Hammer, and Michael Hicks. Wysteria: A programming language for generic, mixed-mode multiparty computations. *IEEE S & P*, 2014. 11
- [66] Abhi Shelat and Chih-Hao Shen. Two-output secure computation with malicious adversaries. In Kenneth G. Paterson, editor, *EUROCRYPT*, volume 6632 of *Lecture Notes in Computer Science*, pages 386–405. Springer, 2011. 26
- [67] Elaine Shi, T.-H. Hubert Chan, Emil Stefanov, and Mingfei Li. Oblivious RAM with $O((\log N)^3)$ worst-case cost. In *ASIACRYPT*, 2011. 11, 12
- [68] Emil Stefanov and Elaine Shi. Multi-cloud oblivious storage. In *CCS*, 2013. 11

- [69] Emil Stefanov and Elaine Shi. Oblivstore: High performance oblivious cloud storage. In *IEEE Symposium on Security and Privacy (S & P)*, 2013. 11
- [70] Emil Stefanov, Marten van Dijk, Elaine Shi, Christopher Fletcher, Ling Ren, Xiangyao Yu, and Srinivas Devadas. Path ORAM: an extremely simple oblivious ram protocol. In *In CCS*, 2013. 11
- [71] W. A. Stein et al. *Sage Mathematics Software*. The Sage Development Team, 2011. <http://www.sagemath.org>. 21
- [72] Marten van Dijk, Craig Gentry, Shai Halevi, and Vinod Vaikuntanathan. Fully homomorphic encryption over the integers. In Henri Gilbert, editor, *Advances in Cryptology - EUROCRYPT 2010, 29th Annual International Conference on the Theory and Applications of Cryptographic Techniques, French Riviera, May 30 - June 3, 2010. Proceedings*, volume 6110 of *Lecture Notes in Computer Science*, pages 24–43. Springer, 2010. 21
- [73] Xiao Wang, Hubert Chen, Yan Huang, Abhi Shelat, and Elaine Shi. Oblivious ram for secure computation. In *ACM CCS 2014*, 2014. 26
- [74] Peter Williams and Radu Sion. Usable PIR. In *NDSS*, 2008. 11
- [75] Peter Williams and Radu Sion. Round-optimal access privacy on outsourced storage. In *CCS*, 2012. 11
- [76] Peter Williams, Radu Sion, and Bogdan Carbutar. Building castles out of mud: Practical access pattern privacy and correctness on untrusted storage. In *CCS*, 2008. 11

A LIST OF PAPERS RESULTING FROM PROJECT

All papers published under this project were Contracted Fundamental Research (CFR) and did not require DISTAR public release approval.

A.1 In preparation or submission

1. Susan Hohenberger, Venkata Koppula, Brent Waters. Adaptively Secure Puncturable Pseudorandom Functions in the Standard Model. *In submission*.
2. Joseph Ayo Akinyele, Matthew Green, Matthew Pagano and Avi Rubin. CloudSourcing Cryptography: Automating the Outsourcing of Cryptographic Algorithms, *In preparation for resubmission*.

A.2 In print or to appear

1. Huijia Lin and Rafael Pass. Constant-round Non-malleable Commitments from Any One-way Function. *To appear in Journal of the ACM*.
2. Kai-min Chung, Huijia Lin, Rafael Pass. Constant-round Concurrent Zero-knowledge from Indistinguishability Obfuscation. *Crypto 2015*.
3. Elette Boyle, Kai-min Chung, and Rafael Pass. Large-Scale Multi-party Computation. *Crypto 2015*.
4. Huijia Lin and Rafael Pass. Succinct Garbling Schemes and Application. Manuscript (To appear in *STOC 2015* in a merged version.)
5. Jae Hyun Ahn, Dan Boneh, Jan Camenisch, Susan Hohenberger, abhi shelat and Brent Waters. Computing on Authenticated Data. *Journal of Cryptology* 28(2): 351-395, 2015.
6. Susan Hohenberger, Venkata Koppula, Brent Waters. Universal Signature Aggregators. *Eurocrypt 2015*.
7. Joseph Ayo Akinyele, Matthew Green, Susan Hohenberger and Matthew Pagano. Machine-Generated Algorithms, Proofs and Software for Batch Verification. *Journal of Computer Security* 22(6): 867-912, 2014.
8. E. Ben-Sasson, A. Chiesa, C. Garman, M. Green, I. Miers, E. Tromer, M. Virza. "Zerocash: Practical Decentralized Anonymous E-Cash from Bitcoin", *IEEE Security and Privacy* 2014.
9. Kai-min Chung and Rafael Pass. Parallel Repetition for Interactive Arguments. *SIGACT News* 2014.

10. C. Garman, M. Green, I. Miers, A. Rubin. "Rational Zero: Economic Security for Zerocoin with Everlasting Anonymity", *First Workshop on Bitcoin Research 2014*.
11. Susan Hohenberger, Amit Sahai, Brent Waters. "Replacing a Random Oracle: Full Domain Hash from Indistinguishability Obfuscation", *Eurocrypt 2014*.
12. Elette Boyle, Kai-min Chung, Rafael Pass, "On Extractability Obfuscation", *TCC 2014*.
13. Kai-min Chung, Huijia Lin, Rafael Pass, "Constant-round Concurrent Zero-knowledge from P-certificates", *FOCS 2013*.
14. Ran Canetti, Huijia Lin, Rafael Pass, "From Unprovability to Composable Security", *FOCS 2013*.
15. Huijia Lin and Rafael Pass, "Black-box Constructions of Composable Protocols", *CRYPTO 2013*.
16. K. Chung, R. Pass and S. Telang, "Knowledge-Preserving Interactive Coding", *FOCS 2013*.
17. K. Chung, R. Ostrovsky, R. Pass and I. Visconti, "Simultaneous Resettability From One-way Functions", *FOCS 2013*.
18. Susan Hohenberger and Brent Waters. "Online/Offline Attribute-Based Encryption", *PKC 2014*
19. Christina Garman, Matthew Green and Ian Miers. "Decentralized Anonymous Credentials", *NDSS 2014*
20. Steven Myers, Mone Sergi, abhi shelat, Blackbox Construction of A More than Non-Malleable CCA1 Encryption Scheme from Plaintext Awareness, *J. of Comp. Sec.*, Vol 21, No 5, pp 721–748.
21. Nathaniel Husted, Steven Myers, Emergent Properties & Security: The Complexity of Security as a Science, New Security Paradigms Workshop 2014.
22. Nathaniel Husted, Steven Myers, Paul Grubbs, abhi Shelat, "GPU and CPU Parallelization of Secure Two-Party Computation?", *ACSAC'13*
23. abhi shelat and Chih-hao Shen, "Faster two-party secure computation with minimal assumptions." *ACM CCS'2013*
24. Joseph Ayo Akinyele, Matthew Green, Susan Hohenberger. "Using SMT Solvers to Automate Design Tasks for Encryption and Signature Schemes", *ACM CCS 2013*
25. Susan Hohenberger, Amit Sahai, Brent Waters. Full Domain Hash from (Leveled) Multilinear Maps and Identity-Based Aggregate Signatures. *CRYPTO'13*.

26. Ian Miers, Christina Garman, Matthew Green, Aviel Rubin. Zerocoin: Anonymous Distributed e-Cash from Bitcoin. *IEEE Security and Privacy*'13.
27. "PCF: A Portable Circuit Format For Scalable Two-Party Secure Computation.", Kreuter, shelat, Mood, Butler *Accepted to USENIX SECURITY*'13
28. "Black-Box Proof of Knowledge of Plaintext and Multiparty Computation with Low Communication Overhead" Steven Myers, Mona Sergi, abhi shelat. *TCC* 2013.
29. Eleanor Birrell, Kai-min Chung, Rafael Pass, Sidharth Telang, "Randomness-Dependent Message Security", *TCC*'12
30. Rafael Pass, "Unprovable Security of Perfect NIZK and Non-interactive Non-malleable Commitments", *TCC*'13
31. Per Austrin, Johan Håstad, Rafael Pass, "On the Power of Many 1-Bit Provers". *Innovations in Theoretical Computer Science Conference*'13
32. Kai-min Chung, Huijia Lin, Mohammad Mahmoody, Rafael Pass, "On The Power of Non-uniform Proofs of Security". *Innovations in Theoretical Computer Science Conference*'13
33. Joseph Ayo Akinyele, Matthew Green, Susan Hohenberger and Matthew Pagano. Machine-Generated Algorithms, Proofs and Software for the Batch Verification of Digital Signature Schemes. *In ACM CCS*'12
34. Huijia Lin, Rafael Pass and Muthu Venkitasubramaniam, "UC from OT". *To appear in AsiaCrypt*'12.
35. Huijia Lin and Rafael Pass, "Blackbox Construction of Composable Protocols" *In CRYPTO*'12
36. Mohammad Mahmoody and Rafael Pass, "The Curious Case of Non-interactive Commitments." *In CRYPTO*'12
37. Kai-min Chung, Rafael Pass, Dustin Tseng, "The Knowledge Tightness of Parallel Zero-Knowledge". *TCC* 2012.
38. Ben Kreuter, Chih-hao Shen, and abhi shelat. "Towards Billion-gate Secure Two-party Computation." *USENIX Security*'12
39. Susan Hohenberger, Allison Lewko and Brent Waters. Detecting Dangerous Queries: A New Approach for Chosen Ciphertext Security. *EUROCRYPT*, 2012.
40. David Cash, Matthew Green and Susan Hohenberger. New Definitions and Separations for Circular Security. *Public Key Cryptography*, 2012.

41. Jae Hyun Ahn, Dan Boneh, Jan Camenisch, Susan Hohenberger, abhi shelat and Brent Waters. Computing on Authenticated Data. *TCC*, 2012.
42. Kai-Min Chung and Rafael Pass. The Randomness Complexity of Parallel Repetition. *FOCS*, 2011.
43. Jan Camenisch, Susan Hohenberger and Michael Ostergaard Pedersen. Batch Verification of Short Signatures. *Journal of Cryptology*, 25(4): 723-747, 2012.
44. Matthew Green, Susan Hohenberger and Brent Waters. Outsourcing the Decryption of ABE Ciphertexts. *USENIX Security*, 2011.

Constant Round Concurrent Zero-knowledge from IO

Kai-Min Chung

Huijia Lin*

Rafael Pass[†]

December 8, 2014

Abstract

We present a constant-round concurrent zero-knowledge protocol for NP. Our protocol relies on the existence of families of collision-resistant hash functions, one-way permutations, and indistinguishability obfuscators for $\mathbf{P}/poly$ (with slightly super-polynomial security).

*University of California, Santa Barbara, rachel.lin@cs.ucsb.edu.

[†]Cornell University, {rafael,sidtelang}@cs.cornell.edu. Work supported in part by a Alfred P. Sloan Fellowship, Microsoft New Faculty Fellowship, NSF Award CNS-1217821, NSF CAREER Award CCF-0746990, NSF Award CCF-1214844, AFOSR YIP Award FA9550-10-1-0093, and DARPA and AFRL under contract FA8750-11-2-0211. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the Defense Advanced Research Projects Agency or the US Government.

1 Introduction

Zero-knowledge ($\mathcal{ZK}$) interactive proofs [?] are paradoxical constructs that allow one player (called the Prover) to convince another player (called the Verifier) of the validity of a mathematical statement $x \in L$, while providing *zero additional knowledge* to the Verifier. Beyond being fascinating in their own right, $\mathcal{ZK}$ proofs have numerous cryptographic applications and are one of the most fundamental cryptographic building blocks.

The notion of concurrent zero knowledge, first introduced and achieved in the paper by Dwork, Naor and Sahai [?], considers the execution of zero-knowledge proofs in an asynchronous and concurrent setting. More precisely, we consider a single adversary mounting a coordinated attack by acting as a verifier in many concurrent executions (called sessions). Concurrent $\mathcal{ZK}$ proofs are significantly harder to construct and analyze. Since the original protocol by Dwork, Naor and Sahai (which relied on so called “timing assumptions”), various other concurrent $\mathcal{ZK}$ protocols have been obtained based on different set-up assumptions (e.g., [?, ?, ?, ?, ?, ?]), or in alternative models (e.g., super-polynomial-time simulation [?, ?]).

In the standard model, without set-up assumptions (the focus of our work,) Canetti, Kilian, Petrank and Rosen [?] (building on earlier works by [?, ?]) show that concurrent $\mathcal{ZK}$ proofs for non-trivial languages, with “black-box” simulators, require at least $\tilde{\Omega}(\log n)$ number of communication rounds. Richardson and Kilian [?] constructed the first concurrent $\mathcal{ZK}$ argument in the standard model without any extra set-up assumptions. Their protocol, which uses a black-box simulator, requires $O(n^\epsilon)$ number of rounds. The round-complexity was later improved in the work of Kilian and Petrank (KP) [?] to $\tilde{O}(\log^2 n)$ round. More recent work by Prabhakaran, Rosen and Sahai [?] improves the analysis of the KP simulator, achieving an essentially optimal, w.r.t. black-box simulation, round-complexity of $\tilde{O}(\log n)$; see also [?] for an (arguably) simplified and generalized analysis.

The central open problem in the area is whether a *constant-round* concurrent $\mathcal{ZK}$ protocol (for a non-trivial language) can be obtained. Note that it could very well be the case that all “classic” zero-knowledge protocols already are concurrent zero-knowledge; thus, simply assuming that those protocols are concurrent zero-knowledge yields an assumption under which constant-round concurrent zero-knowledge (trivially) exists—in essence, we are assuming that for every attacker a simulator exists. Furthermore, as shown in [?] (and informally discussed in [?]) under various “extractability” assumptions of the knowledge-of-exponent type [?, ?, ?], constant-round concurrent zero-knowledge is easy to construct. But such extractability assumptions also simply assume that for every attacker, a simulator (in essence, “the extractor” guaranteed by the extractability assumption) exists. In particular, an explicit construction of the concurrent zero-knowledge simulator is not provided—it is simply assumed that one exists. For some applications of zero-knowledge such as *deniability* (see e.g., [?, ?]), having an explicit simulator is crucial. Rather, we are here concerned with the question of whether constant-round concurrent zero-knowledge, *with an explicit simulator* exists.

1.1 Towards Constant-round Concurrent Zero-Knowledge

Recently, the authors [?] provided a first construction a constant-round concurrent zero-knowledge protocol with an explicit simulator, based on a new cryptographic hardness assumption—the existence of so-called **P-certificates**, roughly speaking, succinct non-interactive arguments for languages in **P**. An issue with their protocol, however, is that soundness only holds for *uniform* polynomial-

time attackers (as opposed to non-uniform ones).¹ Additionally, whereas the existence $\mathbf{P}$ -certificates is a falsifiable assumption [?, ?] it is unclear whether the existence of $\mathbf{P}$ -certificates itself can be based on some more natural hardness assumption.

A very recent elegant work by Pandey, Prabhakaran and Sahai [?] takes a different approach and instead demonstrates the existence of constant-round concurrent zero-knowledge protocol with an explicit simulator based on the existence of *differing-input obfuscation* (**diO**) for (restricted classes of) $\mathbf{P}/poly$ [?, ?, ?]. Whereas the assumption that a particular scheme is a **diO** is an “extractability” assumption (similar in flavor to knowledge-of-exponent type [?, ?, ?] assumptions), the intriguing part of the scheme of Pandey et al [?] is that the extractability assumption is only used to prove *soundness* of the protocol; concurrent zero-knowledge is proved in the “standard” model, through providing an explicit simulator. Nevertheless, **diO** is a strong and subtle assumption—indeed, as shown by recent work, unless we restricting the class of programs for which **diO** should hold, we may end up with a notion that is unsatisfiable [?, ?, ?]). Furthermore, there are currently no known approaches for basing **diO** on more “natural” (or in fact *any*) hardness (as opposed to extractability) assumption.

1.2 Our Results

In this paper, we combine the above-mentioned two approaches. Very roughly speaking, we will use obfuscation to obtain a variant of the notion of a $\mathbf{P}$ -certificate, and we next show that this variant still suffices to obtain constant-round concurrent zero-knowledge (where the soundness conditions holds also against non-uniform PPT attackers). More importantly, rather than using **diO**, we are able to use *indistinguishability obfuscation* (**iO**) [?, ?]. Following the groundbreaking work of Garg et al [?], there are now several candidate constructions of **iO** that can be based on hardness assumptions on (approximate) multilinear maps [?, ?].

Theorem. *Assume the existence of indistinguishability obfuscation for $\mathbf{P}/poly$ (with slightly super-polynomial security), one-way permutations and collision-resistant hashfunction. Then there exists a constant-round concurrent zero-knowledge argument for NP.*

In more details, our approach proceeds in the following steps:

- We first observe that a warm-up case considered in [?]¹—which shows the existence of constant-round concurrent zero-knowledge based on, so-called, *unique $\mathbf{P}$ -certificates* (that is, $\mathbf{P}$ -certificates for which there exists at most one accepting certificate for each statement) directly generalizes also to unique $\mathbf{P}$ -certificates in the Common *Random* String model (a.k.a. the Uniform Random String model (URS)) satisfying an *adaptive soundness* property (where the statement to be proved can be selected after the URS).
- We next show that by appropriately modifying the protocol, we can handle also unique $\mathbf{P}$ -certificates in the URS model satisfying even just a “static” soundness condition (where the statement needs to be selected before the URS is picked), and additionally also unique $\mathbf{P}$ -certificates (with static soundness) in the Common Reference String (CRS) model, where the reference string no longer is required to be uniform. Unique $\mathbf{P}$ -certificates in the CRS model can be constructed based on the existence of **diO** for $\mathbf{P}/poly$ [?], and as such this preliminary step already implies the result of [?] in a modular way (but with worse concrete round complexity).

¹Or holds against non-uniform attackers under significantly stronger, and no longer falsifiable, assumptions.

- We next consider a more relaxed variant of unique **P**-certificates in the CRS model—which we refer to as *delegetable unique P-certificates*—where the CRS is allowed to be *statement dependent* but only a “small” (in particular, independent of the statement length) part of the CRS generation requires using secret coins. By relying on **iO** for **P/poly**, we next show that the protocol can be generalized to work also with such delegetable unique **P**-certificates.
- We finally leverage recent results on delegation of computation based on **iO** from [?, ?, ?] and show that the beautiful protocol of Koppula, Lewko and Waters [?] can be modified into a delegetable unique **P**-certificate.

1.3 Outline of Our Techniques

We here provide a detailed outline of our techniques. As mentioned, our construction heavily relies on a “warm-up” case of the construction of [?], which we start by recalling (closely following the description in [?]). The starting point of the construction of [?] is the construction is Barak’s [?] non-black-box zero-knowledge argument for NP. We start by very briefly recalling the ideas behind his protocol (following a slight variant of this protocol due to [?]).

Barak’s protocol Roughly speaking, on common input 1^n and $x \in \{0, 1\}^{\text{poly}(n)}$, the Prover **P** and Verifier V , proceed in two stages. In Stage 1, P starts by sending a computationally-binding commitment $c \in \{0, 1\}^n$ to 0^n ; V next sends a “challenge” $r \in \{0, 1\}^{2n}$. In Stage 2, P shows (using a witness indistinguishable argument of knowledge) that either x is true, or there exists a “short” string $\sigma \in \{0, 1\}^n$ such that c is a commitment to a program M such that $M(\sigma) = r$.²

Soundness follows from the fact that even if a malicious prover P^* tries to commit to some program M (instead of committing to 0^n), with high probability, the string r sent by V will be different from $M(\sigma)$ for every string $\sigma \in \{0, 1\}^n$. To prove ZK, consider the non-black-box simulator S that commits to the code of the malicious verifier V^* ; note that by definition it thus holds that $M(c) = r$, and the simulator can use $\sigma = c$ as a “fake” witness in the final proof. To formalize this approach, the witness indistinguishable argument in Stage 2 must actually be a witness indistinguishable *universal argument* (WIUA) [?, ?] since the statement that c is a commitment to a program M of *arbitrary* polynomial-size, and that $M(c) = r$ within some *arbitrary* polynomial time, is not in NP.

Now, let us consider concurrent composition. That is, we need to simulate the view of a verifier that starts $m = \text{poly}(n)$ concurrent executions of the protocol. The above simulator no longer works in this setting: the problem is that the verifier’s code is now a function of *all* the prover messages sent in different executions. (Note that if we increase the length of r we can handle a bounded number of concurrent executions, by simply letting σ include all these messages).

So, if the simulator could commit not only to the code of V^* , but also to a program M that generates all other prover messages, then we would seemingly be done. And at first sight, this doesn’t seem impossible: since the simulator S is actually the one generating all the prover messages, why don’t we just let M be an appropriate combination of S and V^* ? This idea can indeed be implemented [?, ?], but there is a serious issue: if the verifier “nests” its concurrent executions, the running-time of the simulation quickly blows up exponentially—for instance, if we have three nested sessions, to simulate session 3 the simulator needs to generate a WIUA regarding the computation needed to generate a WIUA for session 2 which in turn is regarding the generation of the WIUA of

²We require that C is a commitment scheme allowing the committer to commit to an arbitrarily long string $m \in \{0, 1\}^*$. Any commitment scheme for fixed-length messages can easily be modified to handle arbitrarily long messages by asking the committer to first hash down m using a collision-resistant hash function h chosen by the receiver, and next commit to $h(m)$.

session 1 (so even if there is just a constant overhead in generating a WIUA, we can handle at most $\log n$ nested sessions).

Unique P-certificates to The Rescue: The “Warm-Up” Case from [?] As shown in [?], the blow-up in the running-time can be prevented using Unique **P**-certificates. Roughly speaking, we say that (P, V) is a **P**-certificate system if (P, V) is a non-interactive proof system (i.e., the prover send a single message to the verifier, who either accepts or rejects) allowing an efficient prover to convince the verifier of the validity of any *deterministic polynomial-time computation* $M(x) = y$ using a “certificate” of some *fixed* polynomial length (independent of the size and the running-time of M) whose validity the verifier can check in some fixed polynomial time (independent of the running-time of M). The **P**-certificate system is *unique* if there exists at most one accepted proof for any statement.

The protocol proceeds just as Barak’s protocol except that Stage 2 is modified as follows: instead of having P prove (using a WIUA) that either x is true, or there exists a “short” string $\sigma \in \{0, 1\}^{2n}$ such that c is a commitment to a program M such that $M(\sigma) = r$, we now ask P to use a WIUA to prove that either x is true, or

- **commitment consistency:** c is a commitment to a program M_1 , and
 - **input certification:** there exists a vector $\lambda = ((1, \pi_1), (2, \pi_2), \dots)$ and a vector of messages $\vec{m}$ such that π_i certifies that $M_1(\lambda_{<j})$ outputs m_j in its j ’th communication round, where $\lambda_{<j} = ((1, \pi_1), \dots, (j-1, \pi_{j-1}))$, and
 - **prediction correctness:** there exists a **P**-certificate π of length n demonstrating that $M_1(\lambda) = r$.

Soundness of the modified protocol, roughly speaking, follows since by the unique certificate property, for every program M_1 it inductively follows that for every j , m_j is uniquely defined, and thus also the *unique* (accepting) certificate π_j certifying $M_1(\lambda_{<j}) = m_j$; it follows that M_1 determines a unique vector λ that passes the input certification conditions, and thus there exists a single r that make M_1 also pass the prediction correctness conditions. Note that we here inherently rely on the fact that the **P**-certificate is unique to argue that the sequence λ is uniquely defined. (Technically, we here need to rely on a **P**-certificate that is sound for slightly super-polynomial-time as there is no a-priori polynomial bound on the running-time of M_1 , nor the length of λ .)

To prove zero-knowledge, roughly speaking, our simulator will attempt to commit to its own code in a way that prevents a blow-up in the running-time. Recall that the main reason that we had a blow-up in the running-time of the simulator was that the generation of the WIUA is expensive. Observe that in the new protocol, the only expensive part of the generation of the WIUA is the generation of the **P**-certificates π ; the rest of the computation has *a-priori* bounded complexity (depending only on the size and running-time of V^*). To take advantage of this observation, we thus have the simulator only commit to a program that generates prover messages (in identically the same way as the actual simulator), but getting certificates $\vec{\pi}$ as input.

In more detail, to describe the actual simulator S , let us first describe two “helper” simulators S_1, S_2 . S_1 is an interactive machine that simulates prover messages in a “right” interaction with V^* . Additionally, S_1 is expecting some “external” messages on the “left”—looking forward, these “left” messages will later be certificates provided by S_2 . See Figure 1 for an illustration of the communication patterns between S_1, S_2 and V^* .

S_1 proceeds as follows in the right interaction. In Stage 1 of every session i , S_1 first commits to a machine $\tilde{S}_1(j', \tau)$ that emulates an interaction between S_1 and V^* , feeding S_1 input τ as messages on the left, and finally $\tilde{S}_1$ outputs the verifier message in the j ’th communication round in the

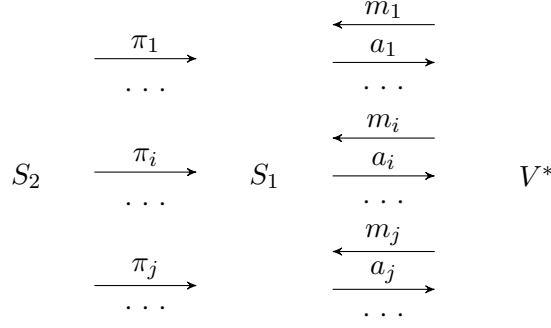


Figure 1: Simulation using **P**-certificates.

right interaction with V^* . (Formalizing what it means for S_1 to commit to $\tilde{S}_1$ is not entirely trivial since the definition of $\tilde{S}_1$ depends on S_1 ; we refer the reader to the formal proof for a description of how this circularity is broken.³ S_1 next simulates Stage 2 by checking if it has received a message (j, π_j) in the left interaction, where j is the communication round (in the right interaction with V^*) where the verifier sends its random challenge and expects to receive the first message of Stage 2; if so, it uses $M_1 = \tilde{S}_1$ (and the randomness it used to commit to it), j and σ being the list of messages received by S_1 in the left interaction, as a "fake" witness to complete Stage 2.

The job of S_2 is to provide **P**-certificates π_j for S_1 allowing S_1 to complete its simulation. S_2 emulates the interaction between S_1 and V^* , and additionally, at each communication round j , S_2 feeds S_1 a message (j, π_j) where π_j is a **P**-certificate showing that $\tilde{S}_1(j, \sigma_{<j}) = r_j$, where $\sigma_{<j}$ is the list of messages already generated by S_2 , and r_j is the verifier message in the j 'th communication round. Finally, S_2 outputs its view of the full interaction.

The actual simulator S just runs S_2 and recovers from the view of S_2 the view of V^* and outputs it. Note that since S_1 has polynomial running-time, generating each certificate about $\tilde{S}_1$ (which is just about an interaction between S_1 and V^*) also takes polynomial time. As such S_2 can also be implemented in polynomial time and thus also S .

Finally, indistinguishability of this simulation, roughly speaking, follow from the hiding property of the commitment in Stage 1, and the WI property of the WIUA in Stage 2. (There is another circularity issue that arises in formalizing this—as S_1 in essence needs to commit to its own randomness—but it can be dealt with as shown in [?]; we here omit the details as they are not important for our modifications to the protocol.)

Generalizing to Unique P-certificates in CRS model The key technical contribution in [?] was to generalize the above approach to deal also with "non-unique" **P**-certificates. Here we instead aim to generalize the above approach to work with **P**-certificates in the CRS model, but still relying on the uniqueness property.

Let us first note that if we had access to unique **P**-certificate in the URS (i.e., the uniform reference string) model satisfying an *adaptive soundness* property (where the statement to be proved can be selected after the URS, then above-mentioned protocol directly generalized to work with them (as opposed to using unique **P**-certificates in the "plain" model) by simply having the Verifier send the URS along with its first message of the protocol.⁴

We next note that the protocol can be further generalized to handle also unique **P**-certificates in the URS model satisfying even just a *static soundness* condition (where the statement needs to

³Roughly speaking, we let S_1 take the description of a machine M as input, and we then run S_1 on input $M = S_1$.

⁴To make this work, we need to rely on **P**-certificates in the URS model with perfect completeness.

be selected before the URS is picked) by proceeding as follows:

- We add a Stage 1.5 to the protocol where the Prover is asked to provide a commitment c_2 to 0^n and then asked to provide a WIUARG that either $x \in L$ or c_2 is a commitment to a “well-formed” statement (but not that the statement is true) for the **P**-certificate in use in Stage 2.
- Stage 2 of the protocol is then modified to first have the Verifier send the URS for the **P**-certificate, and then requiring that the prover uses a **P**-certificate for the statement committed to in c_2 . In other words, we require the Prover to commit in advance, and prove knowledge of, the statement to be used in the **P**-certificate and thus static soundness suffices.

Additionally, this approach generalizes also to deal with unique **P**-certificates in the Common Reference String (CRS) model (where the reference string no longer needs to be uniform), by having the Verifier provide a zero-knowledge proof that the CRS was well-formed.⁵

Generalizing to Delegetable P-certificates The notion of a **P**-certificate in the CRS model requires that the same CRS can be used to prove *any* statement q of any (polynomially-related) length. We will now consider a weaker notion of a **P**-certificate in the CRS model, where the CRS is “statement-dependent”—that is, the CRS is generated as a function of the statement q to be proved—in essence, such **P**-certificates can be viewed as specific instances of a *two-round* delegation protocol. But whereas the CRS may depend on the statement, we still restrict it in several important ways:

- As before, the length of the CRS is “short” (independent of the length of the statement q).
- Additionally, only a “small” part of the generation procedure relies on secret coins. More precisely, the CRS generation procedure proceeds in three steps: 1) first, secret coins are used to generate a public parameter PP and a secret parameter K (this is done independently of the statement q), 2) next, only PP is used to *deterministically* process the statement q into a “short” digest d (independent of the length of q), and 3) the digest d and the secret parameter K is efficiently processed to finally generate the CRS (independent of the length of q). To emphasize, only step 2 requires work that is proportional to the length of q , but this work only requires public information.

We now generalize the above approach to also work with delegetable unique **P**-certificates.

- Instead of having the Verifier send the CRS in the clear (which it cannot compute as it does not know the statement q on which it will be run), it simply runs part 1 of the CRS generation procedure to generate PP and K and sends just the public-parameter PP to the Prover.
- The Prover is then asked to provide a third commitment c_3 to 0^n and provide a WIUARG that either $x \in L$ or c_3 is a correctly computed digest d (w.r.t., PP) to the statement q committed to in c_2 . (In essence, the Verifier is delegating the computation of d to the Prover.)
- Next, the Verifier sends an indistinguishability obfuscation $\tilde{\Pi} = \mathbf{iO}(\Pi)$ of a program Π that on input a decommitment (d', r') to c_3 processes d' and K into a CRS ρ and outputs it. (The reason that the Verifier cannot generate ρ in the clear is that digest d cannot be sent to the Verifier in the clear; recall that the honest prover will never compute any such digest,

⁵Again, we here rely on **P**-certificates in the CRS model with perfect completeness.

it is meant to commit to 0^n and prove that $x \in L$.) Additionally, the verifier gives a zero-knowledge proof that the obfuscation is correctly computed (and using the same random coins that were used to generate PP).

- Then, the Prover provides a commitment c_4 to 0^n and provides a WI proof of knowledge that $x \in L$ or c_4 is a commitment to a CRS ρ computed by applying the obfuscated code $\tilde{\Pi}$ to a proper decommitment of c_3 .
- Finally, in Stage 2 of the protocol, we require the Prover to provide **P**-certificates w.r.t to the CRS ρ committed to in c_4 .

Note that if c_3 is perfectly binding, then by **iO** security of the obfuscation, we can replace Π with a program that has the CRS ρ hardcoded and does not depend on K , and this suffices for arguing that soundness of the protocol still holds. On the other hand, the simulation can proceed just as before except that now it uses the obfuscated code $\tilde{\Pi}$ to generate the CRS ρ and commit to it in c_4 .

Realizing Delegetable Unique P-Certificates We finally leverage recent results on delegation of computation based on **iO** for circuits from [?, ?, ?] and show that the beautiful protocol of Koppula, Lewko and Waters [?] can be massaged (and slightly modified) into a delegetable unique **P**-certificate.

Let us point out that, just as [?], our protocol requires the use of **P**-certificates that satisfy a slightly strong soundness condition—namely, we require soundness to hold against circuits of size $T(\cdot)$ where $T(\cdot)$ is some “nice” (slightly) super-polynomial function (e.g., $T(n) = n^{\log \log \log n}$). To achieve such (delegetable) **P**-certificates, we thus need to rely on **iO** for **P**/*poly* secure against $T(\cdot)$ -size circuits.

2 Introduction

3 Preliminaries

4 P-certificates

We consider the following canonical languages for **P**: for every constant $c \in N$, let $L_c = \{(M, x, y) : M(x) = y \text{ within } |x|^c \text{ steps}\}$. Let $T_M(x)$ denotes the running time of M on input x .

Definition 1 (Two-Message **P**-certificate). *A tuple of probabilistic interactive Turing machines, $(\text{Gen}, \text{P}_{\text{cert}}, \text{V}_{\text{cert}})$, is a (Two-Message) **P**-certificate system if there exist polynomials l_{CRS} , l_{π} , and the following holds:*

Syntax and Efficiency: *For every $c \in N$, every $q = (M, x, y) \in L_c$, and every $k \in N$, the verification of the statement proceed as follows:*

CRS GENERATION: $\text{CRS} \xleftarrow{\$} \text{Gen}(1^k, c, q)$, where Gen runs in time $\text{poly}(k, |q|)$.

PROOF GENERATION: $\pi \xleftarrow{\$} \text{P}_{\text{cert}}(1^k, c, q, \text{CRS})$, where P_{cert} runs in time $\text{poly}(k, |x|, \min(T_M(x), |x|^c))$ with $T_M(x) \leq |x|^c$ the running time of M on input x . The length of the proof π is bounded by $l_{\pi}(k)$.

PROOF VERIFICATION: $b = \text{V}_{\text{cert}}(1^k, \text{CRS}, \pi)$, where V_{cert} runs in time $\text{poly}(k, |\text{CRS}|)$.

(Perfect) Completeness: For every $c, d \in N$, there exists a negligible function μ such that for every $k \in N$ and every $q = (M, x, y) \in L_c$ such that $|q| \leq k^d$, the probability that in the above execution V_{cert} outputs 1 is 1.

Definition 2 (Selective Strong Soundness of **P**-certificate). We say that a **P**-certificate system $(\text{Gen}, P_{\text{cert}}, V_{\text{cert}})$ is (selectively) strong sound if the following holds:

- Strong Soundness: There exists some “nice” super-polynomial function⁶ $T(k) \in k^{\omega(1)}$ and some “nice” super-constant function⁷ $C(\cdot) \in \omega(1)$ such that for every probabilistic algorithm P^* with running-time bounded by $T(\cdot)$, there exists a negligible function μ , such that, for every $k \in N$, $c \leq C(k)$,

$$\Pr \left[\begin{array}{ccc} (q, \text{st}) & \xleftarrow{\$} & P^*(1^k, c) \\ \text{CRS} & \xleftarrow{\$} & \text{Gen}(1^k, c, q) \\ \pi & \xleftarrow{\$} & P^*(\text{st}, \text{CRS}) \end{array} : V_{\text{cert}}(1^k, \text{CRS}, \pi) = 1 \wedge q \notin L_c \right] \leq \mu(k)$$

Definition 3 (Uniqueness of **P**-certificate). We say that a **P**-certificate system $(\text{Gen}, P_{\text{cert}}, V_{\text{cert}})$ is unique if for every $k \in N$, every string $\text{CRS} \in \{0, 1\}^*$, there exists at most one string $\pi \in \{0, 1\}^*$, such that $V_{\text{cert}}(1^k, \text{CRS}, \pi) = 1$.

Rachel:

[This is a very strong uniqueness requirement, in particular, any CRS string (even ones that are not in the support of Gen) has at most one matching proof.]

Delegatable CRS Generation

Definition 4 (Delegatable CRS Generation). We say that a (two-message) **P**-certificate $(\text{Gen}, P_{\text{cert}}, V_{\text{cert}})$ has delegatable CRS generation if the CRS generation algorithm Gen consists of three subroutines ($\text{Setup}, \text{PreGen}, \text{CRSGen}$), and there are polynomials l_d and l_κ , such that, the following holds:

Delegatable CRS Generation: $\text{Gen}(1^k, c, q)$ proceeds in the following three steps:

1. Generate parameters: $(PP, K) \xleftarrow{\$} \text{Setup}(1^k, c)$, where Setup runs in time $\text{poly}(k)$. We call PP the public parameter and K the key.
2. (Public) statement processing: $d \xleftarrow{\$} \text{PreGen}(PP, q)$, where PreGen runs in time $\text{poly}(k, |q|)$, and the length of d is bounded by $l_d(k)$. We call d the digest of the statement.
3. (Private) CRS generation: $\kappa \xleftarrow{\$} \text{CRSGen}(PP, K, d)$, where CRSGen runs in time $\text{poly}(k)$, and the length of κ is bounded by $l_\kappa(k)$.

Finally, Gen outputs $\text{CRS} = (PP, \kappa)$.

The reason that we say such a CRS generation procedure is delegatable is because the only part of computation that depends on the statement is the statement processing step; all other steps runs in time a fixed polynomial in the security parameter. However, the statement processing step depends only on the public parameter and the statement; hence to ensure soundness, one only needs to ensure the correctness of this computation, without ensuring the “secrecy” of the computation. Therefore, we also call this step “public” statement processing.

⁶For instance, $T(n) = n^{\log \log \log n}$.

⁷For instance, $C(k) = \log \log \log n$.

4.1 Instantiation of P-certificates with Delegatable CRS Generation

Our instantiation relies on the “message hiding encoding” introduced in the recent work by Koppula, Lewko and Waters [?], as a step towards constructing (succinct) indistinguishability obfuscation for Turing machines. Roughly speaking, a message hiding encoding scheme proceeds as follows: Given any message msg (usually generated at random in applications), it transforms a Turing machine computation, M on input x (with time bound T), into an encoding enc , which when decoded yields msg if $M(x) = 1$ (in T steps) and $\perp$ otherwise; on the other hand, the security of the message hiding encoding guarantees that the encoding enc for a non-accepting computation (M, x) hides the message msg . Below, we recall their definition: Let $\Pi_M^T(x)$ denote the Turing machine that runs $M(x)$ for T steps and outputs 1 if the computation accepts and $\perp$ otherwise.

Definition 1 (Message Hiding Encoding [?]). *A message hiding encoding scheme MHE consists of two PPT algorithms (MHE.enc, MHE.dec) satisfying the following properties*

Syntax and Efficiency: *For any Turing machine M , input $\text{inp} \in \{0, 1\}^*$, message $\text{msg} \in \{0, 1\}^*$, time bound $T \in \mathbb{N}$, and security parameter $k \in \mathbb{N}$,*

1. *Encoding: The encoding algorithm $\text{MHE.enc}(1^k, M, T, \text{inp}, \text{msg})$ outputs an encoding enc , in time $\text{poly}(k, |M|, |\text{inp}|, |\text{msg}|, \log T)$ (independent of the running time of the computation.)*
2. *Decoding: The decoding algorithm $\text{MHE.dec}(1^k, M, \text{inp}, T, \text{enc})$ outputs a message msg or $\perp$, in time $\text{poly}(k, |M|, |\text{inp}|, \log T, \min(T_M(x), T))$, where $T_M(x)$ is the running time of M on input x .*

Correctness: *For any Turing machine M , input $\text{inp} \in \{0, 1\}^*$, message $\text{msg} \in \{0, 1\}^*$, time bound $T \in \mathbb{N}$, and security parameter $k \in \mathbb{N}$, if $\Pi_M^T(x) = 1$, then*

$$\text{MHE.dec}(1^k, M, \text{inp}, T, \text{MHE.enc}(1^k, M, T, \text{inp}, \text{msg})) = \text{msg}$$

Definition 2 (Message Hiding Property). *A message hiding encoding scheme MHE is secure if for every PPT adversary A^* , and polynomial Γ , there is a negligible function ε , such that, for every security parameter $k \in \mathbb{N}$, every messages $\text{msg}_0, \text{msg}_1 \in \{0, 1\}^k$, M of description size at most k , time bound $T \leq p(k)$, and input $\text{inp} \in \{0, 1\}^{p(k)}$, such that, $\Pi_M^T(\text{inp}) = 0$, it holds that,*

$$\Pr \left[\begin{array}{l} b \xleftarrow{\$} \{0, 1\} \\ (\text{st}, \text{msg}_0, \text{msg}_1, M, T, \text{inp}) \xleftarrow{\$} A^*(1^k) : \wedge T \leq \Gamma(k) \\ \text{enc} \xleftarrow{\$} \text{MHE.enc}(1^k, M, T, \text{inp}, \text{msg}_b) \quad \wedge A^*(\text{st}, \text{enc}) = b \end{array} \right] \leq 1/2 + \varepsilon(k)$$

Furthermore, MHE is super-polynomially secure if there exists a super-polynomial functions Γ' , such that the above condition holds for every Γ' -time adversary and function Γ' .

The message hiding encoding is similar to and can be viewed as a weakening of randomized encoding [?] in the following sense: The encoding enc for M, x with message msg , can also be viewed as an encoding for the augmented Turing machine $\tilde{M}(x, \text{msg})$ that outputs msg if $M(x) = 1$ and $\perp$ otherwise; while randomized encoding guarantees the privacy of the whole input (x, msg) , the message hiding encoding only guarantees privacy of a part of the input msg .

In [?], a construction of a message hiding encoding is provided assuming the existence of indistinguishability obfuscation for circuits and one-way function.

Theorem 1. *Assume the existence of an indistinguishability obfuscation for circuits, an injective PRG and an IND-CPA secure public-key encryption scheme (that are super-polynomially secure), there is a message hiding encoding scheme (that is super-polynomially secure).*

P-certificates from Message Hiding Encoding: It is known that randomized encoding (and its slightly enhanced variant of garbling schemes) can be used to ensure the correctness of a computation, as explored in many previous works, for example in [?, ?] and formalized in [?]. In fact, for ensuring correctness, it suffices to use a “message hiding encoding” as observed in [?]. Here, the message msg can be viewed as the correctness proof, and the message hiding property ensures that a prover can only obtain msg if the underlying computation is accepting, which implies computational soundness. This naturally suggests a two message proof system for P : Let $\text{Ver}(c, q)$ for $q = (M, x, y)$ be the universal verification algorithm that verifies if $M(x) = y$ in $|x|^c$ steps; it outputs 1 if so and 0 otherwise; it is easy to see that the run time of Ver is bounded by $\alpha|x|^c$ with a universal constant α .

CRS GENERATION $\text{Gen}(1^k, c, q)$: Sample $\pi \xleftarrow{\$} \{0, 1\}^k$ at random. Compute the message hiding encoding $\text{enc} \xleftarrow{\$} \text{MHE.enc}(1^k, M = \text{Ver}, T = \alpha|x|^c, \text{inp} = q, \text{msg} = \pi)$, with π as the message. Additionally compute $y = f(\pi)$ using an injective one-way function f . Outputs CRS string $\text{CRS} = (\text{enc}, y)$.

PROOF GENERATION $\text{P}_{\text{cert}}(1^k, c, q, \text{CRS})$: Parse $\text{CRS} = (\text{enc}, y)$. Decode $z = \text{MHE.dec}(1^k, \text{Ver}, |q|^c, q, \text{enc})$. If $f(z) = y$, output proof $\pi = z$; otherwise, output $\perp$.

PROOF VERIFICATION $\text{V}_{\text{cert}}(1^k, \text{CRS}, \pi)$: Parse $\text{CRS} = (\text{enc}, y)$. Accept if $f(\pi) = y$, and reject otherwise.

Efficiency: The proof verification algorithm V_{cert} runs in strict polynomial time. The complexity of the CRS and proof generation is determined by the complexity of the encoding and decoding algorithm of the message hiding encoding scheme: It follows from the efficiency of MHE.enc that Gen runs in time $\text{poly}(k, |\text{Ver}|, |q|, |\pi|, \log(\alpha|x|^c)) = \text{poly}(k, |q|)$, and from the efficiency of MHE.dec that P_{cert} runs in time $\text{poly}(k, |\text{Ver}|, |q|, |\pi|, \log(|q|^c), \min(t^*, \alpha|x|^c)) = \text{poly}(k, |q|, \min(t^*, |x|^c))$, where t^* is the running time of M on input x . Moreover, the length of the proof is exactly $|\pi| = k$. In summary, the above system satisfies the efficiency requirement of **P-certificates**.

Strong Soundness: It follows directly from standard techniques that the message hiding property of MHE implies that for any constant c , the above system is secure against any PPT cheating prover trying to prove a statically chosen false statement q w.r.t. language L_c . This is because, for a false statement, the computation $\text{Ver}(c, q)$ is not accepting. Thus, it follows from the message hiding property of MHE that, the honest encoding enc of Ver with input (c, q) and message π is indistinguishable from an encoding enc' of Ver , (c, q) and a different message, say, 0^n . Therefore, if a cheating prover can produce a valid proof for q when receiving an honest $\text{CRS} = (\text{enc}, f(\pi))$ with polynomial probability, it can still produce a valid proof when receiving $\text{CRS}' = (\text{enc}', f(\pi))$. Since a valid proof is π , the cheating prover violates the one-wayness of f . Thus soundness holds.

To obtain strong soundness, we rely on complexity leveling. Assume that MHE and the injective one-way function is super-polynomially secure w.r.t. to a super-polynomial function Γ . There must exist another super-polynomial function Γ' and a super-constant function β' , such that, $\Gamma'(k)^{\beta'(k)} \leq \Gamma(k)$ (for example, let Γ' be equal to $2^{\beta(k) \log k}$ for $\beta(k) = \omega(1)$; set $\beta'(k) = \beta(k)^{1/2}$ and $\Gamma'(k) = 2^{\beta'(k) \log k}$). It follows from the same argument that the above argument system is sound against all

Γ' -time cheating provers who chooses false statement q w.r.t. any language L_c for $c < \beta'(k)$. This implies that the system is strong sound.

Uniqueness: For any CRS string $\text{CRS}(\text{enc}, y)$, it follows from the injectiveness of the one-way function f , that there is at most one string π , such that, $\text{Ver}(1^k, \text{CRS}, \pi) = 1$, that is, $f(\pi) = y$.

Summarizing, we have,

Theorem 2. *Assume the existence of a message hiding encoding scheme and an injective one-way function (that are both super-polynomially secure), there is a (two-message) $\mathbf{P}$ -certificate system with (strong) soundness and uniqueness.*

Delegatable CRS Generation. The message hiding encoding scheme of [?] has certain special structure, such that, the resulting construction of $\mathbf{P}$ -certificates directly have delegatable CRS generation. The special property is that their encoding algorithm can be divided into three steps matching exactly the three steps in delegatable CRS generation:

- (i) First, it generates certain public parameters and a key, depending only on the security parameter k and the time bound T . (Namely, this step runs their **Setup-Acc** and **Steup-ltr** algorithms; let PP denote the output of these two algorithms and K is a randomly sampled puncturable PRF key).
- (ii) then, the input of the computation x is processed using the security parameter and public parameters to produce a digest of the input. (Namely, this step runs their **Write-Store**, **Prep-Write**, and **Update** algorithms iteratively with the input x and the public parameters PP , to compute a digest w of the input. Note that their input processing step also produces a processed input denoted as $store$, which in an overly simplified view, is similar to a Merkle Hash tree built with leaves x^8 ; and $store$ is also a part of the encoding. However, we notice that the rest of the encoding does not depend on $store$, and since it can be re-computed by the decoder given x and the public parameter, it can hence be omitted from the encoding.)
- (iii) finally, the encoding is produced depending only on the security parameter, the digest of the input, the public parameter, and the key. (Namely, this step runs the **Setup-Spl**, **Sign-Spl** using the PRF key K and the digest w , and then obufscates using IO a program that depends on the TM M , the time bound T , the public parameter PP and K .)

These three steps for generating an encoding corresponds exactly to the **Setup**, **PreGen** and **CRSGen** algorithms in a delegatable CRS generation, with the **CRSGen** additionally computes the image $y = f(\pi)$. Thus, combining Theorem 2 with the construction of message hiding encoding of [?], and noticing the special structure of its encoding algorithm, we have,

Corollary 1. *Assume the existence of an indistinguishability obfuscation for circuits, an injective PRG and an IND-CPA secure public-key encryption scheme (that are all super-polynomially secure), there is a (two-message) $\mathbf{P}$ -certificate system with (strong) soundness, uniqueness, and delegatable CRS generation.*

5 Our Protocol

We proceed to describe formally our protocol, (P, V) . The protocol relies on the following primitives:

⁸The actual computation of $store$ is much more complicated. In an over-simplified view, it is similar to a Merkle hash tree computed using a specially crafted hash function implemented using IO.

- A non-interactive *perfectly binding* commitment scheme com . We assume without loss of generality that com only needs n bits of randomness to commit to any n -bit string, (as it can always expand these n bits into a longer sequence using a PRG).
- A strong (two-message) $\mathbf{P}$ -certificate system $(\text{Gen}, \text{P}_{\text{cert}}, \text{V}_{\text{cert}})$ with delegatable CRS generation $\text{Gen} = (\text{Setup}, \text{PreGen}, \text{CRSGen})$. The strong soundness property is associated with parameter $T(\cdot)$ and $C(\cdot)$, where $T(\cdot)$ is a “nice” super-polynomial function and $C(\cdot)$ is a “nice” super-constant function. The uniqueness property ensures that for every string CRS , there exists at most one proof π that is accepted by $\text{V}_{\text{cert}}(1^k, \text{CRS}, \pi) = 1$. This allows us to define the following deterministic oracle $\mathcal{O}_{\text{V}_{\text{cert}}}^n$, which will be used in the CZK protocol later.

$$\mathcal{O}_{\text{V}_{\text{cert}}}^n(\text{CRS}) = \begin{cases} \pi & \text{If there exists unique } \pi \text{ s.t. } \text{V}_{\text{cert}}(1^n, \text{CRS}, \pi) = 1 \\ \perp & \text{otherwise} \end{cases}$$

We call $\mathcal{O}_{\text{V}_{\text{cert}}}^n$ the $\mathbf{P}$ -certificate oracle. Additionally, we consider a universal emulator Emulator^n that on input (P, x, O) emulates the execution of a *deterministic oracle machine* P on input x with oracle $\mathcal{O}_{\text{V}_{\text{cert}}}^n$ as follows: It parses O as an array; to answer the i^{th} query CRS_i from P , it checks whether O_i is the right answer from this CRS (i.e., $\text{V}_{\text{cert}}(1^n, \text{CRS}_i, O_i) = 1$); if so, it returns O_i to P ; otherwise, it aborts and outputs $\perp$. Finally, the emulator outputs the output of P .

For simplicity, we assume that the lengths of the CRS, the proof π , and the digest of statement d are all bounded by n , the security parameter. This is without loss of generality, and can be achieved by scaling down the security parameter.

- A family of hash functions $\{\mathcal{H}_n\}_n$: to simplify the exposition, we here assume that both com and $\{\mathcal{H}_n\}_n$ are collision resistant against circuits of size $T'(\cdot)$, where $T'(\cdot)$ is “nice” super-polynomial function. As in [?], this assumption can be weakened to just collision resistance against polynomial-size circuits by modifying the protocol to use a “good” error-correcting code ECC (i.e., with constant distance and with polynomial-time encoding and decoding), and replace commitments $\text{com}(h(\cdot))$ with $\text{com}(h(\text{ECC}(\cdot)))$.
- An indistinguishability obfuscator $i\mathcal{O}$ for circuits.
- A constant-round WIUA argument system, a constant-round WISSP proof system, and a constant-round $\mathcal{ZK}$ argument system.

Let us now turn to specifying the protocol (P, V) . The protocol makes use of three parameters: $m(\cdot)$ is a polynomial that upper bounds the number of concurrent sessions; $\Gamma(\cdot)$ is a “nice” super-polynomial function such that $T(n), T'(n) \in \Gamma(n)^{\omega(1)}$, and $D(\cdot)$ is a “nice” super-constant function such that $D(n) \leq C(n)$. Let $m = m(n)$, $\Gamma = \Gamma(n)$ and $D = D(n)$. In the description below, when discussing $\mathbf{P}$ -certificates, we always consider the language L_D . For simplicity, below we do not explicitly discuss about the length of the random strings used by various algorithms.

The prover P and the verifier V , on common input 1^n and x and private input a witness w to P , proceed as follow:

Phase 1—Program Slot: P and V exchanges the following three messages.

- (a) V chooses a randomly sampled hash function $h \leftarrow \mathcal{H}_n$.
- (b) P sends a commitment c to 0^n using com , and random coins ρ_1 .

(c) V replies with a random “challenge” r of length $3mn$.

We call (c, r) the program-slot.

NOTE: In the simulation, the simulator commits to a program $\tilde{S}_1$.

Phase 2—Commit to Statement: P and V exchanges the following messages.

- (a) P sends a commitment c_2 to 0^n using com , and random coins ρ_2 .
- (b) P gives a WIUA argument of the statement that either $x \in L$ OR there exists $\tilde{S}_1 \in \{0, 1\}^{\Gamma(n)}$, $j \in [m]$, $s \in \{0, 1\}^n$, $\pi \in \{0, 1\}^n$, $\sigma \in \{0, 1\}^{\Gamma(n)}$, ρ , ρ_2 such that,

Knowledge of Statement: $c_2 = \text{com}(h(q); \rho_2)$, where $q \in \{0, 1\}^{3\Gamma}$.

Correctness of Statement: The statement q satisfy the following properties:

- **USE OF EMULATOR:** q can be parsed into $(\text{Emulator}^n, (\tilde{S}_1, (1^n, j, s), \sigma), r)$.
- **PROGRAM CONSISTENCY:** $c = \text{com}(h(\tilde{S}_1); \rho)$.

If the argument is not accepting, V aborts.

NOTE: By definition of the emulator Emulator^n , on input $(\tilde{S}_1, (1^n, j, s), \sigma)$, it will emulate the execution of the deterministic oracle machine $\tilde{S}(1^n, j, s)$ with oracle $\mathcal{O}_{V_{\text{cert}}}^n$ using answers stored in array σ .

The purpose of this Phase is twofold: First, it enforces a cheating prover to commit to the “trapdoor” statement before the CRS of the $\mathbf{P}$ -certificate is generated, and hence the soundness of the protocol only relies on the selective soundness of the $\mathbf{P}$ -certificate. Second, it checks whether the “trapdoor” statement has the right structure, in particular, the statement is about whether $\tilde{S}_1^{\mathcal{O}_{V_{\text{cert}}}}(1^n, j, s) = r$ with σ containing all the correct oracle answers (as enforced by Emulator^n). In the simulation, the simulator commits to the “trapdoor” statement, $q = (\text{Emulator}^n, (\tilde{S}_1, (1^n, j, s), \sigma), r)$, here.

Phase 3—Delegate Public Statement Processing: V delegates the public statement processing to P :

- (a) V generates $(PP, K) = \text{Setup}(1^n, D; \rho_{\text{Setup}})$ using random coins r_{CRS} , and sends PP .
- (b) P sends a commitment c_3 to 0^n using com , and random coins ρ_3 .
- (c) P gives a WIUA argument of the statement that either $x \in L$ OR there exists, $d \in \{0, 1\}^n$, $q \in \{0, 1\}^{3\Gamma}$, ρ_{PreGen} , ρ_2 , ρ_3 , such that,

Statement Consistency: $c_2 = \text{com}(h(q); \rho_2)$.

Digest Consistency: $c_3 = \text{com}(d; \rho_3)$.

Correctness of Digest: $d = \text{PreGen}(PP, q; \rho_{\text{PreGen}})$.

If the argument is not accepting, V aborts.

NOTE: The purpose of this Phase is to allow the verifier to delegate the computation of the digest of the statement to P . In simulation, the simulator will compute, commit to and prove correctness of $d = \text{PreGen}(PP, q; \rho_{\text{PreGen}})$. V cannot compute d itself, since (1) it does not know the “trapdoor” statement q and (2) the computation takes $\text{poly}(n, |q|)$, which is too expensive for the verifier.

Phase 4—Delegate Private CRS Generation: V delegates the private CRS generation to P :

- (a) V sends the indistinguishability obfuscation $\Lambda \xleftarrow{\$} i\mathcal{O}(\mathbf{P})$ of program $\mathbf{P} = \mathbf{P}^{n, c_3, PP, K, \rho_{\text{CRSGen}}}$ with c_4 , K , and a random string ρ_{CRSGen} hardwired in. $\mathbf{P}$ on input (d', ρ') checks whether $c_3 = \text{com}(d', \rho')$ and outputs $\kappa = \text{CRSGen}(PP, K, d'; \rho_{\text{CRSGen}})$ if it is the case, and $\perp$ otherwise. The functionality of $\mathbf{P}$ is described formally in Figure 2.

Circuit $\mathbf{P} = \mathbf{P}^{n, c_3, PP, K, \rho_{\text{CRSGen}}}$: On input (d', ρ') where $d' \in \{0, 1\}^n$ and $\rho' \in \{0, 1\}^n$, does:

- (a) Check if $c_3 = \text{com}(d'; \rho')$; if not, output $\perp$.
- (b) Otherwise output $\kappa = \text{CRSGen}(PP, K, d'; \rho_{\text{CRSGen}})$.

Circuit $\mathbf{Q} = \mathbf{Q}^{n, c_3, \kappa}$: On input (d', ρ') where $d' \in \{0, 1\}^n$ and $\rho' \in \{0, 1\}^n$, does:

- (a) Check if $c_3 = \text{com}(d'; \rho')$; if not, output $\perp$.
- (b) Otherwise output κ .

The above circuits are padded to their maximum size.

Figure 2: Circuits used in the construction and proof of CZK protocol $\langle P, V \rangle$

- (b) V gives a $\mathcal{ZK}$ argument of the statement that there exists $K \in \{0, 1\}^n$, ρ_{Setup} , ρ_{CRSGen} , $\rho_{i\mathcal{O}}$, such that,

Correctness of Public Parameter: $(PP, K) = \text{Setup}(1^n, D; \rho_{\text{Setup}})$.

Correctness of Obfuscation: $\Lambda = i\mathcal{O}(\mathbf{P}^{c_3, PP, K, \rho_{\text{CRSGen}}}; \rho_{i\mathcal{O}})$

If the argument is not accepting, P aborts.

- (c) P sends commitment c_4 of 0^n using com and random coins ρ_4 .
- (d) P gives a $\mathcal{WISSP}$ proof of the statement that either $x \in L$ OR there exists $\text{CRS} \in \{0, 1\}^n$, $d' \in \{0, 1\}^n$, ρ' , ρ_4 , such that,

CRS Consistency: $c_4 = \text{com}(\text{CRS}; \rho_4)$.

Correctness of CRS: $\text{CRS} = (PP, \kappa)$ and $\kappa = \overline{\mathbf{P}}(d', \rho')$.

If the proof is not accepting, V aborts.

NOTE: The purpose of this Phase is to allow the verifier to delegate the computation of CRS to P . In simulation, the simulator will compute, commit to, and prove correctness of $\text{CRS} = (PP, \kappa)$, with $\kappa = \overline{\mathbf{P}}(d, \rho_3)$. V cannot compute κ itself, even though the computation takes only polynomial time in n , since d cannot be revealed to V in order to ensure the indistinguishability of the simulation. On the other hand, to ensure the “privacy” of the CRS computation, V delegates this computation via obfuscation.

Phase 5—Final Proof: P gives the final proof:

- (a) P gives a $\mathcal{WISSP}$ proof of the statement that either $x \in L$ OR there exists $\pi \in \{0, 1\}^n$, $\text{CRS} \in \{0, 1\}^n$, ρ_4 , such that,

CRS Consistency: $c_4 = \text{com}(\text{CRS}; \rho_4)$,

Proof Verification: π verifies w.r.t. CRS , $V_{\text{cert}}(1^n, \text{CRS}, \pi) = 1$.

V accepts if the proof is accepting.

NOTE: In the simulation, the simulator will compute the proof $\pi \xleftarrow{\$} \text{P}_{\text{cert}}(1^k, D, q, \text{CRS})$, and succeed in the final proof by using π and CRS, ρ_4 generated in the last phase as “trapdoor” witness.

Theorem 3. *The above protocol $\langle P, V \rangle$ is a concurrent $\mathcal{ZK}$ argument system for NP.*

The completeness of the protocol follows from the completeness of the WIUA argument of knowledge, *WISSP*, and the $\mathcal{ZK}$ argument. Below, we prove first the concurrent zero knowledge property and then the soundness of the protocol.

5.1 Proof of Concurrent Zero-Knowledge

The goal of our simulator is to try to “commit to its own code” and prove about its own execution using **P**-certificates in a way that prevents a blow-up in the running-time. Note that the only expensive part of this process is the generation of the **P**-certificates $\tilde{\pi}$; the rest of the computation has *a-priori* bounded complexity (depending only on the size and running-time of V^*). To take advantage of this observation, we thus have the simulator only commit to an oracle program that generates prover messages (in identically the same way as the actual simulator), but getting certificates $\tilde{\pi}$ from the **P**-certificate oracle $\mathcal{O}_{V_{\text{cert}}}$.

In more detail, to describe the actual simulator S , let us first describe two “helper” simulators S_1, S_2 . Roughly speaking, S_1 is an interactive machine that simulates prover messages in a “right” interaction with V^* . Additionally, S_1 expects to have access to oracle $\mathcal{O}_{V_{\text{cert}}}$ on the “left”, in particular, at any point, it can send a CRS string CRS and gets back the $\pi = \mathcal{O}_{V_{\text{cert}}}(\text{CRS})$ the unique accepting certificate w.r.t. this CRS (or $\perp$, if such a certificate does not exist); the oracle will be simulated by S_2 , who provides these “left” certificates.

Let us turn to a formal description of the S_1 and S_2 . To simplify the exposition, we assume w.l.o.g that V^* has its non-uniform advice z hard-coded, and is deterministic (as it can always get its random tape as non-uniform advice).

On a high-level, $S_1(1^n, x, M, s, \ell)$ acts as a prover in a “right” interaction, communicating with a concurrent verifier V^* , while accessing oracle on the “left”. (The input x is the statement to be proved, the input M will later be instantiated with the code of S_1 , and the input (s, ℓ) is used to generate the randomness for S_1 ; s is the seed for the forward secure pseudorandom generator g , and ℓ is the number of n -bit long blocks to be generated using g .) A communication round in the “right” interaction with V^* refers to a verifier message (sent by V^*) followed by a prover message (sent by S_1).

PROCEDURE OF SIMULATOR S_1 : Let us now specify how S_1 generates prover messages in its “right” interaction with V^* . $S_1^{\mathcal{O}_{V_{\text{cert}}}}(1^n, x, M, s, \ell)$ acts as follows:

Generate Randomness: Upon invocation, S_1 generates its “random-tape” by expanding the seed s ; more specifically, let $(s_\ell, s_{\ell-1}, \dots, s_1), (q_\ell, q_{\ell-1}, \dots, q_1)$ be the output of $g(s, \ell)$. We assume without loss of generality that S_1 only needs n bits of randomness to generate any prover message (it can always expand these n bits into a longer sequence using a PRG); in order to generate its j^{th} prover message, it uses q_j as randomness.

Simulate Phase 1—“Commit to its own code”: Upon receiving a hash function h_i in session i during the j^{th} communication round, S_1 provides a commitment c_i to (the hash of) the deterministic oracle machine $\tilde{S}_1(1^n, \alpha, s') = \text{wrap}(M(1^n, x, M, s', \alpha), V^*, \alpha)$, where $\text{wrap}(A, B, \alpha)$ is the program that lets A communicate with B for α rounds, while allowing A to access oracle $\mathcal{O}_{V_{\text{cert}}}$, and finally outputting B ’s message in the j^{th} communication round.

NOTE: That is, $\tilde{S}_1(1^n, \alpha, s', \tau)$ emulates α rounds of an execution between S_1 and V^* where S_1 expands out the seed s' into α blocks of randomness and additionally have access to $\mathcal{O}_{V_{cert}}$.

Simulate Phase 2—“Commit to the trapdoor statement”: Upon receiving a challenge r_i in session i during the j^{th} communication round, S_1 needs to commit to the “trapdoor” statement it will later prove in the final proof. To do so, it prepares statement $q_i = (\text{Emulator}^n, (\tilde{S}_1, (1^n, j, s_j), \tau_{j-1}), r_i)$, where τ_{j-1} is the list of oracle answers received by S_1 in the first $j - 1$ communication rounds.

NOTE: That is, the “trapdoor” statement is that the execution of $\tilde{S}_1(1^n, j, s_j)$, emulated by Emulator^n , outputs r , when its k^{th} oracle queries is answered using $\tau_{j-1,k}$; additionally, the validity of each answer is checked by ^{n} (i.e., the answer must be an accepting proof w.r.t. the query CRS string).

By construction of $\tilde{S}_1$, this means after j communication rounds between S_1 and V^* , where S_1 uses randomness expanded out from s_j , and oracle answers τ_{j-1} , V^* outputs r_i in the j^{th} communication round. Note that since we only require $\tilde{S}_1$ to generate the j^{th} verifier message, giving him the seed (s_j, j) as input suffices to generate all prover messages in rounds $j' < j$. It follows from the consistency requirement of the forward secure PRG that $\tilde{S}_1$ using (s_j, j) as seed will generate the exact same random sequence for the $j - 1$ first blocks as if running $\tilde{S}_1$ using (s, ℓ) as seed. Therefore, the “trapdoor” statement holds.

In later communication rounds, when S_1 receives a message from V^* belonging to the WIUA in Phase 2 of session i , S_1 proves honestly that it knows the statement q_i it is committing to in session i , and the statement is correctly formatted and consistent with the program $\tilde{S}_1$ committed to in Phase 1 of session i .

Simulate Phase 3— “Process the trapdoor statement”: Upon receiving a public parameter PP_i in session i during the j^{th} communication round, S_1 needs to commit to the digest d_i of the “trapdoor” statement q_i of session i . To do so, it computes honestly $d_i \xleftarrow{\$} \text{PreGen}(PP_i, q_i)$ and commits to d_i using com , and randomness ρ_i .

In later communication rounds, when S_1 receives a message from V^* belonging to the WIUA in Phase 3 of session i , S_1 proves honestly that it knows d_i committed to in Phase 3 of session i and it is computed correctly w.r.t. PP_i and a statement q_i committed to in Phase 2 of session i .

Simulate Phase 4— “Compute the CRS”: Upon receiving an obfuscated program Λ_i , S_1 acts as an honest verifier of the $\mathcal{ZK}$ argument to verify that PP_i and Λ_i in session i are correctly generated. Upon receiving the last message of the $\mathcal{ZK}$ argument, in the j^{th} communication round, S_1 needs to commit to the CRS_i of session i . To do so, it computes $\kappa_i = \Lambda_i(d_i, \rho_i)$. If the output is $\perp$, S_1 aborts. Otherwise, it commits to $\text{CRS}_i = (PP_i, \kappa_i)$ using com .

In later communication rounds, when S_1 receives a message from V^* belonging to the $\mathcal{WISSP}$ in Phase 4 of session i , S_1 proves honestly that it knows κ_i committed to in Phase 4 of session i and it is computed correctly w.r.t. Λ_i and a digest d_i committed to in Phase 3 of session i .

Simulate Phase 5— “Prove the trapdoor statement using P-certificate”: Upon receiving the last message from V^* in Phase 4 of session i , during the j^{th} communication round, S_1 needs to prove in the $\mathcal{WISSP}$ proof that there is a **P**-certificate that verifies the validity of the “trapdoor” statement q_i w.r.t. the CRS string CRS_i committed to in Phase 4 of session i . To do so, it sends query CRS_i to its oracle $\mathcal{O}_{V_{cert}}$, and obtains answer π_i . It aborts if $\pi_i = \perp$.

Otherwise, S_1 provides an honest $WISSP$ that $V_{\text{cert}}(1^n, \text{CRS}_i, \pi_i) = 1$ w.r.t. CRS_i which is the committed value in Phase 4 of session i .

PROCEDURE OF SIMULATOR S_2 : $S_2(1^n, x, M, s, \ell)$ internally emulates ℓ messages of an execution between $S_1(1^n, x, M, s, \ell)$ and V^* , and simulates the oracle $\mathcal{O}_{V_{\text{cert}}}$ for S_1 . In a communication round j when S_1 sends an oracle query CRS_i for a session i , S_2 generates a certificate π_i of the statement $q_i = (\text{Emulator}^n, (\tilde{S}_1, (1^n, j', s_{j'}), \tau_{j'-1}), r_{j'})$ w.r.t. CRS_i , that is, $\pi_i \stackrel{\$}{\leftarrow} \text{P}_{\text{cert}}(1^n, D, q_i, \text{CRS}_i)$ (where j' is the round in which the challenge r_i is sent by V^* , q_i and CRS_i are generated by S_1 (emulated internally by S_2) in Phase 2 and 4 of session i). S_2 checks if indeed $V_{\text{cert}}(1^n, \text{CRS}_i, \pi_i) = 1$, it outputs fail if this is not the case, and otherwise, feeds π_i to S_1 . Finally, S_2 outputs its view (which in particular, contains the view of V^*) at the end of the execution.

PROCEDURE OF THE FINAL SIMULATOR S : The final simulator $S(1^n, x)$ simply runs $S_2(1^n, x, S_1, s, T(n + |x|))$, where s is a uniformly random string of length n and $T(n + |x|)$ is a polynomial upper-bound on the number of messages sent by V^* given the common input $1^n, x$, and extracts out and outputs, the view of V^* from the output of S_2 . (In case that S_2 outputs fail, S outputs fail as well.)

Running-time of S . Let us first argue that S_1 runs in polynomial time.

1. In Phase 1, it only takes S_1 polynomial-time to generate the commitments (since V^* has a polynomial-length description, and thus also the code of $\tilde{S}_1$).
2. In Phase 2, it also only takes S_1 polynomial time to commit to the statements q_i (since Emulator^n , $(1^n, j, s_j)$, and r have fixed polynomial lengths, and $\tilde{S}_1$ and τ_{j-1} have polynomial length description, depending on the size of V^*). Furthermore, the witnesses of the WIUA in Phase 2 has polynomial length; by the relative prover efficiency condition of the WIUA, each such proof only requires some fixed polynomial-time.
3. In Phase 3, processing the statements q_i takes time polynomial in the length of the statement and n , which is polynomial. Furthermore, committing to the outputs d_i and proving about their correctness using WIUA also takes only polynomial time (by the relative prover efficiency of WIUA).
4. In Phase 4, since the CRS generation is very efficient, taking time polynomial in only the security parameter, S_1 completes all Phase 4 in polynomial time.
5. In Phase 5, the simulator proves about the verification of a $\mathbf{P}$ -certificate w.r.t. to a CRS string committed to in Phase 4. Since both steps takes time $\text{poly}(n)$, S_1 completes all Phase 5 in polynomial time.

Overall, the whole execution of S_1 takes some fixed polynomial time (in the length of V^* and thus also in the length of x .) It directly follows that also $\tilde{S}_1$'s running-time is polynomially bounded.

Finally, since S_2 is simply providing certificates about the execution of $\tilde{S}_1$, it follows by the relative prover efficiency condition of $\mathbf{P}$ -certificates, that S_2 runs in polynomial time, and thus also S .

Indistinguishability of the simulation Fix any cheating verifier V^* , we first argue that during the execution of S for simulating the view of V^* , the probability that S_2 (and hence S) outputs fail is negligible. By construction, S_2 outputs fail when for some session i , the proof π_i that it constructs honestly using P_{cert} does not verify w.r.t. the CRS_i that S_1 computes. It follows from the soundness of the $\mathcal{ZK}$ argument in Phase 4 of session i that, with overwhelming probability, V^* in session

i computes $(PP, K) \xleftarrow{\$} \text{Setup}(1^n, D)$ and the obfuscation $\Lambda \xleftarrow{\$} i\mathcal{O}(\mathbf{P})$ of $\mathbf{P} = \mathbf{P}^{n, c_3, PP, K, \rho_{\text{CRSGen}}}$ correctly w.r.t. some random strings ρ_{Setup} and $\rho_{i\mathcal{O}}$. In this case, since S_1 evaluates PreGen , commits to the produced digest d_i , and evaluates Λ_i honestly, it follows from the perfect correctness of the indistinguishability obfuscator, the perfect completeness of the $\mathbf{P}$ -certificate system, and the perfect binding property of com that as long as q_i is a true statement, S_2 would generate an accepting proof for it w.r.t. CRS_i . By construction, q_i is a true statement. Therefore, the probability that S_2 outputs fail is negligible.

Below we argue about the indistinguishability of the simulation conditioning on that S_2 does not output fail. Assume that there exists a cheating verifier V^* , a distinguisher D and a polynomial p such that the real view and the simulated view of V^* can be distinguished by D with probability $\frac{1}{p(n)}$ for infinitely many n . More formally, for infinitely many $n \in N$, $x \in L \cap \{0, 1\}^{\text{poly}(n)}$, $w \in \mathbf{R}_L(x)$ and $z \in \{0, 1\}^{\text{poly}(n)}$, it holds that

$$|\Pr[D(\text{View}_{V^*} \langle P(w), V^*(z) \rangle(1^n, x)) = 1] - \Pr[D(S(1^n, x, z)) = 1]| \geq \frac{1}{p(n)} \quad (1)$$

Consider such n, x, z (and assume that z is hard-coded into the description of V^*), and consider $T = T(n + |x|)$ hybrid experiments (recall that $T(n + |x|)$ is the maximum number of communication rounds given common input $1^n, x$).

- In hybrid H_j , the first j communication rounds are simulated exactly as by S (using pseudo-randomness), but all later communication round $j' > j$ are generated honestly using *true* randomness q'_j being uniformly distributed in $\{0, 1\}^n$. More precisely, every prover commitment sent after round j is a commitment to 0^n (i.e., Step 1-(b), 2-(a), 3-(b) and 4-(c)); every WIUA argument (i.e., Step 2-(b), 3-(c)) and every WLSSP proof (i.e., Step 4-(d), 5-(a)) that *start after* round j uses the true witness w instead of “fake” witnesses that S uses; however, every WIUA argument and WLSSP proof that start at or before round j are still proven using (appropriate) “fake” witnesses as S does; importantly, all prover messages generated after round j uses truly random coins.

It follows by Equation 1 and a hybrid argument that there exist some j and a polynomial p'' such that D distinguishes H_j and H_{j+1} with probability $\frac{1}{p''(n)}$. Now, consider another hybrid experiment $\tilde{H}_j$ that proceeds just at H_{j+1} , but where true randomness is used in communication round $j + 1$ (but still committing to the the same values as S does and using “fake” witness as S does). It follows by the forward security of the PRG g that the outputs of H_{j+1} and $\tilde{H}_j$ are indistinguishable—the reason we need *forward security* is that to emulate communication rounds $j' \leq j$, the seeds $s_{j'}$ may need to be known (as they are part of the “trapdoor” statements). Indistinguishability of $\tilde{H}_j$ and H_j follows directly by either the hiding property of the commitment scheme (if in the $j + 1$ round, the prover message is a commitment), or the witness indistinguishability property of the WIUA or WLSSP (if in this round, the prover message is a message of WIUA or WLSSP). It thus leads to a contradiction and completes the proof of the indistinguishability of the simulation.

5.2 Proof of Soundness

Large-Scale Secure Computation: Multi-party Computation for (Parallel) RAM Programs

Elette Boyle^{1*}, Kai-Min Chung², and Rafael Pass^{3**}

¹ Technion Israel, eboyle@alum.mit.edu

² Academia Sinica, kmchung@iis.sinica.edu.tw

³ Cornell University, rafael@cs.cornell.edu

Abstract. We present the first efficient (i.e., polylogarithmic overhead) method for securely and privately processing large data sets over multiple parties with *parallel, distributed algorithms*. More specifically, we demonstrate load-balanced, statistically secure computation protocols for computing Parallel RAM (PRAM) programs, handling $(1/3 - \epsilon)$ fraction malicious players, while preserving up to polylogarithmic factors the computation, parallel time, and memory complexities of the PRAM program, aside from a one-time execution of a broadcast protocol per party. Additionally, our protocol has **polylog** communication locality—that is, each of the n parties speaks only with **polylog**(n) other parties.

1 Introduction

Large data sets, such as medical data, genetic data, transaction data, the web and web access logs, and network traffic data, are now in abundance. Much of the data is stored or made accessible in a distributed fashion, having necessitated the development of efficient distributed protocols that compute over such data. In particular, novel programming models for processing large data sets with *parallel, distributed algorithms*, such as MapReduce (and its implementation Hadoop) are emerging as crucial tools for leveraging this data in important ways.

But these methods require that the data itself is revealed to the participating servers performing the computation—and thus blatantly violate the privacy of potentially sensitive data. As a consequence, such methods

* The research of the first author has received funding from the European Union's Tenth Framework Programme (FP10/ 2010-2016) under grant agreement no. 259426 ERC-CaC, and ISF grant 1709/14.

** Pass is supported in part by a Alfred P. Sloan Fellowship, Microsoft New Faculty Fellowship, NSF Award CNS-1217821, NSF CAREER Award CCF-0746990, NSF Award CCF-1214844, AFOSR YIP Award FA9550-10-1-0093, and DARPA and AFRL under contract FA8750-11-2- 0211. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the Defense Advanced Research Projects Agency or the US Government.

cannot be used in many critical applications (e.g., discovery of causes or treatments of diseases using genetic or medical data).

In contrast, methods such as secure multi-party computation (MPC), introduced in the seminal works of Yao [Yao86] and Goldreich, Micali and Wigderson [GMW87], enable securely and privately performing any computation on individuals private inputs (assuming some fraction of the parties are honest). However, despite great progress in developing these techniques, there are no MPC protocols whose efficiency and communication requirements scale to the modern regime of large-scale distributed, parallel data processing.

We are concerned with merging these two approaches. In particular,

We seek MPC protocols that efficiently (technically, with polylogarithmic overhead) enable secure and private processing of large data sets with parallel, distributed algorithms.

Explicitly, in this large-scale regime, the following properties are paramount:

1. *Exploiting Random Access.* Computations on large data sets are frequently “lightweight”: accessing a small number of dynamically chosen data items, relying on conditional branching, and/or maintaining small memory. This means that converting a program first into a circuit to enable its secure computation, which immediately obliterates these gains, will not be a feasible option.
2. *Exploiting Parallelism.* In fact, as mentioned, to effectively solve large-scale problems, modern programming models heavily leverage parallelism. The notion of a Parallel RAM (PRAM) better captures such computing models. In the PRAM model of computation, several (polynomially many) CPUs run simultaneously, potentially communicating with one another, while accessing the same shared external memory. We consider a PRAM model with a variable number of CPUs but with a fixed activation structure (i.e., what processors are activated at which time steps is fixed). Note that such a model simultaneously captures RAMs (a single CPU) and circuits (the circuit topology dictates the CPU activation structure).
3. *Exploiting Plurality of Users.* In the setting of MPC we would like to leverage not only parallelism within a single party (i.e., if a party has multiple CPUs that may run in parallel), but also that we have a large number of parties that can run in parallel. So, if we have n parties, each with k processors, we ideally would like to securely compute PRAMs that use nk CPUs (as opposed to just k CPUs).

Additionally, the following desiderata are often of importance:

3. *Load balancing.* When the data set contains tens or hundreds of thousands of users’ data, it is often unreasonable to assume that any single user can provide memory, computation, or communication resources on the order of the data of *all users*. Rather, we would like to *balance* the load across nodes.
4. *Communication Locality.* In many cases, establishing a secure communication channel with a large number of distinct parties may be costly, and thus we would like to minimize the *locality of communication* [BGT13]: that is, the number of total parties that each party must send and receive message to during the course of the protocol.

To date, no existing work addresses secure computation of Parallel RAM programs. Indeed, nearly all results in MPC require a *circuit* model for the function being evaluated (including the line of work on scalable MPC [DI06, DIK⁺08, DKMS12, ZMS14]), and thus inherit resource requirements that are linear in the circuit size. Even for (sequential) RAM, the only known protocols either only handle two parties [OS97, GKK⁺11, LO13, GGH⁺13], or in the context of multi-party computation require all parties to store *all inputs* [DMN11], rendering the protocol useless in a large-scale setting (even forgetting about computation load balancing and locality).

1.1 Our Results

We present a statistically secure MPC for (any sequence of) PRAMs handling $(1/3 - \epsilon)$ fraction static corruptions in a synchronous communication network, with secure point-to-point channels. In addition, our protocol is strongly *load balanced* and *communication local* (i.e., $\text{polylog}(n)$ locality). We state our theorem assuming each party itself is a k -processor PRAM, for parameter k .

Theorem 1 (Informal – Main Theorem). *For any constant $\epsilon > 0$ and polynomial parallelism parameter $k = k(n)$, there exists an n -party statistically secure (with error negligible in n) protocol for computing any adaptively chosen sequence of PRAM programs Π_j with fixed CPU activation structures (and that may have bounded shared state), handling $(1/3 - \epsilon)$ fraction static corruptions with the following complexities, where each party is a k -processor PRAM (and where $|x|, |y|$ denote per-party input and output size,⁴ $\text{space}(\Pi)$, $\text{comp}(\Pi)$, and $\text{time}(\Pi)$ denote the worst-case space, computation, and (parallel) runtime of Π , and $\text{CPUs}(\Pi)$ denotes the number of CPUs of Π):*

- Computation per party, per Π_j : $\tilde{O}(\text{comp}(\Pi_j)/n + |y|)$.
- Time steps, per Π_j : $\tilde{O}(\text{time}(\Pi_j) \cdot \max\{1, \frac{\text{CPUs}(\Pi)}{nk}\})$.
- Memory per party: $\tilde{O}(|x| + |y| + \max_{j=1}^N \text{space}(\Pi_j)/n)$.
- Communication Locality: $\tilde{O}(1)$.

given a one-time preprocessing phase with complexity:

- Computation per party: $\tilde{O}(|x|)$, plus single broadcast of $\tilde{O}(1)$ bits.
- Time steps: $\tilde{O}(\max\{1, \frac{|x|}{k}\})$.

Additionally, our protocol achieves a strong “online” load-balancing guarantee: at all times during the protocol, all parties’ communication and computation loads vary by at most a constant multiplicative factor (up to a $\text{polylog}(n)$ additive term).

Remark 1 (Round complexity). As is the case with all general MPC protocols in the information-theoretic setting to date, the round complexity of our protocol corresponds directly with the time complexity (as when restricted to circuits, parallel complexity corresponds to circuit depth). That is, for each evaluated PRAM program Π_j , the protocol runs in $\tilde{O}(\text{time}(\Pi_j))$ sequential communication rounds to securely evaluate Π_j .

⁴ For simplicity of exposition, we assume all parties have the same input size and receive the same output.

Remark 2 (On the achieved parameters). Note that in terms of memory, each party only stores her input, output, and her “fair” share of the required space complexity, up to polylogarithmic factors. In terms of computation (up to polylogarithmic factors), each party does her “fair” share of the computation, receives her outputs, and in addition is required to read her entire input at an initial preprocessing stage (even though the computations may only involve a subset of the input bits; this additional overhead of “touching” the whole input once is necessary to achieve security).⁵ Finally, the time complexity corresponds to the parallel complexity of the PRAM being computed, as long as the combined number of available processors nk from all parties matches or exceeds the number of required parallel processes of the program (and degrades with the corresponding deficit).

Remark 3 (Instantiating the single-use broadcast). The broadcast channel can be instantiated either by the $O(\sqrt{n})$ -locality broadcast protocol of King *et al.* [KSSV06], or the $\text{polylog}(n)$ -average locality protocol of [BSGH13] at the expense of a cost of a one-time per-party computational cost of $O(\sqrt{n})$, or average cost of $\text{polylog}(n)$, respectively. We separate the broadcast cost from our protocol complexity measures to emphasize that any (existing or future) broadcast protocol can be directly plugged in, yielding associated desirable properties.⁶

1.2 Construction Overview

Our starting point is an *Oblivious PRAM (OPRAM)* compiler [BCP14b,GO96], a tool that compiles any PRAM program into one whose memory access patterns are independent of the data (i.e., “oblivious”). Such a compiler (with polylogarithmic overhead) was recently attained by [BCP14b]. Indeed, it is no surprise that such a tool will be useful toward our goal. It has been demonstrated in the sequential setting that Oblivious (sequential) RAM (ORAM) compilers can be used to build secure 2-party protocols for RAM programs [OS97,GKK⁺11,LO13,GGH⁺13]. Taking a similar approach, building upon the OPRAM compiler of [BCP14b] directly yields 2-party protocols for PRAMs.

However, OPRAM on its own does not directly provide a solution for *multi*-party computation (when there are many parties). While this approach gives protocols whose complexities scale well with the RAM (or

⁵ For general secure computation, and even if we restrict to functionalities that only access a few parties’ inputs, and only a few bits of their data, essentially all parties must perform computation at least $\Omega(|x|)$. To see this, consider secure computation of a “multi-party Private Information Retrieval (PIR)” functionality: each party $i > 1$ has as input some “big data” x_i , and party 1 has as input a party index i and an index j into their data x_i . The functionality returns $x_i[j]$ (i.e., the j ’th bit of party i ’s data) to party 1 and nothing to everyone else. We claim that each party $i > 1$ must access every bit of x_i ; if not, it learns that that particular bit of its data was not requested, which it cannot learn in an ideal execution of the functionality.

⁶ For instance, it remains open to achieve statistically secure broadcast with worst-case $\text{polylog}(n)$ locality.

PRAM) complexity of the programs, the complexities grow poorly with the number of parties. Indeed, the only current technique for securely evaluating a RAM program on *multiple* parties' inputs [DMN11] is for all parties to hold secret shares of all parties' inputs, and then jointly execute (using standard MPC for circuits) the trusted CPU instructions of the OPRAM-compiled version of the program. This means each party must communicate and maintain information of size equivalent to *all parties'* inputs, and everyone must talk to everyone else for every time step of the RAM program evaluation.

One may attempt to improve the situation by first electing a small $\text{polylog}(n)$ -size representative committee of parties, and then only performing the above steps within this committee. This approach drops the total communication and computation of the protocol to reasonable levels. However, this approach does not save the subset of elected parties from carrying the burden of the entire computation. In particular, each elected party must memory storage equal to the size of *all parties' inputs combined*, making the protocol unusable for “large-scale” computation. In this paper, we provide a new approach for dealing with this issue. We show how to use an OPRAM in a way that achieves balancing of memory, computation, and communication across all parties.

Our MPC construction proceeds in the following steps:

1. **From OPRAM to MPC.** Given an OPRAM, we begin by considering MPC in a “benign” adversarial setting, which we refer to as *oblivious multi-party computation*, where all parties are assumed to be honest, and we only require that an external attacker that views communication and activation (including memory and computation usages) patterns does not learn anything about the inputs. We show:
 - (a) OPRAM yields efficient *memory-balanced* oblivious MPC for PRAM.
 - (b) Using committee election techniques (à la [KLST11, DKMS12, BGT13]), any oblivious multi-party computation can be compiled into a standard secure MPC with only polylog overhead (and a one-time use of a broadcast channel per party).
2. **Load Balancing & Communication Locality.** We next show semi-generic compilers for “nice” (formally defined) *oblivious multi-party* protocols, each introducing only $\text{polylog}(n)$ overhead:
 - (a) From any “nice” protocol to one whose computation and communication are *load-balanced*.
 - (b) From any “nice” protocol to one that is both *load-balanced* and *communication local* (i.e., $\text{polylog}(n)$ locality).

Our final result is obtained by combining the above steps and observing that Step 1(b) preserves load-balancing and communication locality (and thus can be applied *after* Step 2). Let us mention that *just* Step 1 (together with existing construction of ORAMs) already yields the first MPC protocol for (sequential) RAM programs in which no party must store all parties' inputs. Additionally, *just* Step 1 (together with the OPRAM construction of [BCP14b]) yields the first MPC for PRAMs.

We now expand upon each of these steps.

MPC from OPRAM Recall that our construction proceeds via an intermediate notion of *oblivious* security, in which we do not require security against corrupted parties, but rather against an external adversary who sees the activation patterns (i.e., accessed memory addresses and computation times) and communication patterns (i.e., sender/receiver ids and message lengths) of parties throughout the protocol.

Oblivious MPC from OPRAM. At a high level, our protocol will emulate a *distributed* OPRAM⁷ structure, where the CPUs and memory cells in the OPRAM are *each associated with parties*. (Recall that we need only achieve “oblivious” security, and thus can trust individual parties with these tasks). The “CPU” parties will control the evaluation flow of the (OPRAM-compiled) program, communicating with the parties emulating the role of the appropriate memory cells for each address to be accessed in the (OPRAM-compiled) database.

The distributed OPRAM structure will enable us to evenly spread the memory burden across parties, incurring only $\text{polylog}(n)$ overhead in total memory and computation, and while guaranteeing that the communication patterns between committees (corresponding to data access patterns) do not reveal information on the underlying secret values.

This framework shares a similar flavor to the protocols of [DKMS12,BGJK12], which assign committees to each of the gates of a circuit being evaluated, and to [BGT13], which uses CPU and input committees to direct program execution and distributedly store parties’ inputs. The distributed OPRAM idea improves and conceptually simplifies the input storage handling of Boyle *et al.* [BGT13], in which n committees holding the n parties’ inputs execute a distributed “oblivious input shuffling” procedure to break the link between which committees are communicating and which inputs are being accessed in the computation.

Compiling from “Oblivious” Security to Malicious Security. We next present a general compiler taking an oblivious protocol to one that is secure against $(1/3 - \epsilon)n$ statically corrupted malicious parties. (This step can be viewed as a refinement and formalization of ideas from [KLST11,DKMS12,BGT13].) We ensure the compiler tightly preserves the computation, memory, load-balancing, and communication locality of the original protocol, up to $\text{polylog}(n)$ factors (modulo a one-time broadcast per party). This enables us to apply the transformation to any of the oblivious protocols resulting from the intermediate steps in our progression.

At a high level, the compiler takes the following form: (1) First, the parties collectively elect a large number of “good” committees, each of size $\text{polylog}(n)$, where “good” means each committee is composed of at least $2/3$ honest parties, and that parties are spread roughly evenly across committees. (2) Each party will verifiably secret share his input among the corresponding committee C_i . (3) From this point on, the role of each party P_i in the original protocol will be emulated by the corresponding

⁷ We remark that the term “distributed OPRAM” was used with a different meaning in [LO13], in regard to an OPRAM that was split across two users.

committee C_i . That is, each local P_i computation will be executed via a small-scale MPC among C_i , and each communication from P_i to P_j will be performed via an MPC among committees C_i and C_j .

The primary challenge in this step is how to elect such committees while incurring only $\text{polylog}(n)$ locality and computation per party. To do so, we build atop the “almost-everywhere” scalable committee election protocol of King *et al.* [KSSV06] to elect a single good committee, and then show that one may use a $\text{polylog}(n)$ -wise independent function family $\{F_s\}_{s \in S}$ to elect the remaining committees with small description size (in the fashion of [KLST11,BGT13], for the case of combinatorial samplers and computational pseudorandom functions), with committee i defined as $C_i := F_s(i)$ for fixed random seed s .

We remark that, aside from the one-time broadcast, this compiler preserves load balancing and $\text{polylog}(n)$ locality. Indeed, load balancing is maintained since the committee setup procedure is computationally inexpensive, and each party appears in roughly the same number of “worker” committees. The locality of the resulting protocol increases by an additive $\text{polylog}(n)$ for the committee setup, and a multiplicative $\text{polylog}(n)$ term since all communications are now performed among $\text{polylog}(n)$ -size committees instead of individual parties.

Load Balancing Distributed Protocols

Load-balancing (Without Locality). We now show how to modify our protocol such that the total computational complexity and memory balancing are preserved, while additionally achieving a strong *computation* load balancing property—with high probability, at all times throughout the protocol execution, every party performs close to $1/n$ fraction of current total work, up to an additive $\text{polylog}(n)$ amount of work. This will hold simultaneously for both computation and communication.⁸

We present and analyze our load-balancing solution in the intermediate *oblivious* MPC security setting (recall that one can then apply the compiler from Step 2(b) above to obtain malicious MPC with analogous load-balancing). Let us mention that there is a huge literature on “load-balanced distributed computation” (e.g., [ACMR95,MPS02,MR98,AAK08]): As far as we can tell, our setting differs from the typical studied scenarios in that we must load balance an underlying distributed protocol, as opposed to a collection of independent “non-communicating jobs”. Indeed, the main challenge in our setting is to deal with the fact that “jobs” talk to one another, and this communication must remain efficient also be made load balanced. Furthermore, we seek a load-balanced solution with communication locality.

We consider a large class of arbitrary (potentially load-unbalanced and large-locality) distributed protocols Π , where we view each party in this underlying protocol as a “job”. Our goal is to load-balance Π by passing

⁸ Note that while our current protocol is memory balanced, it is currently rather imbalanced in computation: e.g., the parties emulating OPRAM CPUs are required to perform computation that is proportional to the whole PRAM computation.

“jobs” between “workers” (which will be the actual parties in the new protocols). More precisely, we start off with any protocol Π that satisfies the following (natural) “nice” properties:

- Each “job” has $\text{polylog}(n)$ size state;
- In each round, each “job” performs at most $\text{polylog}(n)$ computation and communication;
- In each round, each “job” communicates (either sending or receiving a message) to at most one other “job”.

It can be verified that these properties hold for our oblivious MPC for PRAM protocol.

Our load-balanced version of such a protocol first randomly⁹ efficiently assigns “workers” (i.e., parties) to “jobs”. Next, whenever a worker W has performed “enough” work for a particular job J , it randomly selects a replacement worker W' and passes the job over to it (that is, it passes over the state of the job J —which is “small” by assumption). The key obstacle in our setting is that the job J may later communicate with many other jobs, and all the workers responsible for those jobs need to be informed of the switch (and in particular, who the new worker responsible for the job J is). Since the number of jobs is $\Omega(n)$, workers cannot afford to store a complete directory of which worker is currently responsible for each job.

We overcome this obstacle by first modifying Π to ensure that it has small locality—this enables each job to only maintain a short list of the workers currently responsible for the “*neighboring*” jobs. We achieve this locality by requiring that parties (i.e., jobs) in the original protocol Π route their messages along the hypercube. Now, whenever a worker W for a job J is being replaced by some worker W' , W informs all J ’s neighboring jobs (i.e., the workers responsible for them) of this change. We use the Valiant-Brebner [VB81] routing procedure to implement the hypercube routing because it ensures a desirable “low-congestion property,” which in our setting translates to ensuring that the overhead of routing is not too high for any individual worker.

The above description has not yet mentioned what it means for a worker to have done “enough” work for a job J . Each round a job is active (i.e., performing some computation), its “cost” increases by 1—we refer to this as an *emulation cost*. Additionally, each time a worker W is switched out from a job J , then J ’s and each of J ’s neighboring jobs’ costs are increased by 1—we refer to this as a *switch cost*. Finally, once a job’s (total) cost has reached a particular threshold τ , its cost is reset to 1 and the worker responsible for the job is switched out. The threshold τ is set to $2 \log M + 1$ where M is the number of jobs.

We show: (1) This switching does not introduce too much overhead. We, in fact, show that the total induced switching cost is bounded above by the emulation cost. (2) The resulting total work is load balanced across workers—we show this by first demonstrating that the protocol is load-balanced in expectation, and then using concentration to argue our stronger online load-balancing property.

⁹ In the actual analysis, we show that it also suffices to use $\text{polylog}(n)$ -wise independent randomness to pick this and subsequent assignments.

Finally, note that although communication between *jobs* is being routed through the hypercube, and thus the job communication protocol has small locality, the final load-balanced protocol, being run by *workers*, does *not* have small locality. This is because workers are assigned the role of many different jobs over time, and may possibly speak to a new set of neighbors for each position. (Indeed, over time, each worker will eventually need to speak to every other worker). We next show how to modify this protocol to achieve *locality*, while preserving load-balancing.

Achieving Both Load-Balancing and Locality. In our final step, we show how to modify the above-mentioned protocol to also achieve locality. We modify the protocol to also let *workers* route messages through a low-degree network (on top of the routing in the previous step). This immediately ensures locality. But, we must be careful to ensure that the additional message passing does not break load-balancing.

A natural idea is to again simply pass messages between *workers* along a low-degree hypercube network via Valiant-Brebner (VB) routing [VB81]. Indeed, the low-congestion property will ensure (as before) that routing does not incur too large an overhead for each worker.

However, when analyzing the overall load balance (for workers), we see an inherent distinction between this case and the previous. Previously, the nodes of the hypercube corresponded to *jobs*, each emulated by workers who swap in and out over time. When the underlying jobs protocol required job s to send a message to job t , the resulting message routing induced a cost along a path of neighboring jobs (that is, the workers emulating them), *independent of which workers are currently emulating them*. This independence, together with the fact that a worker passes his job after performing “enough” work for it, enabled us to obtain concentration bounds on overall load balancing over the random assignment of workers to jobs.

Now, the nodes correspond directly to *workers*. When the underlying jobs protocol requires a message transferred from job s to job t , routing along the workers’ graph must traverse a path from the *worker currently emulating job s* to the *worker currently emulating job t* , removing the crucial independence property from above. Even worse, workers along the routing path can now incur costs *even if they are not assigned to any job*. In this case, it is not even clear that job passing in of itself will be sufficient to ensure balancing.

To get around these issues, we add an extra step in the VB routing procedure (itself inspired by [VB81]) to break potential bad correlations. The idea is as follows: To route from the worker W_s emulating job s to the worker W_t emulating job t , we first route (as usual) from W_s to a *random* worker W_u , and then from W_u to W_t ; i.e., travel from W_s to W_t by “walking into the woods” and back. We may now partition the cost of routing into these two sub-parts, each associated with a single active job (s or t). Now, although workers along the worker-routing path will still incur costs from this routing (even though their jobs may be completely unrelated), the *distribution* of these costs on workers depends only on the identity of the initiating worker (W_s or W_t). We may thus

generalize the previous analysis to argue that if the expectation of work is load-balanced, then it still has concentration in this case.

For a modular analysis, we formalize the required properties of the underlying communication network and routing algorithm (to be used for the s -to- u and u -to- t routing) as a *local load-balanced routing network*, and show that the hypercube network together with VB routing satisfies these conditions.

1.3 Discussion and Future Work

With the explosive growth of data made available in a distributed fashion, and the growth of efficient parallel, distributed algorithms (such as those enabled by MapReduce) to compute on this data, ensuring privacy and security in such large-scale parallel settings is of fundamental importance. We have taken the first steps in addressing this problem by presenting the first protocols for secure multi-party computation, that with only polylogarithmic overhead, enable evaluating PRAM programs on a (large) number of parties' inputs. Our work leaves open several interesting open problems:

Honest Majority. We have assumed that $2/3$ of the players are honest.

In the absence of a broadcast channel,¹⁰ it is known that this is optimal. But if we assume the existence of a broadcast channel, it may suffice to assume $1/2$ fraction honest players.

Asynchrony. Our protocol assumes a synchronous communication network. We leave open the handling of asynchronous communication.

Trading efficiency for security. An interesting avenue to pursue are various tradeoffs between boosted efficiency and partial sacrifices in security. For example, in some settings, it is not detrimental to leak which parties' inputs were used within the computation; in such scenarios, one could then hope to remove the one-time $\Theta(n|x|)$ input preprocessing cost. Similarly, it may be acceptable to reveal the input-specific resources (runtime, space) required by the program on parties inputs; in such cases, we may modify the protocol to take only input-specific runtime and use input-specific memory.

In this work we focus only on achieving standard "full" security. However, we remark that our protocol can serve as a solid basis for achieving such tradeoffs (e.g., a straightforward tweak to our protocol results in input-specific resource use).

Communication complexity. As with all existing generic multi-party computation protocols in the information-theoretic setting, the communication complexity of our protocol is equal to its computation complexity. In contrast, in the computational setting (based on cryptographic assumptions), protocols with communication complexity below the complexity of the evaluated function have been constructed by relying on *fully homomorphic encryption (FHE)* [Gen09] (e.g., [Gen09,AJLA⁺12,MSS13]). We leave as an interesting open question whether FHE-style techniques can be applied also to our protocol to improve the communication complexity, based on computational assumptions.

¹⁰ While the statement of our result makes use of a broadcast channel, as we mention, this channel can also be instantiated with known protocols.

1.4 Overview of the Paper

Section 2 contains preliminaries. In Section 3 we provide our ultimate theorem, and the sequence of intermediate notions and theorems which combine to yield this final result. We refer the reader to the full version of this work [BCP14a] for a complete descriptions and proofs.

2 Preliminaries

2.1 Multi-party Computation (MPC)

Protocol Syntax. We model parties as (parallel) RAM machines. An n -party protocol Φ is described as a collection of n (parallel) RAM programs $(P_i)_{i \in [n]}$, to be executed by the respective parties, containing additional special communication instructions $\text{Comm}(i, \text{msg})$, indicating for the executing party to send message msg to party i .

The per-party space, computation, and time complexities of the protocol $\Phi = (P_i)_{i \in [n]}$ are defined directly with respect to the corresponding party's PRAM program P_i , where each Comm is charged as a single computation time step. (See Section 2.2 for a definition of $\text{CPUs}(P)$, $\text{space}(P)$, $\text{comp}(P)$, $\text{time}(P)$ for PRAM P). The analogous total protocol complexities are defined as expected: Namely, $\text{space}(\Phi)$ and $\text{comp}(\Phi)$ are the *sums*, $\text{space}(\Phi) = \sum_{i \in [n]} \text{space}(P_i)$, $\text{comp}(\Phi) = \sum_{i \in [n]} \text{comp}(P_i)$, and $\text{time}(\Phi)$ is the *maximum*, $\text{time}(\Phi) = \max_{i \in [n]} \text{time}(P_i)$.

MPC Security. We consider the standard notion of (statistical) MPC security. We refer the reader to e.g. [BGW88] for more a more complete description of MPC security within this setting.

2.2 Parallel RAM (PRAM) Programs

A Concurrent Read Concurrent Write (CRCW) m -processor *parallel random-access machine (PRAM)* with memory size n consists of numbered processors $\text{CPU}_1, \dots, \text{CPU}_m$, each with local memory registers of size $\log n$, which operate synchronously in parallel and can make access to shared “external” memory of size n .

A PRAM program Π (given m, n , and some input x stored in shared memory) provides CPU-specific execution instructions, which can access the shared data via commands $\text{Access}(r, v)$, where $r \in [n]$ is an index to a memory location, and v is a word (of size $\log n$) or $\perp$. Each $\text{Access}(r, v)$ instruction is executed as:

1. **Read** from shared memory cell address r ; denote value by v_{old} .
2. **Write** value $v \neq \perp$ to address r (if $v = \perp$, then take no action).
3. **Return** v_{old} .

In the case that two or more processors simultaneously initiate $\text{Access}(r, v_i)$ with the same address r , then all requesting processors receive the previously existing memory value v_{old} , and the memory is rewritten with the value v_i corresponding to the lowest-numbered CPU i for which $v_i \neq \perp$.

We more generally support PRAM programs with a dynamic number of processors (i.e., m_i processors required for each time step i of the computation), as long as this sequence of processor numbers $m_1, m_2, \dots$ is fixed, public information. The complexity of our OPRAM solution will scale with the number of required processors in each round, instead of the maximum number of required processors.

We consider the following *worst-case* metrics of a PRAM (over all inputs):

- $\text{CPUs}(\Pi)$: number of parallel processors required by Π .
- $\text{space}(\Pi)$: largest database address accessed by Π .
- $\text{time}(\Pi)$: maximum number of time steps taken by any processor to evaluate Π (where each Access is charged as a single step).¹¹
- $\text{comp}(\Pi)$: the total sum of all computation steps of active CPUs evaluating Π (which, for programs with fixed activation schedules as we consider, is a fixed value).

3 Local, Load-Balanced MPC for PRAM

Ultimately, we construct a protocol that securely realizes the ideal functionality $\mathcal{F}_{\text{PRAMs}}$ (Figure 1) for evaluating a sequence of PRAM programs (with bounded state maintained between program) on parties' fixed inputs. For simplicity of exposition, we assume each party has equal input size and receives the same output. We further assume the total remnant state from one program execution to the next is bounded in size by the combined input size of all parties.¹²

Theorem 2 (Main Theorem). *For any constant $\epsilon > 0$ and polynomial parallelism parameter $k = k(n)$, there exists an n -party statistically secure (with error negligible in n) protocol realizing the functionality $\mathcal{F}_{\text{PRAMs}}$, handling $(1/3 - \epsilon)$ fraction static corruptions with the following complexities, where each party is a k -processor PRAM (and where $|x|, |y|$ denote per-party input and output size, $\text{space}(\Pi)$, $\text{comp}(\Pi)$, and $\text{time}(\Pi)$ denote the worst-case space, computation, and (parallel) runtime of Π , and $\text{CPUs}(\Pi)$ denotes the number of CPUs of Π):*

- Computation per party, per Π_j : $\tilde{O}(\text{comp}(\Pi_j)/n + |y|)$.
- Time steps, per Π_j : $\tilde{O}(\text{time}(\Pi_j) \cdot \max\{1, \frac{\text{CPUs}(\Pi_j)}{nk}\})$.
- Memory per party: $\tilde{O}(|x| + |y| + \max_{j=1}^N \text{space}(\Pi_j)/n)$.
- Communication Locality: $\tilde{O}(1)$.

given a one-time preprocessing phase with complexity:

- Computation per party: $\tilde{O}(|x|)$, plus single broadcast of $\tilde{O}(1)$ bits.
- Time steps: $\tilde{O}(\max\{1, \frac{|x|}{k}\})$.

¹¹ We remark that the PRAM time complexity of any function f is bounded above by its circuit depth complexity (where the PRAM complexity of f is defined as the minimal value of $\text{time}(\Pi)$ of any PRAM Π which evaluates f).

¹² To support larger shared state size $\text{space}^{\text{Remnant}}$, the memory requirements of the protocol must grow with an extra additive $\tilde{O}(\text{space}^{\text{Remnant}})$.

Ideal Functionality $\mathcal{F}_{\text{PRAMs}}$:
 $\mathcal{F}_{\text{PRAMs}}$ running with parties $P_1, \dots, P_n$ and an adversary proceeds as follows. The functionality maintains longterm storage of parties' inputs $\{x_i\}_{i \in [n]}$ (each of equal size $|x|$), per-CPU state information state_i , and remnant memory $\text{data}^{\text{Remnant}}$ of total size $\text{space}^{\text{Remnant}} \in O(n \cdot |x|)$ transferred from computation to computation.

- Initialize $\text{data}^{\text{Remnant}} \leftarrow \emptyset$ and $\text{state}_i \leftarrow \emptyset$ for each processor $i \in [m]$.
- Input Submission: Upon receiving an input $(\text{commit}, \text{sid}, \text{input}, x_i)$ from party P_i , record the value x_i as the input of P_i .
- Computation: Upon receiving a tuple $(\text{compute}, \text{sid}, \Pi, \text{space}, \text{time})$ consisting of an m -processor PRAM program Π , a space bound space , and a time bound time , execute Π as $(\text{output}, \text{state}_1, \dots, \text{state}_m, \text{data}^{\text{Remnant}}) \leftarrow \Pi(x_1, \dots, x_n, \text{state}_1, \dots, \text{state}_m, \text{data}^{\text{Remnant}})$ with the current value of state_i for each CPU $i \in [m]$. Send output to all parties.

Fig. 1: The ideal functionality $\mathcal{F}_{\text{PRAMs}}$, corresponding to secure computation of a sequence of adaptively chosen PRAMs on parties' inputs.

Additionally, the protocol achieves $\text{polylog}(n)$ communication locality, and a strong “online” load-balancing guarantee:

Online Load Balancing: For every constant $\delta > 0$, with all but negligible probability in n , the following holds at all times during the protocol: Let cc and $\text{cc}(P_j)$ denote the total communication complexity and communication complexity of party P_j , comp and $\text{comp}(P_j)$ denote the total computation complexity and computation complexity of party P_j , we have

$$\begin{aligned} \frac{(1-\delta)}{n} \text{cc} - \text{polylog}(n) &\leq \text{cc}(P_j) \leq \frac{(1+\delta)}{n} \text{cc} + \text{polylog}(n) \\ \frac{(1-\delta)}{n} \text{comp} - \text{polylog}(n) &\leq \text{comp}(P_j) \leq \frac{(1+\delta)}{n} \text{comp} + \text{polylog}(n). \end{aligned}$$

3.1 Proof of Main Theorem

At a very high level, the proof takes three steps: We first obtain MPC realizing $\mathcal{F}_{\text{PRAMs}}$ with a weaker notion of *oblivious* security. We then show how to attain communication locality and load balancing, while preserving oblivious security. (This combines two steps described within the introduction). Finally, we convert the obliviously secure protocol to one secure in the malicious setting. We now proceed to describe these steps in greater technical detail.

Step 1: Oblivious-Secure MPC for PRAM. Intuitively, an adversary in the oblivious model is not allowed to corrupt any parties, and instead is restricted to seeing the “externally measurable” properties of the protocol (e.g., party response times, communication patterns, etc).

Definition 1 (Oblivious secure MPC). *Secure realization of a functionality F by a protocol in the oblivious model is defined by the following real-ideal world scenario:*

Ideal World: Same as standard MPC without corrupted parties. That is, the adversary learns only public outputs of the functionality F evaluated on honest-party inputs.

Real World: Instead of corrupting parties, viewing their states, and controlling their actions (as in the standard malicious adversarial setting), the adversary is now limited as an external observer, and is given access only to the following information:

- *Activation Patterns:* Complete list of tuples of the form
 - (timestep, party-id, compute-time): Specifying all local computation times of parties.
 - (timestep, party-id, local-mem-addr): Specifying all memory access patterns of parties.
- *Communication Patterns:* Complete list of tuples of the form
 - (timestep, sndr-id, rcvr-id, msg-len): Specifying all sender-receiver pairs, in addition to the corresponding communicated message bit-length.

The output of the real-world experiment consists of the outputs of the (honest) parties, in addition to an arbitrary PPT function of the adversary’s view at the conclusion of the protocol.

(Statistical) Security: For every PPT adversary $\mathcal{A}$ in the real-world execution, there exists a PPT ideal-world adversary $\mathcal{S}$ for which for every environment $\mathcal{Z}$, we have $\text{output}_{\text{Real}}(1^k, \mathcal{A}, \mathcal{Z}) \stackrel{s}{\cong} \text{output}_{\text{Ideal}}(1^k, \mathcal{S}, \mathcal{Z})$.

Toward our result, it will be advantageous to think of computations as composed of several sub-parts, or “jobs,” that each maintain and compute on small polylogarithmic-size state (Note that this is natural in the PRAM setting, where each CPU has polylogarithmic-size local memory). Later, to achieve load balancing, jobs will be assigned to and passed around between “workers,” so that each worker roughly performs the same amount of work. (The small state requirement per job will guarantee that “job passing” is not too expensive). Then, to obtain malicious security, each worker will ultimately be emulated by a committee of parties via small-scale MPCs; because of the polynomial overhead in the underlying MPC protocol, it will be important that this is only done for computations of $\text{polylog}(n)$ size on $\text{polylog}(n)$ -size memory.

We now define the notion of a protocol in the *jobs model*.

Definition 2 (Jobs Model). Let n be a security parameter. A jobs protocol consists of a $\text{poly}(n)$ -size set *Jobs* of agents (called jobs), and a distributed protocol description $\Pi_{\mathcal{J}}$, instructing each job to perform local computations and to communicate over a synchronized network (via point-to-point communication), with the following properties:

- *Bounded memory:* each job’s space complexity is $w \in \text{polylog}(n)$.
- *Bounded per-round computation and communication:* the computation and communication complexity of each job at each round is upper bounded by $w \in \text{polylog}(n)$.

A job is active in a round if it performs computation within this round. A jobs protocol is further said to have injective communication if the following property is satisfied:

- *Injective communication:* each round, a set of jobs are activated, and each sends a single $\text{polylog}(n)$ -sized message to a distinct job.

By convention, we assume the first m_{in} jobs of a jobs protocol are *input jobs*, the last m_{out} are *output jobs*, and the remaining jobs are *helper jobs*. Each input job J_i holds a single-word input $x_i \in \{0,1\}^w$ (for $w \in \text{polylog}(n)$); output and helper jobs have no input. We then have a canonical correspondence between functionalities in the standard n -party setting and the equivalent functionalities in the Worker-Jobs Model:

- Functionality $\mathcal{F}$: In the n -party setting. Accepts inputs x_i from each party P_i , evaluates $y \leftarrow F(x_1 || \dots || x_n)$, outputs the resulting value y to all parties P_i .
- Functionality $\mathcal{F}^{\text{Jobs}}$: In the Jobs Model. Accepts (short) inputs x_u^i from each Input Job, evaluates $y \leftarrow F(x_1 || \dots || x_\ell)$, and distributes the resulting value y (in short pieces) to the Output Jobs.

We may analogously define *oblivious security* of a jobs protocol (where jobs are honest and the adversary sees only “externally measurable” properties of the protocol, as in Definition 1). Within the jobs model, we thus wish to securely realize the functionality $\mathcal{F}_{\text{PRAMs}}^{\text{Jobs}}$, equivalent to $\mathcal{F}_{\text{PRAMs}}$ with the above syntactic change. Note that in the regime of oblivious security, a jobs protocol yields a *memory*-balanced protocol in the standard n -party model, by simply assigning jobs to the n parties evenly.

Theorem 3. *There exists an oblivious-secure protocol in the Jobs Model realizing the functionality $\mathcal{F}_{\text{PRAMs}}^{\text{Jobs}}$ for securely computing a sequence of N adaptively chosen PRAM programs Π_j , with the following complexities (where $n \cdot |x|, |y|$ denote the total input and output size, and $\text{space}(\Pi)$, comp , and $\text{time}(\Pi)$ denote the worst-case space, computation, and (parallel) runtime of Π over all inputs):*

- Number of jobs: $\tilde{O}(n \cdot |x| + |y| + \max_{j \in [N]} \text{space}(\Pi_j))$.
 - Computation complexity, per Π_j : $\tilde{O}(\text{comp}(\Pi_j))$.
 - Time steps, per Π_j : $\tilde{O}(\text{time}(\Pi_j))$.
 - The number of active jobs in each round is $O(\max_{j \in [N]} \text{CPU}(\Pi_j))$.
- given a one-time preprocessing phase with complexity
- Computation complexity: $\tilde{O}(n \cdot |x|)$.
 - Time steps: $\tilde{O}(1)$.

Further, the protocol has injective communication: in each round, each activated job sends a single $\text{polylog}(n)$ -size message to a distinct job.

Recall within the Jobs Model each job is limited to maintaining state of size $\text{polylog}(n)$; thus the *memory requirement* of the above protocol is

$$\tilde{O}\left(n \cdot |x| + |y| + \max_{j \in [N]} \text{space}(\Pi_j)\right),$$

based on the number of required jobs.

Idea of proof. The result builds upon the existence of an Oblivious PRAM compiler with $\text{polylog}(n)$ time and space overhead that is *collision-free* (i.e., where no two CPUs must access the same memory address in the same timestep), which is guaranteed to exist unconditionally based on [BCP14b]. In addition to the standard Input and Output jobs, our protocol will have one Helper job for each of the CPUs and each memory cell in the database of the OPRAM-compiled program. The CPU jobs

store the local state and perform the computations of their corresponding CPU. In each round that the i th CPU's instructions dictate a memory access at location $\text{addr}^{(i)}$, the CPU job i will communicate with the Memory job $\text{addr}^{(i)}$ to perform the access. (Thus, in each round, at most $2 \cdot \text{CPUs}(\text{OPRAM}(\Pi))$ jobs are active, where $\text{OPRAM}(\Pi)$ denotes the OPRAM-compilation of Π). Activation and communication patterns in the resulting protocol are simulatable directly by the OPRAM security. The preprocessing phase of the protocol corresponds to inserting all inputs into the OPRAM-protected database in parallel (i.e., emulating the OPRAM-compiled input insertion program that simply inserts each input x_i into address i of the database).

Step 2: Locality and Load Balancing. This step attains $\text{polylog}(n)$ communication *locality*,¹³ and computation *load balancing* from any jobs protocol Π_J with injective communication. We do so by emulating Π_J by a fixed set of parties (which we sometimes refer to as “workers”), where each worker is assigned several jobs, and will pass jobs to other workers once he has performed a certain amount of work. This yields a standard N -party protocol with a special *decomposable state* structure: i.e., parties’ memory can be decomposed into separate $\text{polylog}(n)$ -size memory blocks, which are only ever computed on independently or in pairs, in steps of $\text{polylog}(n)$ computation per round. This is because parties’ computation is limited to individual jobs to which it was assigned.¹⁴

Definition 3 (Decomposable State). *An N -party protocol Π is said to have decomposable state if for every party P , the local memory mem of P can be decomposed into $\text{polylog}(n)$ -size blocks $\text{mem} = (\text{mem}_1, \text{mem}_2, \dots, \text{mem}_m)$ such that: In each round of Π , the (parallel) local computation performed by party P is described as a list $\{(i, j, f_{i,j})\}_{(i,j) \in I}$ for some $I \subseteq [m] \times [m]$, such that each $f_{i,j}$ has complexity $\text{polylog}(n)$. For each $(i, j) \in I$, party P executes $(\text{mem}_i, \text{mem}_j) \leftarrow f_{i,j}(\text{mem}_i, \text{mem}_j)$.¹⁵ By convention, received communication messages are stored in local memory.*

We achieve the following “fully load-balanced” properties. Note that the first two properties correspond directly to our final load-balancing goal. The final property will be used to ensure that no individual worker is ever assigned drastically more than the expected number of simultaneous parallel computation tasks; this is important since workers will eventually be emulated by (technically, committees of) parties, who themselves may have bounded parallelism capability (i.e., small number of CPUs).

Definition 4 (Fully Load Balanced). *An N -party protocol Π is said to be fully load balanced with respect to security parameter n if the following properties hold:*

¹³ Recall a protocol has (communication) locality $\ell(n)$ if during the course of the protocol every party communicates with at most $\ell(n)$ other parties.

¹⁴ Looking ahead, pairwise computation will be used when emulating job-to-job communication, and will be sufficient when the original jobs protocol has *injective communication*, so that each job communicates with at most one other job per round.

¹⁵ With some canonical resolution for write conflicts. (In our constructions, the sets (i, j) will be disjoint).

- *Memory load balancing:* Let $\text{space}(\Pi)$ denote the total space complexity of protocol Π . For every constant $\delta > 0$, with all but negligible probability in n , every party P_j has space complexity

$$\text{space}(P_j) \leq \frac{(1 + \delta)}{N} \text{space}(\Pi) + \text{polylog}(n).$$

- *Online computation/communication load balancing:* For every constant $\delta > 0$, with all but negligible probability in n , the following holds at all times during the protocol: Let cc and $\text{cc}(P_j)$ denote the total communication complexity and communication complexity of party P_j , comp and $\text{comp}(P_j)$ denote the total computation complexity and computation complexity of party P_j , we have

$$\begin{aligned} \frac{(1 - \delta)}{N} \text{cc} - \text{polylog}(n) &\leq \text{cc}(P_j) \leq \frac{(1 + \delta)}{N} \text{cc} + \text{polylog}(n) \\ \frac{(1 - \delta)}{N} \text{comp} - \text{polylog}(n) &\leq \text{comp}(P_j) \leq \frac{(1 + \delta)}{N} \text{comp} + \text{polylog}(n). \end{aligned}$$

- *Per-round per-party efficiency:*¹⁶ Let A be an upper bound on the number of active jobs at each round in $\Pi_{\mathcal{J}}$. With all but negligible probability in n , the per-round per-party computation complexity is upper bounded by $\tilde{O}(1 + (A/N))$.

Theorem 4. Let $\Pi_{\mathcal{J}}$ be an M -job protocol with computation complexity comp and injective communication, realizing functionality $\mathcal{F}^{\text{Jobs}}$. Then there exists a fully load-balanced (Definition 4) $\tilde{O}(n)$ -party protocol $\Pi_{\mathcal{W}}$ with decomposable states (Definition 3) that realizes $\mathcal{F}$ with total computation $\tilde{O}(\text{comp})$, space complexity $\tilde{O}(M)$, and $\text{polylog}(n)$ locality. If $\Pi_{\mathcal{J}}$ satisfies oblivious security, so does $\Pi_{\mathcal{W}}$.

Idea of proof. Recall that in our construction of $\Pi_{\mathcal{W}}$ (in the introduction), at any point of the protocol execution, each job is assigned to a random worker¹⁷ and is stored in at most 2 workers. This is sufficient to imply memory load balancing by standard concentration and union bounds. Online computation/communication load balancing follows by observing that (i) the job-passing pattern is independent of the worker-job assignment, and (ii) jobs are passed frequently enough before accumulating large cost. This allows us to think of the execution as partitioned into “job chunks” each of which is assigned to a random worker, thus amenable to concentration bounds. The last load-balanced property follows again by the fact that each job is independently assigned to a random worker and that each job only performs $\text{polylog}(n)$ amount of work per round. To obtain locality, we consider a fixed low-degree communication network between workers, and pass messages using a load-balanced

¹⁶ We note that the last two properties are related but incomparable. The online load balancing property focuses on accumulated work, whereas the per-round per-party efficiency concerns upper bounds on per-round work, which is used to bound the required amount of parallelism to execute the protocol with efficient parallel time.

¹⁷ Technically, the initial job-worker assignment is only K -wise independent for $K = \log^3 n$. Nevertheless, this is sufficient for concentration bounds to go through.

routing algorithm. Load balancing of this modified scheme follows by similar, but more delicate analysis.

The resulting protocol has *decomposable state*, since parties' memory and computation are completely local to individual jobs, or pairs of jobs in the case of emulating job-to-job communication (since the starting jobs protocol has injective communication).

Step 3: From Oblivious to Malicious Security. Finally, we present a general transformation that produces an n -party MPC protocol securely realizing a functionality $\mathcal{F}$ against $(1/3 - \epsilon)n$ static corruptions, given any $\tilde{O}(n)$ -party protocol with decomposable states (see Definition 3) realizing the corresponding jobs-model functionality $\mathcal{F}^{\text{jobs}}$ with only *oblivious* security. This step can be viewed as a refinement and formalization of ideas from [KLST11, DKMS12, BGT13].

Theorem 5 (From Oblivious Security to Malicious Security).

Suppose there exists an $N \in \Theta(n \cdot \text{polylog}(n))$ -party oblivious protocol with decomposable state, realizing functionality $\mathcal{F}^{\text{jobs}}$ in space, computation, and (parallel) time complexity space, comp, time . Then for any constant $\epsilon > 0$ there exists an n -party MPC protocol (with error negligible in n) securely realizing the corresponding functionality $\mathcal{F}$ against $(1/3 - \epsilon)n$ static corruptions, with the following complexities (where each party is a PRAM with possibly many processors), given a one-time preprocessing phase with a single broadcast of $\tilde{O}(1)$ bits per party:

- *Per-party memory:* $\tilde{O}(\text{space}/n)$.
- *Total computation:* $\tilde{O}(\text{comp})$.
- *Time complexity:* $\tilde{O}(\text{time})$.

In addition, if the original protocol has $\tilde{O}(1)$ locality and is fully load-balanced (i.e., satisfying all properties of Definition 4), then the resulting protocol additionally possesses the following properties:

- *Communication locality* $\tilde{O}(1)$.
- *Online computation load balancing*, as in Definition 4(c).
- *Time complexity* $\tilde{O}(\text{time} \cdot \max\{1, \frac{A}{nk}\})$ when each party is limited to being a k -processor PRAM, where A denotes the maximum per-round per-party computation complexity of any party in the original oblivious-secure protocol.¹⁸

Idea of Proof. The compiler takes the following form: (1) First, parties collectively elect a large number of “good” committees, each of size $\text{polylog}(n)$, where “good” means each committee is composed of at least $2/3$ honest parties, and that parties are spread roughly evenly across committees. The one-time broadcast is used to reach full agreement on the first committee. (2) Each party verifiably secret shares his input among the corresponding committee C_i . (3) From this point on, the role of each party P_i in the original protocol will be emulated by the corresponding committee C_i via small-scale MPCs. Since committees are only

¹⁸ In particular, for our MPC for PRAMs protocol formed by combining Steps 1 and 2, the parameter A will correspond to the number of CPUs required in the evaluated PRAM Π , with polylog overhead.

size $\text{polylog}(n)$, the memory, computation, and time complexity overhead is small. Since parties are spread across committees, the protocol remains load balanced. Finally, by using a perfectly secure underlying MPC protocol (such as [BGW88]), the only information revealed corresponds directly to the “observable” properties (communication patterns, etc.), thus reducing directly to oblivious security (as per Definition 1).

References

- [AAK08] Baruch Awerbuch, Yossi Azar, and Rohit Khandekar. Fast load balancing via bounded best response. In *SODA 2008*, pages 314–322, 2008.
- [ACMR95] Micah Adler, Soumen Chakrabarti, Michael Mitzenmacher, and Lars Eilstrup Rasmussen. Parallel randomized load balancing (preliminary version). In *STOC 1995*, pages 238–247, 1995.
- [AJLA⁺12] Gilad Asharov, Abhishek Jain, Adriana López-Alt, Eran Tromer, Vinod Vaikuntanathan, and Daniel Wichs. Multiparty computation with low communication, computation and interaction via threshold fhe. In *EUROCRYPT*, pages 483–501, 2012.
- [BCP14a] Elette Boyle, Kai-Min Chung, and Rafael Pass. Large-scale secure computation. Cryptology ePrint Archive, Report 2014/404, 2014.
- [BCP14b] Elette Boyle, Kai-Min Chung, and Rafael Pass. Oblivious parallel ram. Cryptology ePrint Archive, Report 2014/594, 2014.
- [BGJK12] Elette Boyle, Shafi Goldwasser, Abhishek Jain, and Yael Taurman Kalai. Multiparty computation secure against continual memory leakage. In *STOC*, pages 1235–1254, 2012.
- [BGT13] Elette Boyle, Shafi Goldwasser, and Stefano Tessaro. Communication locality in secure multi-party computation - how to run sublinear algorithms in a distributed setting. In *TCC*, pages 356–376, 2013.
- [BGW88] Michael Ben-Or, Shafi Goldwasser, and Avi Wigderson. Completeness theorems for non-cryptographic fault-tolerant distributed computation (extended abstract). In *STOC*, pages 1–10, 1988.
- [BSGH13] Nicolas Braud-Santoni, Rachid Guerraoui, and Florian Huc. Fast byzantine agreement. In *PODC*, pages 57–64, 2013.
- [Can01] Ran Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *FOCS*, pages 136–145, 2001.
- [DI06] Ivan Damgård and Yuval Ishai. Scalable secure multiparty computation. In *CRYPTO*, pages 501–520, 2006.
- [DIK⁺08] Ivan Damgård, Yuval Ishai, Mikkel Krøigaard, Jesper Buus Nielsen, and Adam Smith. Scalable multiparty computation with nearly optimal work and resilience. In *CRYPTO*, pages 241–261, 2008.

- [DKMS12] Varsha Dani, Valerie King, Mahnush Movahedi, and Jared Saia. Breaking the $o(nm)$ bit barrier: Secure multiparty computation with a static adversary. *CoRR*, abs/1203.0289, 2012.
- [DMN11] Ivan Damgård, Sigurd Meldgaard, and Jesper Buus Nielsen. Perfectly secure oblivious ram without random oracles. In *TCC*, pages 144–163, 2011.
- [Gen09] Craig Gentry. Fully homomorphic encryption using ideal lattices. In *STOC*, pages 169–178, 2009.
- [GGH⁺13] Craig Gentry, Kenny A. Goldman, Shai Halevi, Charanjit S. Jutla, Mariana Raykova, and Daniel Wichs. Optimizing oram and using it efficiently for secure computation. In *Privacy Enhancing Technologies*, pages 1–18, 2013.
- [GKK⁺11] S. Dov Gordon, Jonathan Katz, Vladimir Kolesnikov, Tal Malkin, Mariana Raykova, and Yevgeniy Vahlis. Secure computation with sublinear amortized work. *IACR Cryptology ePrint Archive*, 2011:482, 2011.
- [GMW87] Oded Goldreich, Silvio Micali, and Avi Wigderson. How to play any mental game or a completeness theorem for protocols with honest majority. In *STOC*, pages 218–229, 1987.
- [GO96] Oded Goldreich and Rafail Ostrovsky. Software protection and simulation on oblivious rams. *J. ACM*, 43(3):431–473, 1996.
- [KLST11] Valerie King, Steven Lonargan, Jared Saia, and Amitabh Trehan. Load balanced scalable byzantine agreement through quorum building, with full information. In *ICDCN*, pages 203–214, 2011.
- [KSSV06] Valerie King, Jared Saia, Vishal Sanwalani, and Erik Vee. Scalable leader election. In *SODA*, pages 990–999, 2006.
- [LO13] Steve Lu and Rafail Ostrovsky. Distributed oblivious ram for secure two-party computation. In *TCC*, pages 377–396, 2013.
- [MPS02] Michael Mitzenmacher, Balaji Prabhakar, and Devavrat Shah. Load balancing with memory. In *(FOCS), 16-19 November 2002, Vancouver, BC, Canada, Proceedings*, pages 799–808, 2002.
- [MR98] S. Muthukrishnan and Rajmohan Rajaraman. An adversarial model for distributed dynamic load balancing. *SPAA '98*, pages 47–54, 1998.
- [MSS13] Steven Myers, Mona Sergi, and Abhi Shelat. Black-box proof of knowledge of plaintext and multiparty computation with low communication overhead. In *TCC*, pages 397–417, 2013.
- [OS97] Rafail Ostrovsky and Victor Shoup. Private information storage (extended abstract). In *STOC*, pages 294–303, 1997.
- [VB81] Leslie G. Valiant and Gordon J. Brebner. Universal schemes for parallel communication. In *STOC*, pages 263–277, 1981.
- [Yao86] Andrew Chi-Chih Yao. How to generate and exchange secrets (extended abstract). In *FOCS*, pages 162–167, 1986.
- [ZMS14] Mahdi Zamani, Mahnush Movahedi, and Jared Saia. Millions of millionaires: Multiparty computation in large networks. *Cryptology ePrint Archive*, Report 2014/149, 2014.

Succinct Garbling Schemes and Applications

Huijia Lin*

Rafael Pass†

October 20, 2014

Abstract

Assuming the existence of **iO** for $P/poly$ and one-way functions, we show how to *succinctly* garble bounded-space computations (BSC) M : the size of the garbled program (as well as the time needed to generate the garbling) only depends on the size and space (including the input and output) complexity of M , but not its running time. The key conceptual insight behind this construction is a method for using **iO** to “compress” a computation that can be performed piecemeal, without revealing anything about it.

As corollaries of our succinct garbling scheme, we demonstrate the following:

- functional encryption for BSC from **iO** for $P/poly$ and one-way functions;
- *reusable* succinct garbling schemes for BSC from **iO** for $P/poly$ and one-way functions;
- succinct **iO** for BSC from sub-exponentially-secure **iO** for $P/poly$ and sub-exponentially-secure one-way functions;
- (Perfect NIZK) SNARGS for bounded space and witness NP from sub-exponentially-secure **iO** for $P/poly$ and sub-exponentially-secure one-way functions.

Previously such primitives were only known to exist based on “knowledge-based” assumptions (such as SNARKs and/or differing-input obfuscation).

We finally demonstrate the first (non-succinct) **iO** for RAM programs (with bounded input and output lengths) with only *poly-logarithmic overhead* based on the existence of sub-exponentially-secure **iO** for $P/poly$ and sub-exponentially-secure one-way functions.

*University of California at Santa Barbara, Email: huijia@cs.ucsb.edu.

†Cornell University, Email: rafael@cs.cornell.edu. Work supported in part by a Alfred P. Sloan Fellowship, Microsoft New Faculty Fellowship, NSF Award CNS-1217821, NSF CAREER Award CCF-0746990, NSF Award CCF-1214844, AFOSR YIP Award FA9550-10-1-0093, and DARPA and AFRL under contract FA8750-11-2-0211. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the Defense Advanced Research Projects Agency or the US Government.

1 Introduction

Since the seminal work of Yao [Yao86] in the 1980s, *garbled circuits* and more generally *garbled programs* (such as e.g., garbled RAMs [LO13]) have been extensively studied. A garbling scheme consists of three procedures: a) *program garbling* method, b) an *input encoding* method, and c) an evaluation procedure. Given a program Π , the garbling method produces a “garbled” version $\tilde{\Pi}$ of Π as well as some key *key*. Next, given the key *key* and some input x , the input encoding method produces an encoding $\tilde{x}$ of x ; this encoding method is “efficient”—the time needed to encode x (and as a consequence also the length of $\tilde{x}$) does not depend on the complexity of Π . Finally, given a garbled program $\tilde{\Pi}$ and an encoded input $\tilde{x}$, the evaluation procedure evaluates $\Pi(x)$; additionally, given only the encoded program and inputs, $\tilde{\Pi}$, $\tilde{x}$, an attacker cannot learn more than just $\Pi(x)$. As a direct application, a garbling scheme enables a client to, in an “off-line” stage, garble a program Π , and next provide the garbled program $\tilde{\Pi}$ to a server. Later, in an on-line stage, when the input x on which the client wants to evaluate Π becomes known, it can *efficiently* and *without revealing neither Π nor x* ask the server to evaluate Π simply by sending it the encoding $\tilde{x}$ of x —in essence, garbling schemes provide a method for privately delegating computation (with an expensive off-line stage). Furthermore, on top of this direct application of garbling schemes, they have found numerous other applications (e.g., secure two-party computation [Yao86], multi-party computations [BMR90], reusable delegation of computation [GGP10], and so on and so forth.)

But a major deficiency of the all known garbling schemes (both of circuit and RAM garbling schemes) is that the size of the garbled program $\tilde{\Pi}$ (and thus also the time needed to generate it) grows linearly with (and in particular is lower-bounded by) the circuit-size/time-complexity of the underlying program Π . Thus, even if get a succinct description of Π (e.g., as a Turing-machine or RAM program), the size of $\tilde{\Pi}$ will be proportional to the running-time of Π .

1.1 Succinct Garbling Schemes

In this work we study *succinct garbling schemes* where the size, as well as the time required to generate, the garbled program $\tilde{\Pi}$ only grows *logarithmically* with the running-time of the underlying program Π (but polynomially in the size of it). We provide constructions of such succinct garbling schemes for general classes of computation (assuming the existence of *indistinguishability obfuscators* [BGI⁺01, GGH⁺13b] for appropriate classes of computation), and next demonstrate several new applications of succinct garbling schemes.

As an initial observation, we show that assuming the existence of “succinct” **iO** for some “nice” class $\mathcal{C}$ of computations [BGI⁺01, BCP14, ABG⁺13], there exists succinct garbling the same class, with the same complexity.

Theorem 1 (Initial observations—Informally stated). *Assume the existence of succinct indistinguishability obfuscators for a “nice” class of computations $\mathcal{C}$ and one-way function. Then, there exists a succinct garbling scheme for $\mathcal{C}$. (In particular, the size of the garbled program depends polynomially on the size of the program, the length of the input and outputs of the program, but only poly-logarithmically on its running-time and space.)*

Roughly speaking, the garbling of a program Π is an obfuscation a slightly modified program Π'_{key} that takes as input an authenticated encrypted input, decrypts and verifies the authenticity of the input (w.r.t the key *key*) and if it the input is valid simply runs Π on this input; the encoding of an input is simply an authenticated encryption of the input.¹

¹The overview oversimplifies. To prove this construction secure assuming only that the underlying obfuscation satisfies **iO** we need to rely on a particular method for authenticated encryption.

Let us remark here that the reason the above-mentioned construction is succinct is that we rely on the succinctness of the underlying **iO** (as e.g., in the definition of **iO** for Turing machine [BGI⁺01, BCP14, ABG⁺13].) As such it provides little into how to achieve succinctness “from-scratch”. Additionally, to date, the only constructions of succinct **iO** for Turing machines rely on “knowledge-based” assumptions—more specifically, the existence of *differing-input obfuscation* [BGI⁺01, BCP14, ABG⁺13]. Rather, we are here interested in basing succinct garbling schemes on some *hardness assumption*. (As we shall see shortly, doing this will also provide insight into constructions of succinct **iO** for Turing machines based on some hardness assumption.)

Our main result shows how to obtain succinct garbling schemes relying only on (non-succinct) **iO** for *circuits*—by now there are several constructions of **iO** [GGH⁺13b, PST14, GLSW14] for $P/poly$ based on specific hardness assumptions regarding graded encodings [GGH13a], and some of them are even falsifiable [PST14, GLSW14]. Our succinct garbling construction from **iO** for $P/poly$, however, only works for any *a-priori bounded-space* computation—in other words, the size of the garbled circuit depends polynomially on the space complexity of the computation, but is independent of its running-time.

Theorem 2 (Main Theorem—Informally stated). *Assume the existence of indistinguishability obfuscators for $P/poly$ and one-way functions. Then, for every polynomial p , there exists a succinct garbling scheme for all polynomial-time programs Π with space-complexity $p(\cdot)$. (In particular, the size of the garbled program depends polynomially on the size of Π , the length of the input and output of Π and the , and the space complexity $s(\cdot)$, but only logarithmically on Π ’s running-time.)*

The key conceptual insight behind our construction is a method for using **iO** to “compress” a computation that can be performed piecemeal, without revealing anything about it. More precisely, in a first step, we show how to obtain a “non-succinct” garbling of bounded-space Turing machines, and next, in a second step, we show how to use **iO** to compress this non-succinct garbling into a succinct one. (We believe that this compression technique may be of independent interest.)

Next, we use this main theorem to enable, among other things, bounded space (polynomial-time) computations in the context of a) functional encryption (FE), b) *resusable* succinct garbling schemes, c) **iO**, and d) succinct non-interactive arguments (SNARGs), assuming **iO** for $P/poly$ and one-way function (for some of these results with sub exponential security); prior to this paper, these primitives could only be constructed based on “knowledge-based” assumptions [GKP⁺13a, BCP14, ABG⁺13] or in the Random Oracle model [Mic00].

1.2 Succinct Garbling Schemes for Bounded-Space Computations

We turn to providing an overview of our construction of succinct garbling schemes for bounded space Turing machines. Our construction proceeds in two steps: we first construct a *non-succinct* garbling scheme, with the property that the garbled program consists of many “small pieces” that can be independently generated. Next, in a second step, we use indistinguishability obfuscation to “compress” the size of the garbled program, by releasing an obfuscated program that takes an index as input and generates the “piece” corresponding to that index. As a result, the final garbled program (namely the obfuscated program) is small and can be efficiently computed, and it is only at the evaluation time that the underlying non-succinct garbled program gets “decompressed” (by running the obfuscated program on all possible indexes to recover it).

A Non-succinct Garbling Scheme: Construction Overview Let us first outline a *non-succinct garbling scheme* for Turing machines² based on any one-way function. Note that a “trivial” approach for achieving this is to simply transform any polynomial-time Turing machine into a polynomial-size circuits and then garble the circuit. While our construction in essence relies on this principle, we provide a construction that uses as a *black-box* a garbling scheme for “small” fixed-sized circuits (and thus this construction may be of independent interest). More precisely, we will rely on the existence of a garbling scheme for circuits (as in [Yao86]) satisfying an additional useful property: the key *key* can be generated independently of the circuit to be garbled (more precisely, we now have a key generation algorithm *Gen* that outputs the key *key*; next, both the encoding and garbling methods receive this key as input); furthermore, we additionally require that encoded inputs can be simulated without knowledge of the circuit to be garbled. We refer to such schemes as *garbling schemes with independent key generation* and note that Yao’s original scheme (which can be based on one-way functions) satisfies this property. Our non-succinct garbling scheme now proceeds as follows for a Turing machine Π with bounded space complexity $s(\cdot)$ and running-time $T(\cdot)$ and inputs of length n . We construct a “chain” of $T(n)$ garbled circuits that evaluate Π step by step. More precisely, we first generate keys $key_1, \dots, key_{T(n)}$ for the $T(n)$ garbled circuits. The i^{th} garbled circuit (which is computed using key key_i) takes as input some state of Π and computes the next state (ie., the state after one computation step); if the next state is a final state, it outputs the outputs generated by Π , otherwise its outputs an *encoding* of this new state using key key_{i+1} . (Note that after $T(n)$ steps we are guaranteed to get to a final state and thus this process is well-defined.)

The input encoding method simply encodes the initial state of Π with input x using the key_1 , and to evaluate the garbled program we simply sequentially evaluate each garbled circuit, using the encodings generated in the previous one as inputs to the next one, and finally outputting the output generated.

A Non-succinct Garbling Scheme: Proof Overview To show that this construction is a secure (non-succinct) garbling scheme we need to exhibit a simulator that given just the output $y = \Pi(x)$ of the program Π on input x and *the number of steps t^* taken by $\Pi(x)$* can simulate the encoded input and program. (The reason we provide the simulation with the number of steps t^* is that we desire a garbling scheme with a “per-instance efficiency”—that is, the evaluation time is polynomial in the actual running-time t^* and not just the worst-case running-time. To achieve such “per-instance efficiency” requires leaking the running-time, which is why the simulator gets access to it.) Towards this, we start by simulating the t^{*th} garbled circuit with the output being set to y ; this simulation generates an encoded input $\widetilde{\text{conf}}_{t^*-1}$ and a garbled program $\widetilde{\Pi}_{t^*}$ (if the simulation is valid, $\widetilde{\text{conf}}_{t^*-1}$ is supposed to be an encoding of the configuration conf_t of the TM after t steps of computation). We then iteratively in descending order simulate the i^{th} ($i < t^*$) garbled circuits $\widetilde{\Pi}_i$ with the output being set to $\widetilde{\text{conf}}_{i+1}$ generated in the previously simulated garbled circuit. We finally simulate the remaining $i > t^*$ garbled circuits $\widetilde{\Pi}_i$ with the output being set to some arbitrary output in the range of the circuit (e.g., simply y), and release conf_1 and $(\widetilde{\Pi}_1, \dots, \widetilde{\Pi}_{T(n)})$. (Note that the fact that we simulate the i^{th} ($i > t^*$) garbled circuit with the output being set to some arbitrary value is fine since encoded inputs to those circuits are not released.)

To prove indistinguishability of this simulation, we consider a sequence of hybrid experiments

²The choice of a Turing machine as the model of computation is arbitrary and the solution work no matter what the model of bounded-space computation (e.g., Turing machine, RAM, PRAM etc) as long as a computation can be decomposed into a sequence of sequential computation steps operating on the memory.

$H_0, \dots, H_{T(n)}$, where in H_j the first j garbled circuits are simulated, and the remaining $T(n) - j$ garbled circuits are honestly generated. To “stitch together” the simulated circuits with the honestly generated ones, the j^{th} garbled circuit is simulated using as output, an honest encoding $\widehat{\text{conf}}_j$ of the actual configuration conf_j of the TM after t steps. It follows from the security of the garbling scheme (and the fact that a only single encoded input is released for circuit $j + 1$) that hybrids H_j and H_{j+1} are indistinguishable and thus also H_0 (i.e., the real experiment) and $H_{T(n)}$ (i.e., the simulation).

Let us finally remark a useful property of the above-mentioned simulation. Due to the fact that we rely on a garbling scheme with *independent key generation*, each garbled circuit can in fact be *independently simulated*—recall that the independent key generation property guarantees that encoded inputs can be simulated without knowledge of the circuit to be computed and thus all simulated encoded inputs $\text{conf}_1, \dots, \text{conf}_{T(n)}$ can be generated in an initial step. Next, the garbled circuits can be simulated in any order.

The Succinct Garbling Scheme: Construction Overview Let us now turn to making this garbling scheme succinct. The key idea is to, instead of releasing the actual garbled circuits, release an obfuscation of the *randomized* program that generates the garbled circuits. More precisely, we release an indistinguishability obfuscation of a program $\Pi^{x,s,s'}(t)$ where $x \in \{0,1\}^n$ is an input to Π , $t \in [T(n)]$ is a “time-step” of Π and s is the seed for a PRF F : $\Pi^{x,s,s'}(t)$ generates and outputs the t^{th} garbled circuit in the non-succinct garbling of Π using pseudo-random coins generated by the PRF with seed s and s' . More specifically, it uses $F(s, t)$ and $F(s, t + 1)$ as randomness to generate key_t and key_{t+1} (recall that the functionality of the t^{th} garbled circuit may depend on key_{t+1}), and uses $F(s', t)$ as randomness for generating the t^{th} garbled circuit.

Now, the new succinct garbled program is the obfuscated program $\Lambda \xleftarrow{\$} i\mathcal{O}(\Pi^{x,s,s'})$, and the encoding $\hat{x}$ of x remains the same as before, except that now generated using pseudo-random coins $F(s, 1)$. Given such a garbled pair Λ and $\hat{x}$, one can compute the output by first generating the entire non-succinct garbled program by computing Λ on every time step t , and evaluating the non-succinct garbling with $\hat{x}$.

The Succinct Garbling Scheme: Proof Overview Given that the new succinct garbled program Λ produces “pieces” of the non-succinct garbled program, the natural idea for simulating the succinct garbled program is to obfuscate a program that produces “pieces” of the simulated non-succinct garbled program. The above-mentioned “independent simulation” property of the non-succinct garbled program enable exactly this.

More precisely, given an output y and the running-time t^* of $\Pi(x)$, the simulator outputs the obfuscation $\tilde{\Lambda}$ of a program $\tilde{\Pi}^{y,t^*,s,s'}$ that on input t :

- outputs the simulation of the t^{th} garbled circuit (as described in the simulation of the non-succinct garbling scheme, and using y as the output of the whole program) using $F(s, t)$ and $F(s, t + 1)$ as randomness to generate $\widehat{\text{conf}}_t$ and $\widehat{\text{conf}}_{t+1}$, and $F(s', t)$ as randomness to generate the simulated garbled circuit;

The encoding of input $\tilde{x}$ is simulated as the non-succinct garbling scheme does, but using pseudo-random coins $F(s, 1)$. (Note that we here strongly rely on the independent simulation property of the non-succinct garbled program constructed above, which in turn relies on the independent key generation property of the underlying garbled circuit.)

It is not hard to see that this simulation works if the obfuscation is virtually black-box secure, as the entire truth tables of the two programs $\Pi^{x,s,s'}$ and $\tilde{\Pi}^{y,t^*,s,s'}$ are indistinguishable when the

hardwired PRF keys s, s' are chosen at random. Our goal, however, is to show that assuming indistinguishability obfuscation suffices. Towards doing this, as above, we consider a sequence of hybrid experiments $H'_0, \dots, H'_{T(n)}$ that obfuscates a sequence of programs that “morph” gradually from Π to $\tilde{\Pi}$. In particular, the program $\tilde{\Pi}_j^{x,s,s'}$ obfuscated in H'_j produces a non-succinct hybrid garbled program as in hybrid H_j in the proof of the non-succinct garbling scheme, except that pseudo-random coins generated using seeds s, s' are used instead of truly random coins. (More specifically, for the first j inputs, $\tilde{\Pi}_j$ produces simulated garbled circuits (as in H_j), and for the rest inputs, it produces honestly generated garbled circuits.)

To prove indistinguishability of any two consecutive hybrids H'_j and H'_{j+1} , we finally rely on the punctured program technique of Sahai and Waters [SW14], to replace pseudo-random coins $F(s, j+1)$, $F(s', j+1)$ for generating the $j+1^{\text{th}}$ simulated garbled circuit with truly random coins, and can then rely on the indistinguishability of the simulation of the $j+1^{\text{th}}$ garbled circuit to conclude the indistinguishability of neighboring hybrids.

A note on the efficiency of the Garbling Scheme Note that the time (and size) of the garbled program depends polynomially on the space bound and the length of the inputs and outputs and only poly-logarithmically in the upper bound on the running-time of the program. The evaluation time, on the other hand, is linear in the time complexity of the program Π —*no matter what model of computation we rely on (e.g., TM, RAM, or PRAM)*, and again polynomial in the space and input/output lengths.

1.3 Applications of (Succinct) Garbling Schemes

We now turn to presenting applications of garbling schemes. While our focus here is applications of *succinct* garbling schemes, as we shall explain shortly, our results are general and lead to interesting corollaries also when relying on non succinct schemes.

Application 1: Succinct Randomized Encoding Recall that a *randomized encoding* [IK02, AIK04] is a method to, given an input x and a function f , (randomly) encode $f(x)$ in a way that leaks nothing beyond just $f(x)$. As is well known, garbling schemes imply randomized encoding: the randomized encoding of f, x is simply the garbling of f and the encoding of x . Whereas previous works on randomized encoding have focused on encoding methods that can be performed by low-depth computations [AIK04], when relying on succinct garbling schemes, we obtain a new type of a *succinct randomized encoding* where the encoding can be produced much more *efficiently* than computing f —in particular, the time needed to produce the encoding only grows poly-logarithmically with the time-complexity of f . We next show how such succinct randomized encodings are useful for applications. For notational convenience, we describe these applications using garbling scheme, but it should be appreciated that for these application succinct randomized encodings actually suffice.

Applications 2: FE, reusable garbling schemes, secure computations We observe that in contexts such as secure computation [GMW87] and functional encryption [SW05, O’N10, BSW12], to evaluate a function f on an input x , it suffices to evaluate the *randomized* function that computes a garbled program of f and an encoding of the input (recall that by the security of the garbling scheme this reveal no more than the output of the function).³ Thus, by plugging-in

³That is, we are computing a randomized encoding of $f(x)$.

our construction of succinct garbling schemes for bounded-space computations (BSC) into earlier constructions of secure computation or *randomized* functional encryption [GJKS]⁴, we directly obtain, assuming **iO** for $P/poly$ and one-way functions, a randomized functional encryption for BSC, and secure computation protocols for bounded-space programs, where the communication complexity grows logarithmically with the running-time of the program to be evaluated. We additionally observe that by combining our construction of functional encryption for BSC with previous results [CIJ⁺13, GKP⁺13b]—[CIJ⁺13] showed that function encryption schemes with indistinguishability-based security implies ones with simulation-based security, which further implies reusable garbling schemes by [GKP⁺13b]—directly yields a construction of *reusable* succinct garbling schemes for BSC from **iO** for $P/poly$ and one-way functions. Summarizing,

Theorem 3 (Informally stated). *Assume the existence of indistinguishability obfuscators for $P/poly$ and one-way function. Then, for every polynomial p , there exists secure constructions of the following primitives which handle computations all polynomial-time computations with space-complexity $s(\cdot)$:*

- reusable succinct garbling;
- functional encryption where the size of the secret key for a function is independent of its running time;
- secure computation where the communication complexity of the protocol is independent of the running-time of the program.

Let us mention that prior to this paper, the second of these primitives could only be constructed based on “knowledge-based” assumptions (such as SNARKs and extractable witness encryption [GKP⁺13a] or differing-input obfuscation [BCP14, ABG⁺13]), and the third one based on incomparable assumptions (namely, FHE—it is unknown whether **iO** and one-way functions imply FHE).

Applications 3: Succinct **iO and SNARGs** Recall that in Theorem 1 we demonstrated how to use succinct **iO** to get succinct garbling schemes—in fact, the construction works for any “nice”. We now show a converse of this result: assuming sub-exponentially-secure **iO** for $P/poly$, the existence of a sub-exponentially-secure succinct garbling scheme for a “nice”⁵ class of algorithms yields an **iO** for the same class of algorithms *with the same complexity*. We first observe that if we had an appropriate notion of **iO** for randomized functionalities, then we could rely on the same argument as in the context of secure computation and functional encryption—instead of evaluating a function f , simply compute the randomized functionality that computes the garbled version of f and the encoded input. We observe that a notion recently considered by Canetti, Lin, Tessaro and Vaikuntanathan [CLTV14] (which can be achieved based on sub-exponentially-secure **iO** and one-way functions) suffices for our purposes as long as the garbling scheme is sub-exponentially secure.

Combined with Theorem 2, this yields succinct **iO** for BSC from sub-exponentially-secure **iO** for $P/poly$ and sub-exponentially secure one-way functions. Plugging in this result into the Perfect NIZK construction of Sahai-Waters [SW14] directly yields a construction of (Perfect NIZK)

⁴The reason we need randomized functional encryption is to be able to compute the randomized garbling function.

⁵Here by “nice”, we mean that $\mathcal{C}$ (1) contains algorithms with a-priori polynomially-bounded input and output lengths, (2) is closed under composition with polynomial-sized circuits, and (3) algorithms contained in $\mathcal{C}_\lambda$ is also contained in $\mathcal{C}_{\lambda'}$, with $\lambda' \geq \lambda$. The last requirement is a technicality in order to enable applying a cryptographic algorithm on an algorithm from $\mathcal{C}_\lambda$ with a bigger security parameter λ' .

SNARGS for bounded-space NP from sub-exponentially-secure **iO** for $P/poly$ and sub-exponentially-secure one-way functions. Summarizing,

Theorem 4 (Informally state). *Assume the existence of sub-exponentially-secure indistinguishability obfuscators for $P/poly$ and sub-exponentially-secure one-way function. Then, for every polynomial p , there exists*

- *iO for all polynomial-time computations with space-complexity $s(\cdot)$;*
- *(Perfect NIZK) SNARGS (with adaptive soundness)⁶ for all languages in NP that can be decided by a non-deterministic polynomial-time Turing machines with space-complexity $s(\cdot)$.*

These primitives were only known to exist based on “knowledge-based” assumptions [BCP14, ABG⁺13, BP13] or in the Random Oracle Model [Mic00].

Application 4: iO for RAM with poly-logarithmic overhead We finally observe that the above observation that in the context of **iO** it suffices to evaluate the garbling of the function instead of directly evaluating the functions is useful also if relying on non-succinct garbling schemes. In particular, by relying on the garbled RAM constructions of [LO13, GHL⁺14] we directly obtain as a corollary,

- **iO** for RAM programs with bounded input and output lengths, where the size of the obfuscated program only grows quasi-linearly with the RAM complexity of the program, assuming **iO** for $P/poly$ and one-way functions, both with sub-exponential security.

There is one subtle detail that needs to be dealt with to obtain the above corollary. If simply relying on any **iO** in the above construction, then the use of this underlying primitive could blow-up the running-time. (If we rely on an **iO** for circuits with quasi-linear overhead then we are fine, but this seems to require stronger assumptions.) Rather, we rely on an observation from Gentry, Halevi, Raykova and Wichs [GHRW14]: the garbled RAM constructions of [LO13, GHL⁺14] satisfy a nice “bit-wise compactness” property, where each bit of the garbled circuit can be independently generated by a “small” circuit of size depending quasi-linearly in the input and output lengths and only poly-logarithmically in the running time.⁷ Thus, (inspired by [GHRW14],) instead of obfuscating the program that generates the whole garbled circuit in one shot, we simply obfuscate *many* “small” programs, each of which generates one bit of the garbled circuit. (Note that the resulting **iO** is not succinct: the size of the obfuscated program depends on the size of the garbled RAM).

Let us remark that a similar construction was recently provided in [GHRW14]; the difference between our construction and theirs is that they propose to obfuscate only a single “small” program that will generate all bits in the garbled RAM, whereas our **iO** consists of the obfuscation of many “small” programs, each of which generates only a single bit in the garbled RAM. But, the authors of [GHRW14] simply conjecture the security of their construction; in contrast, we prove it secure assuming that the underlying obfuscator satisfies sub-exponentially secure **iO**.

⁶The Perfect NIZK construction of [SW14] only satisfies non-adaptive soundness. But by a standard complexity leveraging trick, it can be made to satisfy adaptive soundness. Since we anyway assume sub-exponential security of the **iO** this comes at no cost for us.

⁷More precisely, the size of the small circuit is $\tilde{O}(|R| + n + m) \times \text{poly}(\lambda, \log T)$, where R is the RAM machine under consideration, n and m are its input and output lengths, and T is its running time.

2 Preliminaries

Let $\mathcal{N}$ denote the set of positive integers, and $[n]$ denote the set $\{1, 2, \dots, n\}$. We denote by PPT probabilistic polynomial time Turing machines. The term **negligible** is used for denoting functions that are (asymptotically) smaller than one over any polynomial. More precisely, a function $\nu(\cdot)$ from non-negative integers to reals is called **negligible** if for every constant $c > 0$ and all sufficiently large n , it holds that $\nu(n) < n^{-c}$.

2.1 Models of Computation

In this work we will consider different models of computation. Below we define formally different classes of algorithms; we will start by defining classes of deterministic algorithms of fixed polynomial size, and then move to define classes of randomized algorithms and classes of algorithms of arbitrary polynomial size.

Classes of deterministic algorithms of fixed polynomial size.

Polynomial-time Circuits. For every polynomial D , the class $\text{CIR}[D] = \{\mathcal{C}_\lambda\}$ of include all deterministic circuits of size at most $D(\lambda)$.

NC¹ Circuits. For every constant c and polynomial D , the class $\text{NC}_c[D] = \{\mathcal{C}_\lambda\}$ of polynomial-sized circuits of depth $c \log \lambda$ include all deterministic circuits of size $D(\lambda)$ and depth at most $c \log \lambda$.

Exponential-time Turing Machines. We consider a canonical representation of Turing machines $M = (M', n, m, S, T)$ with $|n| = |m| = |S| = |T| = \lambda$ and $n, m \leq S \leq T$; M takes input x of length n , and runs $M'(x)$ using S space for at most T steps, and finally outputs the first m bits of the output of M' . (If $M'(x)$ does not halt in time T or requires more than S space, M outputs $\perp$.) In other words, given the description M of a Turing machine in this representation, one can efficiently read off its bound parameters denoted as $(M.n, M.m, M.S, M.T)$.

Now we define the class of exponential time Turing machines. For every polynomial D , the class $\text{TM}[D] = \{\mathcal{M}_\lambda\}$ includes all deterministic Turing machines Π_M containing the canonical representation of a Turing machine M of size $D(\lambda)$; $\Pi_M(x, t)$ takes input x and t of length $M.n$ and λ respectively, and runs $M(x)$ for t steps, and finally outputs what M returns.

Remark: Note that machine $\Pi_M(x, t)$ on any input terminates in $t < 2^\lambda$, and hence its output is well-defined. Furthermore, for any two Turing machines M_1 and M_2 , they have the same functionality if and only if they produce identical outputs and run for the same number of steps for every input x . This property is utilized when defining and constructing indistinguishability obfuscation for Turing machines, as in previous work [BCP14].

Exponential-time RAM Machines. We consider a canonical representation of RAM machines $R = (R', n, m, S, T)$ identical to the canonical representation of Turing machines above.

For every polynomial D , the class $\text{RAM}[D] = \{\mathcal{R}_\lambda\}$ of polynomial-sized RAM machines include all deterministic RAM machines Π_R , defined as Π_M above for Turing machines, except that the Turing machine M is replace with a RAM machine R .

Classes of randomized algorithms: The above defined classes contain only deterministic algorithms. We define analogously these classes for their corresponding randomized algorithms. Let $\mathcal{X}[D]$ be any class defined above, we denote by $\text{r}\mathcal{X}[D]$ the corresponding class of randomized algorithms. For example $\text{rCIR}[D]$ denote all randomized circuits of size $D(\lambda)$, and $\text{rTM}[D]$ denote all randomized Turing machines of size $D(\lambda)$.

Classes of (arbitrary) polynomial-sized algorithms: The above defined classes consist of algorithms of a fixed polynomial D description size. We define corresponding classes of arbitrary polynomial size. Let $\mathcal{X}[D]$ be any class defined above, we simply denote by $\mathcal{X} = \cup_{\text{poly } D} \mathcal{X}[D]$ the corresponding class of algorithms of arbitrary polynomial size. For instance, CIR and rCIR denotes all deterministic and randomized polynomial-sized circuits, and TM denotes all polynomial-sized Turing machines.

In the rest of the paper, when we write a family of algorithms $\{AL_\lambda\} \in \mathcal{X}$, we mean $\{AL_\lambda\} \in \mathcal{X}[D]$ for some polynomial D . This means, the size of the family of algorithms is bounded by some polynomial. Below, for convenience of notation, when $\mathcal{X}$ is a class of algorithms of arbitrary polynomial size, we write $AL \in \mathcal{X}_\lambda$ as a short hand for $\{AL_\lambda\} \in \{\mathcal{X}_\lambda\}$.

Classes of well-formed algorithms: In the rest of the preliminary, we define various cryptographic primitives. In order to avoid repeating the definitions for different classes of machines, we provide definitions for general classes of algorithms $\{\mathcal{AL}_\lambda\}$ that can be instantiated with specific classes defined above. In particular, we will work with classes of algorithms that are **well-formed**, satisfying the following properties:

1. For every $AL \in \mathcal{AL}_\lambda$, and input x , AL on input x terminates in 2^λ steps. Note that this also implies that AL has bounded input and output lengths.
2. the size of every ensemble of algorithms $\{AL_\lambda\} \in \{\mathcal{AL}_\lambda\}$ is bounded by some polynomial D in λ , and
3. given the description of an algorithm $AL \in \mathcal{AL}_\lambda$, one can efficiently read off the bound parameters $AL.n, AL.m, AL.S, AL.T$.

All above defined algorithm classes are well-formed. Below, we denote by $T_{AL}(x)$ the running time of AL on input x , and T_{AL} the worst case running time of AL . Note that well-formed algorithm classes are not necessarily efficient; for instance the class of polynomial-sized Turing machines TM contain Turing machines that run for exponential time. In order to define cryptographic primitives for only polynomial-time algorithms, we will use the notation $\text{ALG}^T = \{\mathcal{AL}_\lambda^T\}$ to denote the class of algorithms in $\text{ALG} = \{\mathcal{AL}_\lambda\}$ that run in time $T(\lambda)$ (in particular, these with $AL_\lambda.T < T(\lambda)$).

In the rest of the paper, all algorithm classes are well-formed.

2.2 Garbling Scheme

Definition 1 (Garbling Scheme). *A Garbling scheme $\mathcal{GS}$ for a class of (well-formed) deterministic algorithms $\{\mathcal{AL}_\lambda\}_{\lambda \in \mathbb{N}}$ consists of algorithms $\mathcal{GS} = (\text{Garb}, \text{Encode}, \text{Eval})$ satisfying the following properties:*

Syntax: For every $\lambda \in \mathbb{N}$, $AL \in \mathcal{AL}_\lambda$ and input x ,

- **Garb** is probabilistic and on input $(1^\lambda, AL)$ outputs a pair $(\hat{AL}, \text{key})$.⁸

⁸(Note that as the algorithm class is well-formed, **Garb** implicitly has all bound parameters of AL .)

- *Encode* is deterministic and on input $(\mathbf{key}, x)$ outputs $\hat{x}$.
- *Eval* is deterministic and on input $(\hat{AL}, \hat{x})$ produced by *Garb*, *Encode* outputs y .

Correctness: For every polynomial T and every family of algorithms $\{AL_\lambda\} \in \{\mathcal{AL}_\lambda^T\}$ and sequence of inputs $\{x_\lambda\}$, There exists a negligible function μ , such that, for every $\lambda \in \mathbb{N}$, $AL = AL_\lambda$, $x = x_\lambda$,

$$\Pr[(\hat{AL}, \mathbf{key}) \xleftarrow{\$} \text{Garb}(1^\lambda, AL), \hat{x} \xleftarrow{\$} \text{Encode}(\mathbf{key}, x) : \text{Eval}(\hat{AL}, \hat{x}) \neq AL(x)] \leq \mu(\lambda)$$

Definition 2 (Security of a Garbling Scheme). We say that a Garbling scheme $\mathcal{GS}$ for a class of deterministic algorithms $\{\mathcal{AL}_\lambda\}_{\lambda \in \mathbb{N}}$ is secure if the following holds.

Security: There exists a uniform machine *Sim*, such that, for every non-uniform PPT distinguisher $\mathcal{D}$, every polynomial T' , every sequence of algorithms $\{AL_\lambda\} \in \{\mathcal{AL}_\lambda^{T'}\}$, and sequence of inputs $\{x_\lambda\}$ where $x_\lambda \in \{0, 1\}^{AL_\lambda \cdot n}$, there exists a negligible function μ , such that, for every $\lambda \in \mathbb{N}$, $AL = AL_\lambda$, $x = x_\lambda$ the following holds:

$$\left| \Pr[(\hat{AL}, \mathbf{key}) \xleftarrow{\$} \text{Garb}(1^\lambda, AL), \hat{x} \xleftarrow{\$} \text{Encode}(\mathbf{key}, x) : \mathcal{D}(\hat{AL}, \hat{x}) = 1] \right. \\ \left. - \Pr[(\tilde{AL}, \tilde{x}) \xleftarrow{\$} \text{Sim}(1^\lambda, 1^{|x|}, 1^{|AL|}, (n, m, S, T), T_{AL}(x), AL(x)) : \mathcal{D}(\tilde{AL}, \tilde{x}) = 1] \right| \leq \mu(\lambda)$$

where $(n, m, S, T) = (AL.n, AL.m, AL.S, AL.T)$ and *Sim* runs in time $\text{poly}(\lambda, T'(\lambda))$. Moreover, μ is called the **distinguishing gap**

Furthermore, we say that $\mathcal{GS}$ is δ -**indistinguishable** if the above security condition holds with a distinguishing gap μ bounded by δ . Especially, $\mathcal{GS}$ is **sub-exponentially indistinguishable** if $\mu(\lambda)$ is bounded by $2^{-\lambda^\epsilon}$ for a constant ϵ .

We note that the sub-exponentially indistinguishability defined above is weaker than usual sub-exponential hardness assumptions in that the distinguishing gap only need to be small for PPT distinguisher, rather than sub-exponential time distinguishers.

We remark that in the above definition, simulator *Sim* receives many inputs, meaning that, a garbled pair $\hat{AL}, \hat{x}$ reveals nothing but the following: The output $AL(x)$, instance running time $T_{AL}(x)$, input length $|x|$ and machine size $|AL|$, together with various parameters (n, m, S, T) of AL . We note that the leakage of the instance running time is necessary in order to achieve instance-based efficiency (see efficiency guarantees below). The leakage of $|AL|$ can be avoided by padding machines if an upper bound on their size is known. The leakage of parameters (n, m, S, T) can be avoided by setting them to 2^λ ; see Remark 1 for more details. In particular, when the algorithms are circuits, inputs to the simulation algorithm can be simplified to $(1^\lambda, 1^{|x|}, 1^{|C|}, AL(x))$, since all bound parameters n, m, S, T can be set to 2^λ .

Efficiency Guarantees. we proceed to describe the efficiency requirements for garbling schemes. When considering only circuit classes, all algorithms *Garb*, *Encode*, *Eval* should be polynomial time machines, that is, the complexity of *Garb*, *Eval* scales with the size of the circuit $|C|$, and that of *Encode* with the input length $|x|$. However, when considering general algorithm classes, since the description size $|AL|$ could be much smaller than the running time $AL.T$, or even other parameters $AL.S, AL.n, AL.m$, there could be different variants of efficiency guarantees, depending on what parameters the complexity of the algorithms depends on. Below we define different variants.

Definition 3 (Different Levels of Efficiency of Garbling Schemes). *We say that a garbling scheme $\mathcal{GS}$ for a class of deterministic algorithms $\{\mathcal{AL}_\lambda\}_{\lambda \in \mathbb{N}}$ has succinctness or I/O / space / time-dependent complexity if the following holds.*

Optimal efficiency: *There exists universal polynomials $p_{\text{Garb}}, p_{\text{Encode}}, p_{\text{Eval}}$, such that, for every $\lambda \in \mathbb{N}$, $AL \in \mathcal{AL}_\lambda$ and input $x \in \{0, 1\}^{AL.n}$,*

- $(\hat{A}, \text{key}) \xleftarrow{\$} \text{Garb}(1^\lambda, AL)$ runs in time $p_{\text{Garb}}(\lambda, |AL|, AL.m)$,⁹
- $\hat{x} = \text{Encode}(\text{key}, x)$ runs in time $p_{\text{Encode}}(\lambda, |x|, AL.m)$, and
- $y = \text{Eval}(\hat{A}, \hat{x})$ runs in time $p_{\text{Eval}}(\lambda, |AL|, |x|, AL.m) \times T_{AL}(x)$, with overwhelming probability over the random coins of **Garb**. We note that **Eval** has instance-based efficiency.

I/O-dependent complexity: *The above efficiency conditions hold with $p_{\text{Garb}}, p_{\text{Encode}}, p_{\text{Eval}}$ taking $AL.n$ as additional parameters.*

Space-dependent complexity: *The above efficiency conditions hold with $p_{\text{Garb}}, p_{\text{Encode}}, p_{\text{Eval}}$ taking $AL.S$ as an additional parameter.*

Linear-time-dependent complexity: *The above efficiency conditions hold with $p_{\text{Garb}}, p_{\text{Encode}}$ taking $AL.T$ as an additional parameter and depending (quasi-)linearly in $AL.T$, and the running time of **Eval** is bounded by $p_{\text{Eval}}(\lambda, |AL|, |x|)AL.T$.*

Furthermore, we say that the garbling scheme $\mathcal{GS}$ has **succinct input encodings** if the encoding algorithm $\text{Encode}(\text{key}, x)$ runs in time $p_{\text{Encode}}(1^\lambda, |x|)$.

We say that a garbling scheme is “succinct” if its complexity depends only poly-logarithmically on the time bound. Thus a scheme with space-dependent complexity is succinct for a class of algorithms whose space usage is bounded by a fixed polynomial.

On the dependency on the length of the output. Note that in the optimal efficiency defined above, the complexity of the algorithms depends on the length of their respective inputs and the bound on their output lengths $AL.m$. We argue that this is necessary. This is because that the garbling of an algorithm $\hat{A}L$ together with an encoding of an input $\hat{x}$ encodes the output $AL(x)$, while leaking nothing beyond $AL(x)$. ($\hat{A}L, \hat{x}$ is a randomized encoding of AL, x .) Then, assuming the existence of pseudorandom generators G , the total size of the garbled function $\hat{G}$ and encoded input $\hat{x}$ must be at least the length of the output of the function. Otherwise, the simulator can “compress” random strings with overwhelming probability, which is a contradiction. Therefore, we allow the complexity of the algorithms to depend on the length of the output in optimal efficiency.

Garbling Schemes for Specific Algorithm Classes. Next we instantiate the above definition of garbling scheme for general algorithm classed with concrete classes.

Definition 4 (Garbling Scheme for Polynomial-sized Circuits). *A triplet of algorithms $\mathcal{GS}_{\text{CIR}} = (\text{Garb}_{\text{CIR}}, \text{Encode}_{\text{CIR}}, \text{Eval}_{\text{CIR}})$ is a garbling scheme (with linear-time-dependent complexity) for polynomial sized circuits if it is a garbling scheme for class **CIR** (with linear-time-dependent complexity).*

We note that in the case of circuits, succinctness means the complexity scales polynomially in $|C|$, whereas linear-time-dependency means the complexity scales linearly with $|C|$.

⁹Note that the running time of **Garb** and similarly other algorithms that takes AL as an input, implicitly depends logarithmically on the time bound of AL , as its description contains the time bound $AL.T$.

Definition 5 (Garbling Schemes for Polynomial Time Turing Machines). *A triplet $\mathcal{GS}_{\text{TM}} = (\text{Garb}_{\text{TM}}, \text{Encode}_{\text{TM}}, \text{Eval}_{\text{TM}})$ of algorithms is a garbling scheme with optimal efficiency or I/O- / space- / linear-time-dependent complexity (and succinct input encodings) for Turing machines, if it is a garbling scheme for class TM, with the same level of efficiency.*

Different efficiency requirements impose qualitatively different restrictions. In this work, we will construct a garbling scheme for Turing machines with space-dependent complexity assuming indistinguishability obfuscation for circuits. The construction of garbling scheme from **IO** for Turing machines, sketched in the introduction, has I/O-dependent complexity. On the other hand, we show that a scheme with is impossible; in particular, the complexity of the scheme must scale with the bound on the output length.

Definition 6 (Garbling Schemes for Polynomial Time RAM Machines). *A triplet $\mathcal{GS}_{\text{RAM}} = (\text{Garb}_{\text{RAM}}, \text{Encode}_{\text{RAM}}, \text{Eval}_{\text{RAM}})$ of algorithms is a garbling scheme for polynomial-time RAM machines with optimal efficiency or I/O- / space- / linear-time- dependent-complexity, (and succinct input encodings), if it is a garbling scheme for class RAM, with the same level of efficiency.*

Recently, the works by [LO13, GHL⁺14] give construction of a garbling scheme for RAM machines with linear-time-dependent complexity and succinct input encodings, assuming only one-way functions.

Garbled Circuits with Independent Key Generation. In this work, we will make use of a garbling scheme for circuits with a special structural property. In Definition 4, the key **key** for garbling inputs is generated depending on the circuit (by $\text{Garb}(1^\lambda, C)$); the special property of a circuit garbling scheme is that the **key** can be generated depending only on the length of the input $1^{|x|}$ and the security parameter, which implies that the garbled inputs $\hat{x}$ can also be generated depending only on the plain input x and the security parameter λ , independently of the circuit—we call this *independent key generation*.

Definition 7 (Garbling Scheme for Circuits with Independent Key Generation). *A Garbling scheme $\mathcal{GS} = (\text{Garb}, \text{Encode}, \text{Eval})$ for a deterministic circuit class $\{\mathcal{C}_\lambda\}_{\lambda \in \mathbb{N}}$ has **independent key generation** if the following holds: For every $\lambda \in \mathbb{N}$, and every $C \in \mathcal{C}_\lambda$,*

- *The algorithm **Garb** on input $(1^\lambda, C)$ invokes first **key** $\xleftarrow{\$} \text{Gen}(1^\lambda, 1^{|x|})$ and then $\hat{C} \xleftarrow{\$} \text{Gb}(\text{key}, C)$, where **Gen** and **Gb** are all PPT algorithms.*
- *The security condition holds w.r.t. a simulator **Sim** that on input $(1^\lambda, 1^{|x|}, 1^{|C|}, T_C(x), C(x))$ invokes first $(\tilde{x}, \text{st}) \xleftarrow{\$} \text{Sim} \cdot \text{Gen}(1^\lambda, 1^{|x|})$ and then $\tilde{C} \xleftarrow{\$} \text{Sim} \cdot \text{Gb}((1^\lambda, 1^{|x|}, 1^{|C|}, C(x), \text{st}))$, where **Sim-Gen** and **Sim-Gb** are all uniform PPT algorithms.*

It is easy to check that many known circuit garbling schemes, in particular the construction by Yao [Yao86], has independent key generation.

Proposition 1. *Assume the existence of one-way functions that are hard to invert in Γ time. Then, there exists a garbling scheme $\mathcal{GS}_{\text{CIR}}$ for polynomial-sized circuits with independent key generation that is $\Gamma^{-\varepsilon}$ -indistinguishable for some constant $\varepsilon \in (0, 1)$.*

2.3 Indistinguishability Obfuscation

We recall the definition of indistinguishability obfuscation, adapting to arbitrary classes of algorithms. As before, we first define the syntax, correctness and security of **iO**, and then discuss about different efficiency guarantees.

Definition 8 (Indistinguishability Obfuscator ($i\mathcal{O}$)). *A uniform machine $i\mathcal{O}$ is a indistinguishability obfuscator for a class of deterministic algorithms $\{\mathcal{AL}_\lambda\}_{\lambda \in \mathbb{N}}$, if the following conditions are satisfied:*

Correctness: *For all security parameters $\lambda \in \mathbb{N}$, for all $AL \in \mathcal{AL}_\lambda$, for all input x , we have that*

$$\Pr[AL' \leftarrow i\mathcal{O}(1^\lambda, AL) : AL'(x) = AL(x)] = 1$$

Security: *For every polynomial T , every non-uniform PPT samplable distribution $\mathcal{D}$ over the support $\{\mathcal{AL}_\lambda^T \times \mathcal{AL}_\lambda^T \times \{0, 1\}^{\text{poly}(\lambda)}\}$, and adversary $\mathcal{A}$, there is a negligible function μ , such that, for sufficiently large $\lambda \in \mathbb{N}$, if*

$$\Pr[(AL_1, AL_2, z) \leftarrow \mathcal{D}(1^\lambda) : \forall x, AL_1(x) = AL_2(x), T_{AL'}(x) = T_{AL}(x), \\ (|AL|, AL.n, AL.m, AL.S, AL.T) = (|AL'|, AL'.n, AL'.m, AL'.S, AL'.T)] > 1 - \mu(\lambda)$$

Then,

$$\left| \Pr[(AL_1, AL_2, z) \xleftarrow{\$} \mathcal{D}(1^\lambda) : \mathcal{A}(i\mathcal{O}(1^\lambda, AL_1), z)] \right. \\ \left. - \Pr[(AL_1, AL_2, z) \xleftarrow{\$} \mathcal{D}(1^\lambda) : \mathcal{A}(i\mathcal{O}(1^\lambda, AL_2), z)] \right| \leq \mu(\lambda)$$

*where μ is called the **distinguishing gap** for $\mathcal{D}$ and $\mathcal{A}$.*

*Furthermore, we say that $i\mathcal{O}$ is δ -indistinguishable if the above security condition holds with a distinguishing gap μ bounded by δ . Especially, $i\mathcal{O}$ is **sub-exponentially indistinguishable** if $\mu(\lambda)$ is bounded by $2^{-\lambda^\epsilon}$ for a constant ϵ .*

Note that in the security guarantee above, the distribution $\mathcal{D}$ samples algorithms AL_1, AL_2 that has the same functionality, and matching bound parameters. This means, an obfuscated machine “reveals” the functionality (as desired) and these bound parameters. We remark that the leakage of the latter is without loss of generality: In the case of circuits, all bound parameters are set to 2^λ . In the case of other algorithm classes, say Turing and RAM machines. If an **iO** scheme ensures that one parameter, say $AL.S$, is not revealed, one can simply consider a representation that always sets that parameter to 2^λ ; then security definition automatically ensures privacy of that parameter. See Remark 1 for more details.

Definition 9 (Different Levels of Efficiency of **iO**). *We say that an indistinguishability obfuscator $i\mathcal{O}$ of a class of algorithms $\{\mathcal{AL}_\lambda\}$ has **optimal efficiency**, if there is a universal polynomial p such that for every $\lambda \in \mathbb{N}$, and every $AL \in \mathcal{AL}_\lambda$, $i\mathcal{O}(1^\lambda, AL)$ runs in time $p(\lambda, |AL|)$.*

*Additionally, we say that $i\mathcal{O}$ has **input- / space- / linear-time- dependent complexity**, if $i\mathcal{O}(1^\lambda, AL)$ runs in time $\text{poly}(\lambda, |AL|, AL.n)$ / $\text{poly}(\lambda, |AL|, AL.S)$ / $\text{poly}(\lambda, |AL|)AL.T$.*

We note that unlike the case of garbling schemes, the optimal efficiency of an **iO** scheme does not need to depend on the length of the output. Loosely speaking, the stems from the fact that indistinguishability-based security does not require “programing” outputs, which is the case in simulation-based security for garbling.

iO for Specific Algorithm Classes. We recall the definition of **iO** for polynomial-sized circuits, NC^1 [BGI⁺01]; and give definitions of **iO** for polynomial time Turing machines [BCP14] and RAM machines with different efficiency guarantees.

Definition 10 (Indistinguishability Obfuscator for Poly-sized Circuits and NC^1). *A uniform PPT machine $i\mathcal{O}_{\text{CIR}}(\cdot, \cdot)$ is an indistinguishability obfuscator for polynomial-sized circuits if it is an indistinguishability obfuscator for CIR with optimal efficiency.*

A uniform PPT machine $i\mathcal{O}_{\text{NC}^1}(\cdot, \cdot, \cdot)$ is an indistinguishability obfuscator for NC^1 circuits if for all constants $c \in \mathcal{N}$, $i\mathcal{O}_{\text{NC}^1}(c, \cdot, \cdot)$ is an indistinguishability obfuscator for NC_c with optimal efficiency.

Definition 11 (IO for Turing Machines). *A uniform machine $i\mathcal{O}_{\text{TM}}(\cdot, \cdot)$ is a indistinguishability obfuscator for polynomial-time Turing machines, with optimal efficiency or input- / space-dependent complexity, if it is an indistinguishability obfuscator for the class TM with the same efficiency.*

Recently, the works by [BCP14, ABG⁺13] give constructions of **iO** for Turing machines¹⁰ with input-dependent complexity assuming FHE, differing-input obfuscation for circuits, and P-certificates [CLP13]; furthermore, the dependency on input lengths can be removed—leading to a scheme with optimal efficiency—if assuming SNARK instead of P-certificates.

Definition 12 (**iO** for RAM Machines). *A uniform machine $i\mathcal{O}_{\text{TM}}(\cdot, \cdot)$ is a indistinguishability obfuscator for polynomial-time Turing machines, with optimal efficiency or linear-time-dependent complexity, if it is an indistinguishability obfuscator for the class RAM with the same efficiency.*

Remark 1 (Explicit v.s. Implicit Bound Parameters). *In the above definitions of Garbling Scheme and **iO** for general algorithms, we considered a canonical representation of algorithms AL that gives information of various bound parameters of the algorithm, specifically, the size $|AL|$, bound on input and output lengths $AL.n$, $AL.m$, space complexity $AL.S$, and time complexity $AL.T$. This representation allows us to define, in a unified way, different garbling and **iO** schemes that depend on different subsets of parameters. For instance,*

- *The Garbling and **iO** schemes for TM that we construct in Section 3 and 6 (from **iO** and sub-exp **iO** for circuits respectively) has complexity $\text{poly}(|AL|, AL.S, \log(AL.T))$. (In particular, the size of the garbled TM and obfuscated TM is of this order.)*
- *The garbling scheme for TM constructed (from **iO** for TM) sketched in the introduction has complexity $\text{poly}(|AL|, AL.n, AL.m, \log(AL.T))$.*
- *The garbling scheme for RAM from one-way functions by [LO13, GH⁺14] has complexity scales polynomially in $(|AL|, AL.n, AL.m)$ and quasi-linearly in $AL.T$. This construction leads to an **iO** for RAM (from sub-exp **iO** for circuits) of the same complexity in 6.*

*By using the canonical representation, our general definition allows the garbling or **iO** scheme to depend on any subset of parameters flexibly. Naturally, if a scheme depends on a subset of parameters, the resulting garbled or obfuscated machines may “leak” these parameters (in the above three examples above, the size of the garbled or obfuscated machines leaks the parameters they depend on); thus, the security definitions must reflect this “leakage” correspondingly. The general security definitions 2 and 8 captures this by allowing leakage of all parameters $|AL|, AL.n, AL.m, AL.S, AL.T$.*

¹⁰Their works actually realize the stronger notion of differing-input, or extractability, obfuscation for Turing machines

However, this seems to “overshoot”, as if a specific scheme does not depend on a particular parameter (e.g. $AL.S$), then this parameter should be kept private. This can be easily achieved, by simply considering an algorithm representation that always set that parameter to 2^λ (e.g. $AL.S = 2^\lambda$).

2.4 Puncturable Pseudo-Random Functions

We recall the definition of puncturable pseudo-random functions (PRF) from [SW14]. Since in this work, we only uses puncturing at one point, the definition below is restricted to puncturing only at one point instead of at a polynomially many points.

Definition (Puncturable PRFs). *A puncturable family of PRFs is given by a triple of uniform PPT machines $(\text{PRF}\cdot\text{Gen}, \text{PRF}\cdot\text{Punc}, F)$, and a pair of computable functions $n(\cdot)$ and $m(\cdot)$, satisfying the following conditions:*

Correctness. *For all outputs K of $\text{PRF}\cdot\text{Gen}(1^\lambda)$, all points $i \in \{0, 1\}^{n(\lambda)}$, and $K(-i) = \text{PRF}\cdot\text{Punc}(K, i)$, we have that $F(K(-i), x) = F(K, x)$ for all $x \neq i$.*

Pseudorandom at punctured point. *For every PPT adversary $(\mathcal{A}_1, \mathcal{A}_2)$, there is a negligible function μ , such that in an experiment where $\mathcal{A}_1(1^\lambda)$ outputs a point $i \in \{0, 1\}^{n(\lambda)}$ and a state σ , $K \xleftarrow{\$} \text{PRF}\cdot\text{Gen}(1^\lambda)$ and $K(i) = \text{PRF}\cdot\text{Punc}(K, i)$, the following holds*

$$|\Pr[\mathcal{A}_2(\sigma, K(i), i, F(K, i)) = 1] - \Pr[\mathcal{A}_2(\sigma, K(i), i, U_{m(\lambda)}) = 1]| \leq \mu(\lambda)$$

where μ is called the **distinguishing gap** for $(\mathcal{A}_1, \mathcal{A}_2)$.

Furthermore, we say that the puncturable PRF is **δ -indistinguishable** if the above pseudorandom property holds with a distinguishing gap μ bounded by δ . Especially, the puncturable PRF is **sub-exponentially indistinguishable** if $\mu(\lambda)$ is bounded by $2^{-\lambda^\epsilon}$ for a constant ϵ .

As observed by [BW13, BGI14, KPTZ13], the GGM tree-based construction of PRFs [GGM86] from pseudorandom generators (PRGs) yields puncturable PRFs. Furthermore, it is easy to see that if the PRG underlying the GGM construction is sub-exponentially hard (and this can in turn be built from sub-exponentially hard OWFs), then the resulting puncturable PRF is sub-exponentially pseudo-random.

3 A Succinct Garbling Scheme for BSTM

In this section, we construct a garbling scheme for the class of Turing machines TM with space-dependent complexity. Thus when the space complexity of the TM is bounded, it yields a succinct scheme. We will see in the next section that our construction for Turing machines directly applies to general bounded space computation.

Theorem 5. *Assuming the existence of IO for circuits and one-way functions. There exists a garbling scheme for TM with space-dependent complexity.*

Towards this, we proceed in two steps: In the first step, we construct a *non-succinct* garbling scheme for TM, which satisfies the correctness and security requirements of Definition 1 and 2, except that the garbling and evaluation algorithms can run in time polynomial in both the time and space complexity, $M.T$ and $M.S$, of the garbled Turing machine M (as well as the simulation algorithm); the produced garbled Turing machine is of size in the same order. In the second step,

we show how to reduce the complexity to depend only on the space complexity $M.S$, leading to a garbling scheme with space-dependent complexity. Since in this section, only the space and time bound parameters matter, we will simply write S and T as $M.S$ and $M.T$, and we use the notion D to represent the description size of M .

3.1 A Non-Succinct Garbling Scheme

Overview. The execution of a Turing machine M consists of a sequence of steps, where each step t depends on the description of the machine M and its current configuration conf_t , and produces the next configuration conf_{t+1} . In the Turing machine model, each step takes constant time, independent of the size of the Turing machine and its configuration. However, each step can be implemented using a circuit $\text{Next}^{D,S}$ that on input (M, conf_t) with $|M| \leq D, |\text{conf}_t| \leq S$, outputs the next configuration conf_{t+1} —we call this circuit the “universal next-step circuit”. The size of the circuit is a fixed polynomial p_{Next} in the size of the machine and the configuration, that is, $p_{\text{Next}}(D, S)$. The whole execution of $M(x)$ can be carried out by performing at most T evaluations of $\text{Next}^{D,S}(M, \cdot)$, producing a chain of configurations denoted by,

$\text{CONFIG}(M, x) = (T^*, \text{conf}_1, \dots, \text{conf}_T, \text{conf}_{T+1})$, where $T^* = T_M(x)$, conf_1 is the initial configuration with input x $\{\text{conf}_1, \dots, \text{conf}_{T^*-1}, \text{conf}_{T^*}\}$ are the sequence of configurations until $M(x)$ halts (conf_t is the configuration before the t^{th} step starts), and $\{\text{conf}_{T^*}, \dots, \text{conf}_{T+1}\}$ are simply set to the output $y = M(x)$.

We note that the initial configuration conf_1 can be derived efficiently from x , conf_{T^*} is called the final configuration, which can be efficiently recognized and from which an output y can be extracted efficiently.

When succinctness is not required, the natural idea to garble a T -step Turing machine computation of $M(x)$ is to produce a chain of T garbled circuits $(\hat{\mathbf{C}}_1, \dots, \hat{\mathbf{C}}_T)$, for evaluating the next step circuit $\text{Next}^{D,S}(M, \cdot)$ for M . The t^{th} circuit $\mathbf{C}_t$ is designated to compute from the t^{th} configuration conf_t (as input) to the next conf_{t+1} ; if the produced conf_{t+1} is a final configuration, then it simply outputs the output y ; otherwise, to enable the evaluation of the next garbled circuit $\hat{\mathbf{C}}_{t+1}$, it translates conf_{t+1} into the corresponding garbled inputs $\widehat{\text{conf}}_{t+1}$ for $\hat{\mathbf{C}}_{t+1}$ —we call $\mathbf{C}_t$ the t^{th} *step-circuit*. Then evaluation propagates and the intermediate configurations of the execution of M on x is implicitly computed one by one, until it reaches the final configuration, in which case, an output is produced explicitly (without translating into the garbled inputs of the next garbled circuit). Since each computation step is garbled, and all intermediate configurations, except from the final output y , are “encrypted” as garbled inputs, the entire chain of garbled circuits can be simulated given only the output y .

Finally, we note that each step-circuit $\mathbf{C}_t$ evaluates $\text{Next}^{D,S}(M, \cdot)$ and has the capability of garbling an input for the next garbled circuit $\hat{\mathbf{C}}_t$; this can only be achieved if the circuit garbling scheme has independent key generation, which ensures that the input garbling can be done independently of the circuit garbling, and only takes time polynomial in the length of the input (rather than, in the size of the circuit).

Our Non-Succinct Garbling Scheme. We now describe formally our non-succinct garbling scheme $\mathcal{GS}_{ns} = (\text{Garb}_{ns}, \text{Encode}_{ns}, \text{Eval}_{ns})$. We rely on a garbling scheme for polynomial-sized circuits with independent key generation.

- Let $\mathcal{GS}_{\text{CIR}} = (\text{Garb}_{\text{CIR}}, \text{Encode}_{\text{CIR}}, \text{Eval}_{\text{CIR}})$ be a garbling scheme for polynomial-sized circuits, and Sim_{CIR} the simulation algorithm. We require $\mathcal{GS}_{\text{CIR}}$ to have independent key generation, that is, $\text{Garb}_{\text{CIR}} = (\text{Gen}_{\text{CIR}}, \text{Gb}_{\text{CIR}})$, and $\text{Sim}_{\text{CIR}} = (\text{Sim} \cdot \text{Gen}_{\text{CIR}}, \text{Sim} \cdot \text{Gb}_{\text{CIR}})$ as described in Definition 7.

Let $\text{Next}^{D,S}$ be the universal next step circuit for machine of size at most D and space complexity at most S ; it has a fixed polynomial size $p_{\text{Next}}(D, S)$ and can be generated efficiently given D and S . For every λ and $M \in \text{TM}_\lambda$, our scheme proceeds as follows:

The garbling algorithm $\text{Garb}_{ns}(1^\lambda, M)$:

Let $S = M.S$, $T = M.T$ and $D = |M|$.

Sample $2T$ sufficiently long random strings $\alpha_1, \dots, \alpha_t$ and $\beta_1, \dots, \beta_t$; produce a chain of T garbled circuits using Garb_{CIR} by running the following program for every $t \in [T]$.

Program $\mathbf{P}^{\lambda, S, M}(t ; (\alpha_t, \alpha_{t+1}, \beta_t))$:

1. *Generate the key key_{t+1} for the next garbled circuit:*
If $t < T$, compute the key for the $t + 1^{\text{th}}$ garbled circuit $\text{key}_{t+1} = \text{Gen}_{\text{CIR}}(1^\lambda, 1^S; \alpha_{t+1})$ using randomness α_{t+1} . (Note that key_t is generated for inputs of length S .)
2. *Prepare the step-circuit $\mathbf{C}_t$:*
 Step_t on a S -bit input conf_t (i) compute $\text{conf}_{t+1} = \text{Next}^{D,S}(M, \text{conf}_t)$; (ii) if conf_{t+1} is a final configuration, simply outputs the output y contained in it¹¹; (iii) otherwise, translate conf_{t+1} to the garbled inputs of the $t + 1^{\text{th}}$ garbled circuit, by computing $\widehat{\text{conf}}_{t+1} = \text{Encode}_{\text{CIR}}(\text{key}_{t+1}, \text{conf}_{t+1})$.
3. *Garble the step-circuit $\mathbf{C}_t$:*
Compute the key using randomness α_t , $\text{key}_t = \text{Gen}_{\text{CIR}}(1^\lambda, 1^S; \alpha_t)$, and garble $\mathbf{C}_t$ using randomness β_t , $\widehat{\mathbf{C}}_t = \text{Gb}_{\text{CIR}}(\text{key}_t, \mathbf{C}_t; \beta_t)$,
4. *Output $\widehat{\mathbf{C}}_t$.*

Generate **key** as follows: Compute the key for the first garbled circuit using randomness α_1 , $\text{key}_1 = \text{Gen}_{\text{CIR}}(1^\lambda, 1^S; \alpha_1)$; set **key** = $\text{key}_1 \parallel 1^S$.

Finally, output $\hat{M} = (\widehat{\mathbf{C}}_1, \dots, \widehat{\mathbf{C}}_T), \mathbf{key}$.

The encoding algorithm $\text{Encode}_{ns}(\mathbf{key}, x)$: Let $\text{conf}_1 \in \{0, 1\}^S$ be the initial configuration of M with input x ; compute $\hat{x} = \text{conf}_1 = \text{Encode}_{\text{CIR}}(\text{key}_1, \text{conf}_1)$.

The evaluation algorithm $\text{Eval}_{ns}(\hat{M}, \hat{x})$: Evaluate the chain of garbled circuits $\hat{M} = (\widehat{\mathbf{C}}_1, \dots, \widehat{\mathbf{C}}_T)$ in sequence in T iterations: In iteration t , compute $z = \text{Eval}_{\text{CIR}}(\widehat{\mathbf{C}}_t, \widehat{\text{conf}}_t)$; if z is the garbled inputs $\widehat{\text{conf}}_{t+1}$ for the next garbled circuit $\widehat{\mathbf{C}}_{t+1}$, proceed to the next iteration; otherwise, terminate and output $y = z$.

Next, we proceed to show that $\mathcal{GS}_{ns}$ is a non-succinct garbling scheme for TM.

Efficiency. We summarize the complexity of different algorithms of the non-succinct scheme. It is easy to see that for any Turing machine M with $D = |M|$, $S = M.S$ and $T = M.T$, the garbling algorithm Garb_{ns} runs in time $\text{poly}(\lambda, D, S) \times T$, and produces a garbling machine of size in the same order. Thus the garbling scheme is non-succinct. On the other hand, the encoding and evaluation algorithms Encode_{ns} and Eval_{ns} are all deterministic polynomial time algorithms. Finally, the simulation run in time $\text{poly}(\lambda, D, S) \times T$ as the garbling algorithm.

¹¹Pad y with 0 if it is not long enough

Correctness. We show that for every polynomial T' , every sequence of algorithms $\{M = M_\lambda\} \in \{\text{TM}_\lambda^{T'}\}$, and sequence of inputs $\{x = x_\lambda\}$ where $x_\lambda \in \{0, 1\}^{M.n}$, there exists a negligible function μ , such that,

$$\Pr[(\mathbf{key}, \hat{M}) \xleftarrow{\$} \text{Garb}_{ns}(1^\lambda, M), \hat{x} = \text{Encode}_{ns}(\mathbf{key}, x) : \text{Eval}_{ns}(\hat{M}, \hat{x}) \neq M(x)] \leq \mu(\lambda)$$

Let $\text{CONFIG}(M, x) = (T^*, \text{conf}_1, \dots, \text{conf}_T, \text{conf}_{T+1})$ be the sequence of configurations generated in the computation of $M(x)$, where $T \leq T'(\lambda)$. It follows from the correctness of the circuit garbling scheme Garb_{CIR} that with overwhelming probability (over the randomness of Garb_{ns}), the following is true: (1) for every $t < T^*$, the garbled circuit $\hat{\mathbf{C}}_t$, if given the garbled input $\widehat{\text{conf}}_t$ corresponding to conf_t , computes the correct garbled inputs $\widehat{\text{conf}}_{t+1}$ corresponding to conf_{t+1} , and (2) for $t = T^*$, the garbled circuit $\hat{\mathbf{C}}_{T^*}$, if given the garbled input $\widehat{\text{conf}}_{T^*-1}$ corresponding to conf_{T^*-1} , produces the correct output y . (Note that the evaluation procedure terminates after T^* iterations and circuits $\hat{\mathbf{C}}_t$ for $t > T^*$ are never evaluated). Then since the garbled input $\hat{x}$ equals to the garbled initial configuration $\widehat{\text{conf}}_1$, by conditions (1) and (2), the evaluation procedure produces the correct output with overwhelming probability.

Security. Fix any polynomial T' , any sequence of algorithms $\{M = M_\lambda\} \in \{\text{TM}_\lambda^{T'}\}$, and any sequence of inputs $\{x = x_\lambda\}$ where $x_\lambda \in \{0, 1\}^{M.n}$. Towards showing the security of $\mathcal{GS}_{ns}$, we construct a simulation algorithm Sim_{ns} , and show that the following two ensembles are indistinguishable: For convenience of notation, we suppress the appearance of $M.n$ and $M.m$ as input to Sim .

$$\left\{ \text{real}_{ns}(1^\lambda, M, x) \right\} = \left\{ (\hat{M}, \mathbf{key}) \xleftarrow{\$} \text{Garb}_{ns}(1^\lambda, M), \hat{x} = \text{Encode}_{ns}(\mathbf{key}, x) : (\hat{M}, \hat{x}) \right\}_\lambda \quad (1)$$

$$\left\{ \text{simu}_{ns}(1^\lambda, M, x) \right\} = \left\{ (\tilde{M}, \tilde{x}) \xleftarrow{\$} \text{Sim}_{ns}(1^\lambda, 1^{|x|}, 1^{|M|}, S, T, T_M(x), M(x)) : (\tilde{M}, \tilde{x}) \right\}_\lambda \quad (2)$$

Below we describe the simulation algorithm. Observe that the garbled machine $\hat{M}$ consists of T garbled circuits $(\hat{\mathbf{C}}_1, \dots, \hat{\mathbf{C}}_T)$ and the garbled input $\hat{x}$ is simply the garbled input of the initial configuration conf_0 (corresponding to x) for the first garbled circuit $\hat{\mathbf{C}}_1$. Naturally, to simulate them, the algorithm Sim_{ns} needs to utilize the simulation algorithm $\text{Sim}_{\text{CIR}} = (\text{Sim}\cdot\text{Gen}_{\text{CIR}}, \text{Sim}\cdot\text{Gb}_{\text{CIR}})$ of the circuit garbling scheme, which requires knowing the output of each garbled circuit. In a real evaluation with $\hat{M}, \hat{x}$, the output of the $(T^*)^{\text{th}}$ garbled circuit is $y = M(x)$, the output of the garbled circuits $t < T^*$ is the garbled input $\widehat{\text{conf}}_{t+1}$ for next garbled circuit $t + 1$, and the garbled circuits $t > T^*$ are not evaluated, but for which y is a valid output. Thus, in the simulation, garbled circuits $t = T^*, \dots, T$ can be simulated using output y ; whereas garbled circuits $t = 1, \dots, T^* - 1$ will be simulated using the *simulated garbled inputs* for circuit $t + 1$. More precisely,

The simulation algorithm $\text{Sim}_{ns}(1^\lambda, 1^{|x|}, 1^{|M|}, S, T, T^* = T_M(x), y = M(x))$:

Sample $2T$ sufficiently long random strings $\alpha_1, \dots, \alpha_T, \beta_1, \dots, \beta_T$. Simulate the chain of garbled circuits by running the following program for every $t \in [T]$.

Program $\mathbf{Q}^{\lambda, S, |M|, T^*, y}(t ; (\alpha_t, \alpha_{t+1}, \beta_t))$:

1. Prepare the output out_t for the t^{th} simulated circuit $\hat{\mathbf{C}}_t$:

If $t \geq T^*$, $\text{out}_t = y$. Otherwise, if $t < T^*$, set the output as the garbled input for the next garbled circuits, that is, $\text{out}_t = \widehat{\text{conf}}_{t+1}$ computed from $(\text{conf}_{t+1}, \mathbf{st}_{t+1}) = \text{Sim}\cdot\text{Gen}_{\text{CIR}}(1^\lambda, 1^S ; \alpha_{t+1})$ using randomness α_{t+1} .

Approved for Public Release; Distribution Unlimited.

2. *Simulate the t^{th} step-circuit $\tilde{\mathbf{C}}_t$:*

Given the output out_t , simulate the t^{th} garbled circuit $\tilde{\mathbf{C}}_t$ by computing first $(\widetilde{\text{conf}}_t, \mathbf{st}_t) = \text{Sim} \cdot \text{Gen}_{\text{CIR}}(1^\lambda, 1^S; \alpha_t)$ and then $\tilde{\mathbf{C}}_t = \text{Sim} \cdot \text{Gb}_{\text{CIR}}(1^\lambda, 1^S, 1^q, out_t, \mathbf{st}_t; \beta_t)$, using randomness α_t, β_t where $q = q(\lambda, S)$ is the size of the circuit $\mathbf{C}_t$.

3. *Output $\tilde{\mathbf{C}}_t$.*

Simulate the garbled input $\tilde{x}$ by computing again $(\widetilde{\text{conf}}_1, \mathbf{st}_1) = \text{Sim} \cdot \text{Gen}_{\text{CIR}}(1^\lambda, 1^S; \alpha_1)$ using randomness α_1 , and setting $\tilde{x} = \widetilde{\text{conf}}_1$.

Finally, output $(\tilde{M} = (\tilde{\mathbf{C}}_1, \dots, \tilde{\mathbf{C}}_T), \tilde{x})$.

Towards showing the indistinguishability between honestly generated garbling $(\hat{M}, \hat{x})$ and the simulation $(\tilde{M}, \tilde{x})$, we will consider a sequence of hybrids $\text{hyb}_{ns}^0, \dots, \text{hyb}_{ns}^T$, where hyb_{ns}^0 samples $(\hat{M}, \hat{x})$ honestly, while hyb_{ns}^T generates the simulated garbling $(\tilde{M}, \tilde{x})$. In every intermediate hybrid hyb_{ns}^γ , a hybrid simulator HSim_{ns}^γ is invoked, producing a pair $(\tilde{M}_\gamma, \tilde{x}_\gamma)$. At a high-level, the γ^{th} hybrid simulator on input $(1^\lambda, M, x)$ simulate the first $\gamma - 1$ garbled circuits using the program $\mathbf{Q}$, generates the last $T - \gamma$ garbled circuits honestly using the program $\mathbf{P}$, and simulates the γ^{th} garbled circuits using the program $\mathbf{R}$ described below, which “stitches” together the first $\gamma - 1$ simulated circuits with the last $T - \gamma$ honest circuits into a chain that evaluates to the correct output. More precisely, we will denote by

$\text{COMBINE}[(P_1, S_1), \cdot, (P_\ell, S_\ell)]$ a merged circuit that on input x in the domain X , computes $P_j(x)$ if $x \in S_j$, where $S_1, \dots, S_\ell$ is a partition of the domain X .

The hybrid simulation algorithm $\text{HSim}_{ns}^\gamma(1^\lambda, M, x)$ for $\gamma = 0, \dots, T$:

Compute $T^* = T_M(x)$ and $y = M(x)$, and the intermediate configuration $\text{conf}_{\gamma+1}$ as defined by $\text{CONFIG}(M, x)$.

Sample $2T$ sufficiently long random strings $\{\alpha_t, \beta_t\}_{t \in [T]}$. Simulate the chain of garbled circuits by running the following program for every $t \in [T]$, which combines programs $\mathbf{P}$, $\mathbf{Q}$ and $\mathbf{R}$ as below.

Program $\mathbf{M}^\gamma = \text{COMBINE}[(\mathbf{Q}, [\gamma - 1]), (\mathbf{R}, \{\gamma\}), (\mathbf{P}, [\gamma + 1, T])](t; (\alpha_t, \alpha_{t+1}, \beta_t))$:

- If $t \leq \gamma - 1$, compute $\tilde{\mathbf{C}}_t = \mathbf{Q}^{\lambda, S, |M|, T^*, y}(t; (\alpha_t, \alpha_{t+1}, \beta_t))$; output $\tilde{\mathbf{C}}_t$.
- If $t \geq \gamma + 1$, compute $\hat{\mathbf{C}}_t = \mathbf{P}^{\lambda, S, M}(t; (\alpha_t, \alpha_{t+1}, \beta_t))$; output $\hat{\mathbf{C}}_t$.
- If $t = \gamma$, compute $\tilde{\mathbf{C}}_t = \mathbf{R}^{\lambda, S, \text{conf}_{\gamma+1}}(\gamma; (\alpha_\gamma, \alpha_{\gamma+1}, \beta_\gamma))$ define as follow:

1. *Prepare the output out_γ of the simulated γ^{th} circuit $\tilde{\mathbf{C}}_t$:*

Set the output out_γ to y if $\text{conf}_{\gamma+1}$ is a final configuration. Otherwise, the output should be the garbled input corresponding to $\text{conf}_{\gamma+1}$ for the next garbled circuit; since the $\gamma + 1^{\text{th}}$ circuit is generated honestly, we compute $out_\gamma = \widetilde{\text{conf}}_{\gamma+1}$ by first computing $\text{key}_{\gamma+1} = \text{Gen}_{\text{CIR}}(1^\lambda, 1^S; \alpha_{\gamma+1})$, and then encoding $\widetilde{\text{conf}}_{\gamma+1} = \text{Encode}_{\text{CIR}}(\text{key}_{\gamma+1}, \text{conf}_{\gamma+1})$.

(Note that the difference between program $\mathbf{Q}$ and $\mathbf{R}$ is that the former prepares the output out_γ using simulated garbled input $\widetilde{\text{conf}}_{t+1}$, whereas the latter using honestly generated garbled input $\widetilde{\text{conf}}_{\gamma+1}$.)

2. *Simulate the γ^{th} circuit $\tilde{\mathbf{C}}_t$:*

Given the output out_γ , simulate the γ^{th} garbled circuit $\tilde{\mathbf{C}}_\gamma$ by computing $(\widetilde{\text{conf}}_\gamma, \mathbf{st}_\gamma) = \text{Sim} \cdot \text{Gen}_{\text{CIR}}(1^\lambda, 1^S; \alpha_\gamma)$ and $\tilde{\mathbf{C}}_t = \text{Sim} \cdot \text{Gb}_{\text{CIR}}(1^\lambda, 1^S, 1^q, out_\gamma, \mathbf{st}_\gamma; \beta_\gamma)$, where $q = q(\lambda, S)$ is the size of the circuit $\mathbf{C}_t$.

If $\gamma > 0$, simulate the garbled input $\tilde{x}_\gamma$ as Sim_{ns} does. Otherwise, if $\gamma = 0$, generate the garbled input $\tilde{x}_0$ honestly as in Garb_{ns} and Encode_{ns} .

Finally, output $(\tilde{M}_\gamma = (\tilde{\mathbf{C}}_1, \dots, \tilde{\mathbf{C}}_\gamma, \hat{\mathbf{C}}_{\gamma+1} \hat{\mathbf{C}}_T), \tilde{x}_\gamma)$.

We overload notation $\text{hyb}_{ns}^\gamma(1^\lambda, M, x)$ as the output distribution of the hybrid simulator HSim_{ns}^γ . By construction, in HSim_{ns}^γ , when $\gamma = 0$, $\mathbf{M}^0 = \mathbf{P}$ and the garbled input $\tilde{x}_0$ is generated honestly; thus, $\{\text{hyb}_{ns}^0(1^\lambda, M, x)\} = \{\text{real}_{ns}(1^\lambda, M, x)\}$ (where real_{ns} is the distribution of honestly generated garbling; see equation (1)); furthermore, when $\gamma = T$, $\mathbf{M}^0 = \mathbf{Q}$ and the garbled input $\tilde{x}_\gamma$ is simulated; thus $\{\text{hyb}_{ns}^\gamma(1^\lambda, M, x)\} = \{\text{simu}_{ns}(1^\lambda, M, x)\}$ (where simu_{ns} is the distribution of simulated garbling; see equation (2)). Thus to show the indistinguishability between $\{\text{real}_{ns}(1^\lambda, M, x)\}$ and $\{\text{simu}_{ns}(1^\lambda, M, x)\}$, it suffices to show the following claim:

Claim 1. *For every $\gamma \in \mathbb{N}$, the following holds*

$$\left\{ \text{hyb}_{ns}^{\gamma-1}(1^\lambda, M, x) \right\}_\lambda \approx \left\{ \text{hyb}_{ns}^\gamma(1^\lambda, M, x) \right\}_\lambda$$

Proof. Fix a $\gamma \in \mathbb{N}$, a sufficiently large $\lambda \in \mathbb{N}$, an $M = M_\lambda$ and a $x = x_\lambda$. The only difference between the garbling $(\tilde{M}_{\gamma-1}, \tilde{x}_{\gamma-1})$ sampled by $\text{hyb}_{ns}^{\gamma-1}(1^\lambda, M, x)$ and the garbling $(\tilde{M}_\gamma, \tilde{x}_\gamma)$ sampled by $\text{hyb}_{ns}^\gamma(1^\lambda, M, x)$ is the following: Let conf_γ be the intermediate configuration at the beginning of step γ .

- In $\text{hyb}_{ns}^{\gamma-1}$, the γ^{th} garbled circuit $\hat{\mathbf{C}}_\gamma$ is generated honestly using program $\mathbf{P}$. The circuit $\mathbf{C}_\gamma$ (as described in algorithm Garb_{ns}) is the composition of the circuit $\text{Next}^{\lambda, S}(M, \cdot)$ and the encoding algorithm $\text{Encode}_{\text{CIR}}(\text{key}_{\gamma+1}, \cdot)$, where $\text{key}_{\gamma+1} = \text{Gen}_{\text{CIR}}(1^\lambda, 1^S; \alpha_{\gamma+1})$ is generated honestly.

Furthermore, the first $\gamma - 1$ garbled circuits are simulated using $\mathbf{R}$ and $\mathbf{Q}$. The simulation of the first $\gamma - 1$ circuits as well as the generation of the garbled input $\tilde{x}_\gamma$ depends potentially on the garbled input $\widehat{\text{conf}}_\gamma$ corresponding to conf_γ for $\hat{\mathbf{C}}_\gamma$ (when conf_γ is not a final configuration; see Step 1 in $\mathbf{R}$).

In other words, the output of $\text{hyb}_{ns}^{\gamma-1}$ can be generated by the following alternative sampling algorithm:

- Generate garbled circuits $\gamma + 1, \dots, T$ honestly using program $\mathbf{P}$; prepare the γ^{th} circuit $\mathbf{C}_\gamma$ using $\text{key}_{\gamma+1}$.
- Receive externally honest garbling $(\hat{\mathbf{C}}_\gamma, \widehat{\text{conf}}_\gamma)$ of $(\mathbf{C}_\gamma, \text{conf}_\gamma)$.
- Simulate the first $\gamma - 1$ circuits using $\mathbf{R}$ and $\mathbf{Q}$, with $\widehat{\text{conf}}_\gamma$ hardwired in $\mathbf{R}$.
- In hyb_{ns}^γ , the γ^{th} garbled circuit $\tilde{\mathbf{C}}_\gamma$ is simulated using program $\mathbf{R}$; the output out_γ used for simulation is set to either y (if $\text{conf}_{\gamma+1}$ is a final configuration) or the honestly generated garbled input $\widehat{\text{conf}}_{\gamma+1}$. In other words, $\text{out}_\gamma = \mathbf{C}_\gamma(\text{conf}_\gamma)$, where $\mathbf{C}_\gamma$ is prepared in the same way as above.

Furthermore, the previous $\gamma - 1$ garbled circuits are also simulated using program $\mathbf{Q}$. Their simulation as well as the generation of the garbled input $\tilde{x}_{\gamma+1}$ depends potentially on the corresponding simulated garbled input $\widetilde{\text{conf}}_\gamma$ of $\tilde{\mathbf{C}}_\gamma$.

In other words, the output of hyb_{ns}^γ can be generated by the same alternative sampling algorithm above, except that the second step is modified to:

Approved for Public Release; Distribution Unlimited.

- Receive externally simulated garbling $(\tilde{\mathbf{C}}_\gamma, \widetilde{\text{conf}}_\gamma)$ generated using output $\mathbf{C}_\gamma(\text{conf}_\gamma)$.

Then it follows from the security of the circuit garbling scheme $\mathcal{GS}_{\text{CIR}}$ that the distributions of $(\tilde{\mathbf{C}}_\gamma, \widetilde{\text{conf}}_\gamma)$ and $(\tilde{\mathbf{C}}_\gamma, \widetilde{\text{conf}}_\gamma)$ received externally by the alternative sampling algorithm above are computationally indistinguishable, and thus the distributions of outputs of $\text{hyb}_{ns}^{\gamma-1}$ and hyb_{ns}^γ , which can be efficiently constructed from them, are also indistinguishable $\square$

Finally, by the above claim, it follows from a hybrid argument over γ , that $\{\text{real}_{ns}(1^\lambda, M, x)\}$ and $\{\text{simu}_{ns}(1^\lambda, M, x)\}$ are indistinguishable; Hence, $\mathcal{GS}_{ns}$ is a secure garbling scheme for TM.

3.2 A Garbling Scheme for TM with Space-dependent Complexity

In this section, we construct a garbling scheme $\mathcal{GS} = (\text{Garb}, \text{Encode}, \text{Eval})$ for TM with space-dependent complexity. This scheme will rely on the non-succinct garbling scheme $\mathcal{GS}_{ns} = (\text{Garb}_{ns}, \text{Encode}_{ns}, \text{Eval}_{ns})$ in a non-black-box, but largely modular, way.

Overview. The garbling scheme $\mathcal{GS}_{ns}$ described in the previous section is non-succinct because its garbling algorithm Garb_{ns} runs in time proportional to the time-bound T (and generates a garbling of size proportional to T .) Our first observation is that the “bulk” of the computation of Garb_{ns} is evaluating the *same randomized* program $\mathbf{P}(\cdot)$ for T times *with coordinated random coins*, to create a chain of garbled circuits:

$$\hat{M} = (\hat{\mathbf{C}}_1, \dots, \hat{\mathbf{C}}_T), \quad \hat{\mathbf{C}}_t = \mathbf{P}(t; \alpha_t, \alpha_{t+1}, \beta_t)$$

The complexity of each garbled circuit depends only on the size of M and its space complexity S , that is, $\text{poly}(D, S)$ (independent of T). Our main idea towards constructing a garbling scheme $\mathcal{GS}$ with space-dependent complexity is to *defer* the T executions of $\mathbf{P}$, from garbling time (that is, in Garb), to evaluation time (that is, in Eval), by using an indistinguishability obfuscator $i\mathcal{O}$ for circuits. More specifically, instead of computing the chain of garbled circuits $\hat{M}$ directly, the new garbling algorithm Garb generates an obfuscation of the program $\mathbf{P}$, that is $\bar{\mathbf{P}} = i\mathcal{O}(\mathbf{P})$, and use that as the new garbled machine; (since $\mathbf{P}$ has size $\text{poly}(D, S)$, the obfuscation is “succinct” and so is the new garbling algorithm). The procedure for creating garbled inputs $\hat{x}$ remains the same as in the non-succinct scheme $\mathcal{GS}_{ns}$. Then, on input $(\bar{\mathbf{P}}, \hat{x})$, the new evaluation algorithm Eval first generates the chain of garbled circuits $\hat{M} = (\hat{\mathbf{C}}_1, \dots, \hat{\mathbf{C}}_T)$ by evaluating $\bar{\mathbf{P}}$ on inputs from $1, \dots, T$; once the chain $\hat{M}$ of garbled circuits is generated, the output can be computed by evaluating $\text{Eval}_{ns}(\hat{M}, \hat{x})$ as in the non-succinct scheme $\mathcal{GS}_{ns}$. (Note that to make sure that evaluation algorithm has instance-based efficiency, the algorithm Eval actually generates and evaluates $\hat{\mathbf{C}}_t$ ’s one by one, and terminates as soon as an output is produced.)

To make the above high-level idea go through, a few details need to be taken care of. First, the program $\mathbf{P}$ is randomized, whereas indistinguishability obfuscators only handles deterministic circuits. This issue is resolved by obfuscating, instead, a wrapper program $\mathbb{P}(t)$ that runs $\mathbf{P}(t)$ with pseudo-random coins generated using a PRF on input t . In fact, the use of pseudo-random coins also allows coordinating the random coins used in different invocations of $\mathbf{P}$ on different inputs, so that they will produce coherent garbled circuits that can be run together. The second question is how to simulate the new garbled machine $\bar{\mathbb{P}} \xleftarrow{\$} i\mathcal{O}(\mathbb{P})$. In the non-succinct scheme the chain $\hat{M}$ of garbled circuits is simulated by running the program $\mathbf{Q}$ for T times (again with coordinated random coins),

$$\tilde{M} = (\tilde{\mathbf{C}}_1, \dots, \tilde{\mathbf{C}}_T) \quad \hat{\mathbf{C}}_t = \mathbf{Q}(t; \alpha_t, \alpha_{t+1}, \beta_t)$$

Naturally, in the succinct scheme, the simulation creates $\overline{\mathbb{Q}} \stackrel{\$}{\leftarrow} i\mathcal{O}(\mathbb{Q})$ (where $\mathbb{Q}$ is the de-randomized version for $\mathbf{Q}$, as $\mathbb{P}$ is for $\mathbf{P}$). By the pseudo-randomness of PRF and the security of garbled circuits, we have that the truth tables $\tilde{M}$ and $\tilde{M}$ of $\mathbb{P}$ and $\mathbb{Q}$ are indistinguishable; but this does not directly imply that their obfuscations are indistinguishable. We bridge the gap by considering the obfuscation of a sequence of hybrid programs (as in the security proof of the non-succinct garbling scheme).

$$\forall \gamma \in [0, T+1], \quad \mathbf{M}^\gamma = \text{COMBINE}[(\mathbf{Q}, [\gamma-1]), (\mathbf{R}, \{\gamma\}), (\mathbf{P}, [\gamma+1, T])], \quad \overline{\mathbf{M}}^\gamma \stackrel{\$}{\leftarrow} i\mathcal{O}(\mathbf{M}^\gamma)$$

The sequence of hybrid programs “morphs” gradually from program $\mathbf{P} = \mathbf{M}^0$ to program $\mathbf{Q} = \mathbf{M}^{T+1}$; since every pair of subsequent programs $\mathbf{M}^{\gamma-1}, \mathbf{M}^\gamma$ differs only at two inputs ($\gamma-1$ and γ) with indistinguishable outputs, we can use standard techniques such as puncturing and programing to show that their obfuscations are indistinguishable, and hence so are $\overline{\mathbb{P}}$ and $\overline{\mathbb{Q}}$.

Our Succinct Garbling Scheme. We now describe the formal construction, which relies on the following building blocks.

- A garbling scheme for polynomial-sized circuits, with independent key generation: $\mathcal{GS}_{\text{CIR}} = (\text{Garb}_{\text{CIR}}, \text{Encode}_{\text{CIR}}, \text{Eval}_{\text{CIR}})$, where $\text{Garb}_{\text{CIR}} = (\text{Gen}_{\text{CIR}}, \text{Gb}_{\text{CIR}})$ and its the simulation algorithm is $\text{Sim}_{\text{CIR}} = (\text{Sim}\cdot\text{Gen}_{\text{CIR}}, \text{Sim}\cdot\text{Gb}_{\text{CIR}})$.
- An indistinguishability obfuscator $i\mathcal{O}_{\text{CIR}}(\cdot, \cdot)$ for polynomial-sized circuits.
- A puncturable PRF $(\text{PRF}\cdot\text{Gen}, \text{PRF}\cdot\text{Punc}, \text{F})$ with input length $n(\lambda)$ and output length $m(\lambda)$, where $n(\lambda)$ can be set to any super-logarithmic function $n(\lambda) = \omega(\log \lambda)$, and m is a sufficiently large polynomial in λ .

For every λ and $M \in \text{TM}_\lambda$, the garbling scheme $\mathcal{GS}$ proceeds as follows:

Circuit $\mathbb{P} = \mathbb{P}^{\lambda, S, M, K_\alpha, K_\beta}$: On input $t \in [T]$, does:

Generates pseudo-random strings $\alpha_t = \text{F}(K_\alpha, t)$, $\alpha_{t+1} = \text{F}(K_\alpha, t+1)$ and $\beta_t = \text{F}(K_\beta, t)$;
Compute $\hat{\mathbf{C}}_t = \mathbf{P}^{\lambda, S, M}(t; (\alpha_t, \alpha_{t+1}, \beta_t))$ and output $\hat{\mathbf{C}}_t$.

Circuit $\mathbb{Q} = \mathbb{Q}^{\lambda, S, |M|, T^*, y, K_\alpha, K_\beta}$: On input $t \in [T]$, does:

Generate pseudo-random strings $\alpha_t = \text{F}(K_\alpha, t)$, $\alpha_{t+1} = \text{F}(K_\alpha, t+1)$ and $\beta_t = \text{F}(K_\beta, t)$;
Compute $\tilde{\mathbf{C}}_t = \mathbf{Q}^{\lambda, S, |M|, T^*, y}(t; (\alpha_t, \alpha_{t+1}, \beta_t))$ and output $\tilde{\mathbf{C}}_t$.

The circuits in Figure 1, 2 and 3 are padded to their maximum size.

Figure 1: Circuits used in the construction and simulation of $\mathcal{GS}$

The garbling algorithm $\text{Garb}(1^\lambda, M)$:

1. *Sample PRF keys:* $K_\alpha \stackrel{\$}{\leftarrow} \text{PRF}\cdot\text{Gen}(1^\lambda)$ and $K_\beta \stackrel{\$}{\leftarrow} \text{PRF}\cdot\text{Gen}(1^\lambda)$.
2. *Obfuscate the circuit $\mathbb{P}$:*
Obfuscate the circuit $\mathbb{P}(t) = \mathbb{P}^{\lambda, S, M, K_\alpha, K_\beta}(t)$ as described in Figure 1, which is essentially a wrapper program that evaluates $\mathbf{P}$ on t using pseudo-random coins generated using K_α and K_β as described above. Obtain $\overline{\mathbb{P}} \stackrel{\$}{\leftarrow} i\mathcal{O}(1^\lambda, \mathbb{P})$.

Approved for Public Release; Distribution Unlimited.

3. *Generate the key for garbling input:*

Compute **key** in the same way as the garbling scheme Garb_{ns} does, but using pseudo-random coins generated using K_α . That is, Compute the key for the first garbled circuit using randomness $\alpha_1 = F(K_\alpha, 1)$, $\text{key}_1 = \text{Gen}_{\text{CIR}}(1^\lambda, 1^S; \alpha_1)$; set $\mathbf{key} = \text{key}_1 \parallel 1^S$.

4. *Finally, output $(\bar{\mathbb{P}}, \mathbf{key})$.*

The encoding algorithm $\text{Encode}(\mathbf{key}, x)$: Compute $\hat{x} = \text{Encode}_{ns}(\mathbf{key}, x)$.

The evaluation algorithm $\text{Eval}(\bar{\mathbb{P}}, \hat{x})$: Generate and evaluate the garbled circuits in the non-succinct garbling $\hat{M}$ one by one; terminate as soon as an output is produced. More precisely, evaluation proceeds in T iterations as follows:

At the beginning of iteration $t \in [T]$, previous $t - 1$ garbled circuits has been generated and evaluated, producing garbled input $\widehat{\text{conf}}_t$ ($\widehat{\text{conf}}_1 = \hat{x}$). Then, compute $\hat{\mathbf{C}}_t = \bar{\mathbb{P}}(t)$; evaluate $z = \text{Eval}_{\text{CIR}}(\hat{\mathbf{C}}_t, \widehat{\text{conf}}_t)$; if z is a valid output, terminate and output $y = z$; otherwise, proceed to the next iteration $t + 1$ with $\widehat{\text{conf}}_{t+1} = z$.

Next, we proceed to show that $\mathcal{GS}$ is a garbling scheme for TM with space-dependent complexity.

Correctness. Fix any machine $M \in \text{TM}$ and input x . Recall that the garbling algorithm Garb generates a pair $(\bar{\mathbb{P}}, \mathbf{key})$; the latter is later used by the encoding algorithm Encode to obtain garbled input $\hat{x}$, while the former is later used by the evaluation algorithm Eval to create the non-succinct garbling $\hat{M} = \{\hat{\mathbf{C}}_t = \bar{\mathbb{P}}(t)\}_{t \in [T]}$; the non-succinct garbling $\hat{M}$ is then evaluated with $\hat{x}$ using algorithm Eval_{ns} . The distribution of the garbled input and the non-succinct garbling recovered by Eval is as follows:

$$\mathcal{D}_1 = \left\{ (\bar{\mathbb{P}}, \mathbf{key}) \xleftarrow{\$} \text{Garb}(1^\lambda, M) : \left(\hat{x} = \text{Encode}(\mathbf{key}, x), \quad \hat{M} = \left\{ \hat{\mathbf{C}}_t = \bar{\mathbb{P}}(t) \right\}_{t \in [T]} \right) \right\}$$

It follows from the construction of Garb , Encode and the correctness of the indistinguishability obfuscator that the above distribution $\mathcal{D}_1$ is identical to the distribution $\mathcal{D}_2$ of a garbled pair $(\hat{M}', \hat{x}')$ generated by the algorithms Garb_{ns} , Encode_{ns} of the non-succinct scheme, *using pseudo-random coins*, formalized below.

$$\mathcal{D}_2 = \left\{ K_\alpha, K_\beta \xleftarrow{\$} \text{PRF} \cdot \text{Gen}(1^\lambda), \quad \forall t \in [T], \alpha_t = F(K_\alpha, t), \beta_t = F(K_\beta, t) : \right. \\ \left. \left(\hat{x}' = \text{Encode}_{ns}(\mathbf{key}' = \text{Gen}_{\text{CIR}}(1^\lambda, 1^S; \alpha_1), x), \quad \hat{M}' = \left\{ \hat{\mathbf{C}}_t = \mathbf{P}(t; \alpha_t, \alpha_{t+1}, \beta_t) \right\}_{t \in [T]} \right) \right\}$$

By the pseudo-randomness of PRF, distribution $\mathcal{D}_2$ is computationally indistinguishable from the garbled pair generated by Garb_{ns} , Encode_{ns} , using truly random coins.

$$\mathcal{D}_3 = \left\{ (\hat{M}'', \mathbf{key}'') \xleftarrow{\$} \text{Garb}_{ns}(1^\lambda, M) : \left(\hat{x}'' = \text{Encode}_{ns}(\mathbf{key}'', x), \quad \hat{M}'' \right) \right\}$$

The correctness of the non-succinct garbling scheme $\mathcal{GS}_{ns}$ guarantees that with overwhelming probability, evaluating $\hat{M}''$ with $\hat{x}''$ produces the correct output $y = M(x)$; furthermore, the correct output y is produced after evaluating only the first $T^* = T_M(x)$ garbled circuits. Thus, it follows from the indistinguishability between $\mathcal{D}_1$ and $\mathcal{D}_3$ that, when evaluating a garbled pair $(\hat{M}, \hat{x})$ sampled from $\mathcal{D}_1$, the correct output y is also produced after evaluating the first T^* garbled circuits. Given that $\mathcal{D}_1$ is exactly the distribution of the non-succinct garbled pairs generated in Eval , we have that correctness holds.

Efficiency. We show that the garbling scheme $\mathcal{GS}$ has space-dependent complexity.

- The garbling algorithm $\text{Garb}(1^\lambda, M)$ runs in time $\text{poly}(\lambda, |M|, S)$. This is because Garb produces an obfuscation of the program $\mathbb{P}$ (a de-randomized version of P) which garbles circuits $\mathbf{C}_t$ using pseudo-random coins for every input $t \in [T]$. Since the program $\mathbf{C}_t$ has size $q = \text{poly}(\lambda, |M|, S)$ as analyzed in the non-succinct garbling scheme, so does $\mathbf{P}$ and $\mathbb{P}$ (note that the input range T of these two programs are contained as part of the description of M , and hence $|M| > \log T$). Therefore, Garb takes time $\text{poly}(\lambda, |M|, S)$ to produce the obfuscation of $\mathbb{P}$. Additionally, notice that Garb generates the **key** as the algorithm Garb_{ns} does, which in turn runs $\text{Garb}_{\text{CIR}}(1^\lambda, 1^S)$ and takes time $\text{poly}(\lambda, S)$. Overall, Garb runs in time $\text{poly}(\lambda, |M|, S)$ as claimed.
- Encode runs in time the same as the Encode_{ns} algorithm which is $\text{poly}(\lambda, |M|, S)$.
- The evaluation algorithm Eval on input $(\bar{\mathbb{P}}, \hat{x})$ produced by $(\bar{\mathbb{P}}, \mathbf{key}) \xleftarrow{\$} \text{Garb}(1^\lambda, 1^S)$ and $\hat{x} = \text{Encode}(\mathbf{key}, x)$ runs in time $\text{poly}(\lambda, |M|, S) \times T^*$, $T^* = T_M(x)$, with overwhelming probability.

It follows from the analysis of correctness of $\mathcal{GS}$ that with overwhelming probability over the coins of Garb , the non-succinct garbling $\hat{M}$ defined by $\bar{\mathbb{P}}$ satisfies that when evaluated with $\hat{x}$, the correct output is produced after T^* iterations. Since Eval does not compute the entire non-succinct garbling $\hat{M}$ in one shot, but rather, generates and evaluates the garbled circuits in $\hat{M}$ one by one. Thus it terminates after producing and evaluating T^* garbled circuits. Since the generation and evaluation of each garbled circuit takes $\text{poly}(\lambda, |M|, S)$ time, overall Eval runs in time $T_M(x) \times \text{poly}(\lambda, |M|, S)$ as claimed.

Security. Fix any polynomial T' , any sequence of algorithms $\{M = M_\lambda\} \in \{\text{TM}_\lambda^{T'}\}$, and any sequence of inputs $\{x = x_\lambda\}$ where $x_\lambda \in \{0, 1\}^{M.n}$. Towards showing the security of $\mathcal{GS}$, we construct a simulator Sim , satisfying that the following two ensembles are indistinguishable in λ :

$$\left\{ \text{real}(1^\lambda, M, x) \right\} = \left\{ (\bar{\mathbb{P}}, \mathbf{key}) \xleftarrow{\$} \text{Garb}(1^\lambda, M), \hat{x} = \text{Encode}(\mathbf{key}, x) : (\bar{\mathbb{P}}, \hat{x}) \right\}_\lambda \quad (3)$$

$$\left\{ \text{simu}(1^\lambda, M, x) \right\} = \left\{ (\bar{\mathbb{Q}}, \tilde{x}) \xleftarrow{\$} \text{Sim}(1^\lambda, 1^{|x|}1^{|M|}, S, T, T_M(x), M(x)) : (\bar{\mathbb{Q}}, \tilde{x}) \right\}_\lambda \quad (4)$$

As discussed in the overview, the simulation will obfuscate the program $\mathbf{Q}$ used for simulating the non-succinct garbled machine $\tilde{M} = (\tilde{\mathbf{C}}_1, \dots, \tilde{\mathbf{C}}_T)$. More precisely,

The simulation algorithm $\text{Sim}(1^\lambda, 1^{|x|}, 1^{|M|}, S, T, T^* = T_M(x), y = M(x))$:

1. *Sample PRF keys:* $K_\alpha \xleftarrow{\$} \text{PRF} \cdot \text{Gen}(1^\lambda)$ and $K_\beta \xleftarrow{\$} \text{PRF} \cdot \text{Gen}(1^\lambda)$.
2. *Obfuscate the circuit $\mathbb{Q}$:*
Obfuscate the circuit $\mathbb{Q}(t) = \mathbb{Q}^{\lambda, S, |M|, T^*, y, K_\alpha, K_\beta}(t)$ as described in Figure 1, which is essentially a wrapper program that evaluates $\mathbf{Q}$ on t , using pseudo-random coins $\{\alpha_t, \beta_t\}$ generated by evaluating $\mathbf{F}$ on keys K_α and K_β and inputs $t \in [T]$. Obtain $\bar{\mathbb{Q}} \xleftarrow{\$} i\mathcal{O}(1^\lambda, \mathbb{Q})$.
3. *Simulate the garbled input:*
Simulate the garbled input $\tilde{x}$ in the same way as simulator Sim_{ns} does, but using pseudo-random coins. That is, compute $(\text{conf}_1, \mathbf{st}_1) = \text{Sim} \cdot \text{Gen}_{\text{CIR}}(1^\lambda, 1^S; \alpha_1)$, where $\alpha_1 = \mathbf{F}(K_\alpha, 1)$; set $\tilde{x} = \text{conf}_1$.

4. Finally, output $(\overline{\mathbb{Q}}, \tilde{x})$.

The simulator $\text{Sim}(1^\lambda, 1^{|x|}, 1^{|M|}, S, T, T^*, y = M(x))$ runs in time $\text{poly}(\lambda, |M|, S)$. This follows because the simulator simulates the garbled Turing machine by obfuscating the program $\mathbb{Q}$. As the program $\mathbb{Q}$ simply runs $\mathbf{Q}$ using pseudo-random coins, its size is $\text{poly}(\lambda, |M|, S)$; thus obfuscation takes time in the same order. On the other hand, Sim simulates the garbled input $\tilde{x}$ as the simulator Sim_{ns} does, which simply invokes $\text{Sim}_{\text{CIR}}(1^\lambda, 1^S)$ of the circuit garbling scheme, which takes time $\text{poly}(\lambda, S)$. Therefore, overall the simulation takes time $\text{poly}(\lambda, |M|, S)$ as claimed.

Towards showing the indistinguishability between honestly generated garbling $(\overline{\mathbb{P}}, \hat{x}) \stackrel{\$}{\leftarrow} \text{real}(1^\lambda, M, x)$ and the simulation $(\overline{\mathbb{Q}}, \tilde{x}) \stackrel{\$}{\leftarrow} \text{simu}(1^\lambda, M, x)$ (see equation (3) and (4) for formal definition of **real** and **simu**), we will consider a sequence of hybrids $\text{hyb}^0, \dots, \text{hyb}^T$, where the output distribution of hyb^0 is identical to **real**, while that of hyb^T is identical to **simu**. In every intermediate hybrid hyb^γ , a hybrid simulator HSim^γ is invoked, producing a pair $(\overline{\mathbb{M}}^\gamma, \tilde{x}^\gamma)$, where $\overline{\mathbb{M}}^\gamma$ is the obfuscation of (the de-randomized wrapper of) a merged program $\mathbf{M}^\gamma$ that produces a hybrid chain of garbled circuit as in the security proof of the non-succinct garbling scheme, where the first γ garbled circuits are simulated and the rest are generated honestly. More precisely,

The hybrid simulation algorithm $\text{HSim}^\gamma(1^\lambda, M, x)$ for $\gamma = 0, \dots, T$:

Compute $T^* = T_M(x)$ and $y = M(x)$, and the intermediate configuration $\text{conf}_{\gamma+1}$ as defined by $\text{CONFIG}(M, x)$.

1. Sample PRF keys: $K_\alpha \stackrel{\$}{\leftarrow} \text{PRF}\cdot\text{Gen}(1^\lambda)$ and $K_\beta \stackrel{\$}{\leftarrow} \text{PRF}\cdot\text{Gen}(1^\lambda)$.

2. Obfuscate the circuit $\mathbb{M}^\gamma$:

Obfuscate the circuit $\mathbb{M}^\gamma(t) = (\mathbb{M}^\gamma)^{\lambda, S, M, T^*, y, \text{conf}_{\gamma+1}, K_\alpha, K_\beta}(t)$ as described in Figure 1, which is essentially a wrapper program that evaluates the combined program

$$\mathbf{M}^\gamma = \text{COMBINE}[(\mathbf{Q}, [\gamma - 1]), (\mathbf{R}, \{\gamma\}), (\mathbf{P}, [\gamma + 1, T])](t; (\alpha_t, \alpha_{t+1}, \beta_t)),$$

using pseudo-random coins $\{\alpha_t, \beta_t\}$ generated using K_α and K_β . Obtain $\overline{\mathbb{M}}^\gamma \stackrel{\$}{\leftarrow} i\mathcal{O}(1^\lambda, \mathbb{M}^\gamma)$.

3. Simulate the garbled input:

If $\gamma > 0$, simulate the garbled input $\tilde{x}^\gamma$ in the same way as in **Sim**. Otherwise, if $\gamma = 0$, generate $\tilde{x}^0$ honestly, using **Garb** and **Encode**.

4. Finally, output $(\overline{\mathbb{M}}^\gamma, \tilde{x}^\gamma)$.

Circuit $\mathbb{M}^\gamma = (\mathbb{M}^\gamma)^{\lambda, S, M, T^*, y, \text{conf}_{\gamma+1}, K_\alpha, K_\beta}$: On input $t \in [T]$, does:

Generate pseudo-random strings $\alpha_t = F(K_\alpha, t)$, $\alpha_{t+1} = F(K_\alpha, t + 1)$ and $\beta_t = F(K_\beta, t)$;

Compute $\tilde{\mathbf{C}}_t = \mathbf{M}^\gamma(t; (\alpha_t, \alpha_{t+1}, \beta_t))$ and output $\tilde{\mathbf{C}}_t$, where $\mathbf{M}^\gamma$ is:

$$(\mathbf{M}^\gamma)^{\lambda, S, M, T^*, y, \text{conf}_{\gamma+1}} = \text{COMBINE}[(\mathbf{Q}, [\gamma - 1]), (\mathbf{R}, \{\gamma\}), (\mathbf{P}, [\gamma + 1, T])](t; (\alpha_t, \alpha_{t+1}, \beta_t))$$

The circuits in Figure 1, 2 and 3 are padded to their maximum size.

Figure 2: Circuits used in the security analysis of $\mathcal{GS}$

We describe circuits $\mathbb{M}_1^\gamma$ to $\mathbb{M}_6^\gamma$. They all have parameters $\lambda, S, M, T^*, y, \text{conf}_{\gamma+1}$ hardwired in; for simplicity, we suppress these parameters in the superscript.

Circuit $\mathbb{M}_1^\gamma = (\mathbb{M}_1^\gamma)^{K_\alpha(\gamma+1), K_\beta(\gamma+1), \alpha_{\gamma+1}, \beta_{\gamma+1}}$: On input $t \in [T]$, does:

If $t \neq \gamma$, generate pseudo-random string $\alpha_{t+1} = F(K_\alpha(\gamma+1), t+1)$.

If $t \neq \gamma+1$, generate pseudo-random strings $\alpha_{t+1} = F(K_\alpha(\gamma+1), t)$ and $\beta_t = F(K_\beta(\gamma+1), t)$.

Proceed as $\mathbb{M}^\gamma$ does using random coins $\alpha_t, \alpha_{t+1}, \beta_t$.

Circuit $\mathbb{M}_2^\gamma = (\mathbb{M}_2^\gamma)^{K_\alpha(\gamma+1), K_\beta(\gamma+1), \alpha'_{\gamma+1}, \beta'_{\gamma+1}}$:

Identical to $(\mathbb{M}_1^\gamma)^{K_\alpha(\gamma+1), K_\beta(\gamma+1), \alpha'_{\gamma+1}, \beta'_{\gamma+1}}$, with $\alpha'_{\gamma+1}, \beta'_{\gamma+1}$ sampled at random.

Circuit $\mathbb{M}_3^\gamma = (\mathbb{M}_3^\gamma)^{K_\alpha(\gamma+1), K_\beta(\gamma+1), \widehat{\mathbf{C}}_{\gamma+1}, \widehat{\text{conf}}_{\gamma+1}}$: On input $t \in [T]$, does:

If $t = \gamma+1$, output $\widehat{\mathbf{C}}_{\gamma+1}$.

If $t = \gamma$, set out_γ using $\widehat{\text{conf}}_{\gamma+1}$ as in Step 1 of program **R**; simulate and output $\widetilde{\mathbf{C}}_\gamma$ as in Step 2 of **R**.

Otherwise, compute as $\mathbb{M}_2^\gamma$ does using the punctured keys $K_\alpha(\gamma+1), K_\beta(\gamma+1)$.

Circuit $\mathbb{M}_4^\gamma = (\mathbb{M}_4^\gamma)^{K_\alpha(\gamma+1), K_\beta(\gamma+1), \widetilde{\mathbf{C}}_{\gamma+1}, \widetilde{\text{conf}}_{\gamma+1}}$:

Identical to $(\mathbb{M}_3^\gamma)^{K_\alpha(\gamma+1), K_\beta(\gamma+1), \widetilde{\mathbf{C}}_{\gamma+1}, \widetilde{\text{conf}}_{\gamma+1}}$, with simulated garbling pair $\widetilde{\mathbf{C}}_{\gamma+1}, \widetilde{\text{conf}}_{\gamma+1}$.

Circuit $\mathbb{M}_5^\gamma = (\mathbb{M}_5^\gamma)^{K_\alpha(\gamma+1), K_\beta(\gamma+1), \alpha'_{\gamma+1}, \beta'_{\gamma+1}}$: On input $t \in [T]$, does:

If $t = \gamma+1$, compute $\widetilde{\mathbf{C}}_{\gamma+1}$ using program **R** with randomness $\alpha'_{\gamma+1}, \alpha_{\gamma+2}, \beta'_{\gamma+1}$.

If $t = \gamma$, compute $\widetilde{\mathbf{C}}_\gamma$ using program **Q**, which internally computes $\widetilde{\text{conf}}_{\gamma+1}$ for setting the output out_γ using randomness $\alpha'_{\gamma+1}$.

Otherwise, compute as $\mathbb{M}_4^\gamma$ does using the punctured keys $K_\alpha(\gamma+1), K_\beta(\gamma+1)$.

Circuit $\mathbb{M}_6^\gamma = (\mathbb{M}_6^\gamma)^{K_\alpha(\gamma+1), K_\beta(\gamma+1), \alpha_{\gamma+1}, \beta_{\gamma+1}}$:

Identical to $(\mathbb{M}_5^\gamma)^{K_\alpha(\gamma+1), K_\beta(\gamma+1), \alpha_{\gamma+1}, \beta_{\gamma+1}}$, with $\alpha_{\gamma+1} = F(K_\alpha, \gamma+1)$, $\beta_{\gamma+1} = F(K_\beta, \gamma+1)$

The circuits in Figure 1, 2 and 3 are padded to their maximum size.

Figure 3: Circuits used in the security analysis of $\mathcal{GS}$, continued

We overload the notation $\text{hyb}^\gamma(1^\lambda, M, x)$ as the output distribution of the γ^{th} hybrid. By construction, when $\gamma = 0$, $\mathbf{M}^0 = \mathbf{P}$ and the garbled input $\tilde{x}^0$ is generated honestly; thus, $\{\text{hyb}^0(1^\lambda, M, x)\} = \{\text{real}(1^\lambda, M, x)\}$; furthermore, when $\gamma = T$, $\mathbf{M}^T = \mathbf{Q}$ and the garbled input $\tilde{x}^T$ is simulated; thus $\{\text{hyb}^T(1^\lambda, M, x)\} = \{\text{simu}(1^\lambda, M, x)\}$. Therefore, to show the security of $\mathcal{GS}$, it boils down to proving the following claim:

Claim 2. *For every $\gamma \geq 0$, the following holds*

$$\left\{ \text{hyb}^\gamma(1^\lambda, M, x) \right\}_\lambda \approx \left\{ \text{hyb}^{\gamma+1}(1^\lambda, M, x) \right\}_\lambda$$

Proof. Fix a $\gamma \in \mathbb{N}$, a sufficiently large $\lambda \in \mathbb{N}$, an $M = M_\lambda$ and a $x = x_\lambda$. Note that the only difference between $(\overline{\mathbf{M}}^\gamma, \tilde{x}^\gamma) \stackrel{\$}{\leftarrow} \text{hyb}^\gamma$ and $(\overline{\mathbf{M}}^{\gamma+1}, \tilde{x}^{\gamma+1}) \stackrel{\$}{\leftarrow} \text{hyb}^{\gamma+1}$ is the following:

- For every γ , the underlying obfuscated programs $\mathbb{M}^\gamma, \mathbb{M}^{\gamma+1}$ differ on their implementation for at most two inputs, namely $\gamma, \gamma+1$, and,

Approved for Public Release; Distribution Unlimited.

- when $\gamma = 0$, the garbled input $\tilde{x}^0$ is generated honestly in hyb^0 , whereas $\tilde{x}^1$ is simulated in hyb^1 .

To show the indistinguishability of the two hybrids, we consider a sequence of sub-hybrids from $H_0^\gamma = \text{hyb}^\gamma$ to $H_7^\gamma = \text{hyb}^{\gamma+1}$. Below we describe these hybrids $H_0^\gamma, \dots, H_7^\gamma$, and argue that the output distributions of any two subsequent hybrids are indistinguishable. We denote by $(\overline{M}_i^\gamma, \tilde{x}_i^\gamma)$ the garbled pair produced in hybrid H_i^γ for $i = 0, \dots, 7$. For convenience, below we suppress the superscript γ , and simply use notations $H_i = H_i^\gamma$, $\overline{M}_i = \overline{M}_i^\gamma$, $M_i = M_i^\gamma$ and $\tilde{x}_i = \tilde{x}_i^\gamma$.

Hybrid H_1 : Generate a garbled pair $(\overline{M}_1, \tilde{x}_1)$ by running a simulation procedure that proceeds identically to HSim^γ , except from the following modifications:

- In the first step, puncture the two PRF keys K_α, K_β at input $\gamma + 1$, and obtain $K_\alpha(\gamma + 1) = \text{PRF} \cdot \text{Punc}(K_\alpha, \gamma + 1)$ and $K_\beta(\gamma + 1) = \text{PRF} \cdot \text{Punc}(K_\beta, \gamma + 1)$. Furthermore, compute $\alpha_{\gamma+1} = F(K_\alpha, \gamma + 1)$ and $\beta_{\gamma+1} = F(K_\beta, \gamma + 1)$.
- In the second step, obfuscate a circuit M_1 slightly modified from M^γ : Instead of having the full PRF keys K_α, K_β hardwired in, M_1 has the punctured keys $K_\alpha(\gamma + 1), K_\beta(\gamma + 1)$ and the PRF values $\alpha_{\gamma+1}, \beta_{\gamma+1}$ hardwired in; M_1 proceeds identically to M_1 , except that it uses the punctured PRF keys to generate pseudo-random coins corresponding to input $t \neq \gamma + 1$ and directly use $\alpha_{\gamma+1}, \beta_{\gamma+1}$ as the coins for input $t = \gamma + 1$. See Figure 1 for a description of $M_1 = M_1^\gamma$.

By construction, H_1 only differs from hyb^γ at which underlying program is obfuscated, and program M_1 has the same functionality as M^γ . Thus it follows from the security of indistinguishability obfuscator $i\mathcal{O}$ that, the obfuscated programs $\overline{M}^\gamma$ and $\overline{M}_1$ are indistinguishable. (Furthermore, the garbled inputs $\tilde{x}^\gamma$ and $\tilde{x}_1$ in these two hybrids are generated in the same way.) Thus, we have that the output $(\overline{M}_1, \tilde{x}_1)$ of H_1 is indistinguishable from the output $(\overline{M}^\gamma, \tilde{x}^\gamma)$ of hyb^γ . That is,

$$\left\{ \text{hyb}^\gamma(1^\lambda, M, x) \right\}_\lambda \approx \left\{ H_0(1^\lambda, M, x) \right\}_\lambda$$

Hybrid H_2 : Generate a garbled pair $(\overline{M}_2, \tilde{x}_2)$ by running the same simulation procedure as in H_1 except from the following modifications: Instead of using pseudo-random coins $\alpha_{\gamma+1}$ and $\beta_{\gamma+1}$, hybrid H_2 samples two sufficiently long truly random string $\alpha'_{\gamma+1}, \beta'_{\gamma+1} \xleftarrow{\$} \{0, 1\}^{\text{poly}(\lambda)}$ and replace $\alpha_{\gamma+1}, \beta_{\gamma+1}$ with these truly random strings. More specifically, H_2 obfuscates a program M_2 that is identical to M_1 , but with $(K_\alpha(\gamma + 1), K_\beta(\gamma + 1), \alpha'_{\gamma+1}, \beta'_{\gamma+1})$ hardwired in; furthermore, if $\gamma = 0$, α'_1 (as opposed to α_1) is used to generate the garbled input $\tilde{x}_2$. Since only the punctured keys $K_\alpha(\gamma + 1), K_\beta(\gamma + 1)$ are used in the whole simulation procedure, it follows from the pseudo-randomness of the punctured PRF that the output $(\overline{M}_2, \tilde{x}_2)$ of H_2 is indistinguishable from that $(\overline{M}_1, \tilde{x}_1)$ of hyb_1 . That is,

$$\left\{ H_1(1^\lambda, M, x) \right\}_\lambda \approx \left\{ H_2(1^\lambda, M, x) \right\}_\lambda$$

Hybrid H_3 : Generate a garbled pair $(\overline{M}_3, \tilde{x}_3)$ by running the same simulation procedure as in H_2 with the following modifications:

- Observe that in program M_2 , $\alpha'_{\gamma+1}, \beta'_{\gamma+1}$ are used in the evaluation of at most two inputs, γ and $\gamma + 1$:

For input $\gamma + 1$, program **P** is invoked with input $\gamma + 1$ and randomness $\alpha'_{\gamma+1}, \alpha_{\gamma+2}, \beta'_{\gamma+1}$, in which a circuit $\mathbf{C}_{\gamma+1}$ is prepared depending on $\alpha_{\gamma+2}$, and then obfuscated by computing

$$\text{key}_{\gamma+1} = \text{Gen}_{\text{CIR}}(1^\lambda, 1^S; \alpha'_{\gamma+1}) \quad \widehat{\mathbf{C}}_{\gamma+1} = \text{Gb}_{\text{CIR}}(\text{key}_{\gamma+1}, \mathbf{C}_{\gamma+1}; \beta'_{\gamma+1})$$

If $\gamma > 0$, for input γ , program **R** is invoked with input γ and randomness $\alpha_\gamma, \alpha'_{\gamma+1}, \beta_\gamma$, in which a garbled circuit $\widetilde{\mathbf{C}}_\gamma$ is simulated; the output out_γ used for the simulation depends potentially on an honest garbling of $\text{conf}_{\gamma+1}$, that is,

$$\widehat{\text{conf}}_{\gamma+1} = \text{Encode}_{\text{CIR}} \left(\text{Gen}_{\text{CIR}}(1^\lambda, 1^S; \alpha'_{\gamma+1}), \text{conf}_{\gamma+1} \right)$$

Using out_γ , $\widetilde{\mathbf{C}}_\gamma$ is simulating using randomness $\alpha_\gamma, \beta_\gamma$.

First modification: Hybrid $\mathbf{H}_3$ receives externally the above pair $(\widehat{\mathbf{C}}_{\gamma+1}, \widehat{\text{conf}}_{\gamma+1})$. Instead of obfuscating $\mathbb{M}_2$ (which computes $\widehat{\mathbf{C}}_{\gamma+1}, \widehat{\text{conf}}_{\gamma+1}$ internally), $\mathbf{H}_3$ obfuscates $\mathbb{M}_3$ that has $\widehat{\mathbf{C}}_{\gamma+1}, \widehat{\text{conf}}_{\gamma+1}$ directly hardwired in (as well as $K_\alpha(\gamma + 1), K_\beta(\gamma + 1)$). $\mathbb{M}_3$ on input $\gamma + 1$, directly outputs $\widehat{\text{conf}}_{\gamma+1}$; on input γ , it uses $\widehat{\text{conf}}_{\gamma+1}$ to compute $\widetilde{\mathbf{C}}_\gamma$; on all other inputs, it proceeds identically as $\mathbb{M}_2$. (See Figure 1 for a description of $\mathbb{M}_3$.) It is easy to see that when the correct values $\widehat{\mathbf{C}}_{\gamma+1}, \widehat{\text{conf}}_{\gamma+1}$ are hardwired, the program $\mathbb{M}_3$ has the same functionality as $\mathbb{M}_2$.

- In $\mathbf{H}_2$, if $\gamma = 0$, α'_1 is used for garbling the input,

$$\text{key}_1 = \text{Gen}_{\text{CIR}}(1^\lambda, 1^S; \alpha'_1) \quad \widehat{\text{conf}}_1 = \text{Encode}_{\text{CIR}}(\text{key}_1, \text{conf}_1)$$

where conf_1 is the initial state corresponding to x .

Second modification: Instead, if $\gamma = 0$, hybrid $\mathbf{H}_3$ receives $\widehat{\text{conf}}_1$ externally, and directly outputs it as the garbled inputs $\hat{x}_3 = \widehat{\text{conf}}_1$.

When $\mathbf{H}_3$ receives the correct values of $(\widehat{\text{conf}}_{\gamma+1}, \widehat{\mathbf{C}}_{\gamma+1})$ externally, it follows from the security of $i\mathcal{O}$ that the output distribution of $\mathbf{H}_3$ is indistinguishable from that of $\mathbf{H}_2$. That is,

$$\left\{ \mathbf{H}_2(1^\lambda, M, x) \right\}_\lambda \approx \left\{ \mathbf{H}_3(1^\lambda, M, x) \right\}_\lambda$$

Hybrid $\mathbf{H}_4$: Generate a garbled pair $(\widetilde{\mathbb{M}}_4, \tilde{x}_4)$ by running the same procedure as in $\mathbf{H}_3$, except that $\mathbf{H}_4$ receives externally a simulated pair $(\widetilde{\text{conf}}_{\gamma+1}, \widetilde{\mathbf{C}}_{\gamma+1})$ produced as follows:

$$(\widetilde{\text{conf}}_{\gamma+1}, \mathbf{st}_{\gamma+1}) = \text{Sim} \cdot \text{Gen}_{\text{CIR}}(1^\lambda, 1^S; \alpha'_{\gamma+1}) \quad (5)$$

$$\widetilde{\mathbf{C}}_{\gamma+1} = \text{Sim} \cdot \text{Gb}_{\text{CIR}} \left(1^\lambda, 1^S, 1^q, out_{\gamma+1}, \mathbf{st}_{\gamma+1}; \beta'_{\gamma+1} \right) \quad (6)$$

where $out_{\gamma+1}$ is set to be the output of circuit $\mathbf{C}_{\gamma+1}$ on input $\text{conf}_{\gamma+1}$. Thus, it follows from the security of the circuit garbling scheme $\mathcal{GS}_{\text{CIR}}$ that the simulated pair $(\widetilde{\text{conf}}_{\gamma+1}, \widetilde{\mathbf{C}}_{\gamma+1})$ that hybrid $\mathbf{H}_4$ receives externally is indistinguishable to the honest pair $(\widehat{\text{conf}}_{\gamma+1}, \widehat{\mathbf{C}}_{\gamma+1})$ that $\mathbf{H}_3$ receives externally. Since these two hybrids only differ in which pair they receive externally, it follows that:

$$\left\{ \mathbf{H}_3(1^\lambda, M, x) \right\}_\lambda \approx \left\{ \mathbf{H}_4(1^\lambda, M, x) \right\}_\lambda$$

Hybrid H₅: Generate a garbled pair $(\widetilde{\mathbb{M}}_5, \tilde{x}_5)$ by running the same procedure as in H₄, except that instead of receiving $(\widetilde{\text{conf}}_{\gamma+1}, \widetilde{\mathbf{C}}_{\gamma+1})$ externally, it computes them internally using truly random coins $\alpha'_{\gamma+1}, \beta'_{\gamma+1}$. More precisely,

- It obfuscate a program $\mathbb{M}_5$ that have $K_\alpha(\gamma+1), K_\beta(\gamma+1), \alpha'_{\gamma+1}, \beta'_{\gamma+1}$ hardwired in:
 On input $\gamma+1$, it computes $\widetilde{\mathbf{C}}_{\gamma+1}$ using the program $\mathbf{R}$ with randomness $\alpha'_{\gamma+1}, \alpha_{\gamma+2}, \beta'_{\gamma+1}$ (which computes $\widetilde{\mathbf{C}}_{\gamma+1}$ as described in equations (5) and (6)).
 On input γ , it computes $\widetilde{\mathbf{C}}_\gamma$ using the program $\mathbf{Q}$ with randomness $\alpha_\gamma, \alpha'_{\gamma+2}, \beta_\gamma$ (which computes internally $\widetilde{\text{conf}}_{\gamma+1}$ as described in equation (5)).
 On other inputs $t \neq \gamma, \gamma+1$, it computes as $\mathbb{M}_4$ does.
- If $\gamma = 0$, α'_1 is used for computing $\widetilde{\text{conf}}_1$ as described in equation (5), and then output $\tilde{x}_4 = \widetilde{\text{conf}}_1$.

It follows from the fact that $\mathbb{M}_5$ computes $(\widetilde{\text{conf}}_{\gamma+1}, \widetilde{\mathbf{C}}_{\gamma+1})$ correctly internally, it has the same functionality as $\mathbb{M}_4$; thus, the obfuscation of these two programs are indistinguishable. Combined with the fact that the distribution of the garbled inputs $\tilde{x}_4$ is identical to $\tilde{x}_3$, we have that

$$\left\{ \mathbf{H}_4(1^\lambda, M, x) \right\}_\lambda \approx \left\{ \mathbf{H}_5(1^\lambda, M, x) \right\}_\lambda$$

Hybrid H₆: Generate a garbled pair $(\widetilde{\mathbb{M}}_6, \tilde{x}_6)$ by running the same procedure as in H₅, except that instead of using truly random coins $\alpha'_{\gamma+1}, \beta'_{\gamma+1}$, use pseudo-random coins $\alpha_{\gamma+1} = \mathbf{F}(K_\alpha, \gamma+1)$ and $\beta_{\gamma+1} = \mathbf{F}(K_\beta, \gamma+1)$. In particular, H₆ obfuscates a program $\mathbb{M}_6$ that is identical to $\mathbb{M}_5$ except that $K_\alpha(\gamma+1), K_\beta(\gamma+1), \alpha_{\gamma+1}, \beta_{\gamma+1}$ are hardwired in, and if $\gamma = 0$, α_1 is used to generate the garbled input $\tilde{x}_6$. It follows from the pseudo-randomness of the punctured PRF that:

$$\left\{ \mathbf{H}_6(1^\lambda, M, x) \right\}_\lambda \approx \left\{ \mathbf{H}_5(1^\lambda, M, x) \right\}_\lambda$$

Hybrid H₇: Generate a garbled pair $(\widetilde{\mathbb{M}}_7, \tilde{x}_7)$ by running the hybrid simulator $\mathbf{HSim}^{\gamma+1}$. Note that the only difference between $\mathbf{HSim}^{\gamma+1}$ and the simulation procedure in H₆ is that instead of obfuscating $\mathbb{M}_6$ that has tuple $(K_\alpha(\gamma+1), K_\beta(\gamma+1), \alpha_{\gamma+1}, \beta_{\gamma+1})$ hardwired in, $\mathbf{HSim}^{\gamma+1}$ obfuscates $\mathbb{M}^{\gamma+1}$ that has the full PRF keys K_α, K_β hardwired in and evaluates $\alpha_{\gamma+1}, \beta_{\gamma+1}$ internally.

Since $\mathbb{M}^{\gamma+1}$ and $\mathbb{M}_6^\gamma$ has the same functionality, it follows from the security of $i\mathcal{O}$ that

$$\left\{ \mathbf{H}_6(1^\lambda, M, x) \right\}_\lambda \approx \left\{ \mathbf{H}_5(1^\lambda, M, x) \right\}_\lambda$$

Finally, by a hybrid argument, we conclude the claim. □

Given the above claim, by a hybrid argument over γ , we have that $\{\mathbf{real}(1^\lambda, M, x)\}$ and $\{\mathbf{simu}(1^\lambda, M, x)\}$ are indistinguishable; Hence, $\mathcal{GS}$ is a secure garbling scheme for TM.

4 Succinct Garbling of Bounded Space Computation

In the section, we observe that our approach for constructing a succinct garbling scheme for bounded space TM in the previous two sections applies generally to any *bounded space computation* (e.g., bounded-space RAM). This immediately yields a garbling scheme for any model of computation with space-dependent complexity.

Theorem 6. *Assuming the existence of IO for circuits and one-way functions. There exists a garbling scheme for any abstract model of sequential computation, such as TM and RAM, with space-dependent complexity.*

A Garbing Scheme for Any Bounded Space Computation: Given an underlying circuit garbling scheme $\mathcal{GS} = (\text{Garb}, \text{Encode}, \text{Eval})$ with independent key generation, to construct a garbling scheme $\mathcal{GS}^A$ for $\{\mathcal{AL}_\lambda\}$, proceed in the following two steps:

Step 1: Construct a non-succinct garbling scheme: Observe that the computation of a machine AL of $AL.T$ steps can be divided into $AL.T$ 1-step “blocks” that transforms the current configuration to the next; therefore, to garble AL , it suffices to produce a sequence of “garbled blocks”, one for each 1-step block. The actual programs being garbled is an “augmented block”, whose execution consists of a 1-step block followed by the encoding algorithm of $\mathcal{GS}$ that encodes the output configuration for the next garbled block (when an output is produced, it is output directly without encoding). The final garbling then consists of a sequence of T garbled blocks.

Step 2: Compress the size using IO: As before, we then use **IO** to “compress” the size of the non-succinct garbling constructed in the first step, by giving the obfuscation of the algorithm that on input t , runs **Garb** to garble the t^{th} augmented block, producing the t^{th} garbled block. The obfuscated program is the succinct garbled program.

The efficiency and security analysis remains the same as before. This concludes Theorem 6.

4.1 Improved Construction and Analysis

Notice that our construction of $\mathcal{GS}^A$ uses the underlying circuit garbling scheme $\mathcal{GS}$ in a black-box way. In fact, the scheme does not even require the underlying garbling scheme to be for circuits—*any garbling scheme for any class of algorithms that is “complete”, in particular can be used to implement the augmented blocks suffices.* Below we show that by plugging in the one-time garbled RAM of [LO13, GHL⁺14], and modifying the construction of Theorem 6 slightly, we can improve the efficiency of $\mathcal{GS}^A$ when the algorithm class is RAM. More precisely, we show the following theorem.

Theorem 7. *Assuming the existence of IO for circuits and one-way functions. There exists a garbling scheme $\mathcal{GS}^{\text{RAM}}$ for RAM with linear-space-dependent complexity. Furthermore, for any RAM machine R and input x , such that $T^* = T_R(x)$, evaluation of a garbled pair $(\hat{R}, \hat{x})$ produced by the scheme takes time $T^* \times \text{poly}(\lambda, |R|)$.*

Towards this, we will rely on a RAM garbling scheme that has independent key generation and linear-time dependent complexity, and moreover, is *parallel computable* as defined below.

Parallel Computability: We say that a randomized algorithm $y = P(x; r)$ with K -bit outputs is t -time parallel computable, if there is a set of t -size circuits $\{P_i\}_{i \in [K]}$, such that, for every λ , x , and r ,

$$y_i = P_i^r(x[P_i]) \text{ and } y = y_1 \| y_2 \| \cdots \| y_K$$

where each $P_i^r(x[P_i])$ accesses (a few bits of) the shared random tape r and depends on a fixed subset P_i of bits of x , producing an output bit y_i . The running time of the algorithm is bounded by $K \cdot t$. The following fact will be instrumental later.

Fact 1: The composition of two parallel computable algorithms P and Q is also parallel computable. More precisely, given two randomized algorithms $y = P(x; r)$ and $z = Q(y; r')$ that are respectively t -time and t' -time parallel computable by $\{P_i^r(x[P_i])\}_{i \in [K]}$ and $\{Q_i^{r'}(y[Q_i])\}_{i \in [K']}$, their composition $R(x; r, r') = Q(P(x; r); r')$ is $(t \cdot t')$ -time parallel computable, by $\{R_i^{r, r'}(x[R_i])\}_{i \in [K']}$ defined as,

$$y_i = R_i^{r, r'}(x[R_i]) = Q_i^{r'}(P_{i_1}^r(x[P_{i_1}]), \dots, P_{i_l}^r(x[P_{i_l}])), \text{ where } Q_i = \{i_1, \dots, i_l\} \text{ and } R_i = \cup_{i_j \in Q_i} P_{i_j}$$

A Parallel Computable Garbling Scheme $p\mathcal{GS}$: Towards obtaining Theorem 7, we rely on a parallel computable garbling scheme $p\mathcal{GS} = (p\text{Garb}, p\text{Encode}, p\text{Eval})$ with the following properties. We remark all these properties are satisfied by the construction of [LO13, GH14]. Let R be a RAM machine with parameters n, m, S, T .

Independent key generation. $p\mathcal{GS}$ has independent key generation as defined in Definition 7 with a slight strengthening. We repeat the definition and highlight the strengthening.

- The garbling algorithm $p\text{Garb}$ consists of:

$$(\mathbf{key}, \hat{R}) \xleftarrow{\$} p\text{Garb}(1^\lambda, R) : \mathbf{key} \xleftarrow{\$} p\text{Gen}(1^\lambda), \hat{R} \xleftarrow{\$} p\text{Gb}(\mathbf{key}, R)$$

Strengthening: Different from Definition 7, the PPT key generation algorithm $p\text{Gen}$ depends only on the security parameter 1^λ and not on the length of the input $1^{|x|}$. As a result, the length of $\mathbf{key}$ produced is bounded by $\text{poly}(\lambda)$.

- The simulation procedure $p\text{Sim}$ consists of¹²:

$$(\tilde{R}, \tilde{x}) \xleftarrow{\$} p\text{Sim}(1^\lambda, (|R|, |x|, n, m, S, T), R(x)) : \\ (\tilde{x}, \mathbf{st}) \xleftarrow{\$} p\text{Sim} \cdot \text{Gen}(1^\lambda, |x|), \tilde{R} \xleftarrow{\$} p\text{Sim} \cdot \text{Gb}(1^\lambda, (|R|, |x|, n, m, S, T), R(x), \mathbf{st})$$

Rachel:

[change the definition of independent key generation to not depend on input length, and change the interface of simulation $\text{Sim}(1^\lambda, (|R|, |x|, n, m, S, T), T_R(x), R(x))$. Change the running time of Sim to depend on T .]

Linear complexity. The complexity of algorithms in the the garbling scheme is:

- The garbling algorithm $p\text{Garb}(1^\lambda, R)$ and evaluation algorithm $p\text{Eval}(\hat{R}, \hat{x})$ run in time $\text{poly}(\lambda) \times |R| \times T^{13}$, which also bounds the size Φ_R of the the garbled program $\hat{R}$.

¹²Note that the simulation procedure described below does not receive the instance running time $T_R(x)$. This is because, as seen shortly, the complexity of this RAM garbling scheme is linear in the time complexity of the RAM machine being garbled, and thus does not have instance based efficiency.

¹³In [LO13, GH14], the overhead of garbling is $\text{poly}(\lambda) \times |R| \times \text{poly} \log(n)$, where n is the size of the persistent memory data. Since in this work we do not consider RAM machine with persistent memory data, we ignore this term.

- The input encoding algorithm $\hat{x} \xleftarrow{\$} p\text{Encode}(\mathbf{key}, x)$ runs in time linear in the length of the input $\text{poly}(\lambda)|x|$, which also bounds the size Φ_x of the garbled input $\hat{x}$.

Parallel Computability. The following algorithms are parallel computable:

- The garbling algorithms $p\text{Gb}$ and the simulation procedure $p\text{Sim}\cdot\text{Gb}$ are $\text{poly}(\lambda)|R|$ -time parallel computable, and
- the encoding algorithm $p\text{Encode}$ and the input simulation procedure $p\text{Sim}\cdot\text{Gen}$ are $\text{poly}(\lambda)$ -time parallel computable.

More Efficient Garbling Scheme for Bounded Space RAM: We now use a parallel computable garbling scheme $p\mathcal{GS} = (p\text{Garb}, p\text{Encode}, p\text{Eval})$ with the above properties to construct a more efficient garbling scheme $\mathcal{GS}$ for bounded space RAM that has (1) *linear*-space dependent complexity and (2) produces garbled RAM with $\text{poly}(\lambda, |R|)$ overhead (that is, evaluation of $\hat{R}, \hat{x}$ takes $\text{poly}(\lambda, |R|)T_R(x)$ steps). In comparison, the previous general construction has *polynomial* space dependent complexity and $\text{poly}(\lambda, |R|, S)$ overhead.

Towards this, we plug in the parallel computable garbling scheme $p\mathcal{GS}$ with simulation $p\text{Sim}$ into our general construction with the following modifications.

Modification to Step 1: As before, the first step is constructing a non-succinct garbling scheme, by dividing a RAM computation into small blocks and garbling all of them using $p\mathcal{GS}$.

The only, and key, difference is, instead of dividing a T step RAM computation into T 1-step “blocks”, dividing it into $\lceil T/S \rceil$ S -step “blocks”. As before, each block is then augmented with the encoding algorithms $p\text{Encode}(\mathbf{key}, \cdot)$ for garbling the output configuration; and each augmented block is garbled using $p\text{Garb}$, producing garbled blocks.

Efficiency. We now analyze various efficiency parameters.

- Each augmented block, say the t^{th} , is a RAM consisting of S steps of computation of R followed by $\text{Encode}(\mathbf{key}_{t+1}, \cdot)$ ¹⁴—denote this program as $B(t, R, \mathbf{key}_{t+1}, \cdot)$. Since $\mathbf{key}_{t+1}$ has size $\text{poly}(\lambda)$, we have,

$$\text{size of } B = \Psi = |R| + \text{poly}(\lambda), \quad \text{run-time of } B = \text{poly}(\lambda)S$$

- By the efficiency of $p\text{Garb}$, each garbled block has size

$$\Phi = \text{poly}(\lambda)(\text{size of } B)(\text{run-time of } B) = \text{poly}(\lambda)|R|S$$

- Overall, there are $\lceil T/S \rceil$ blocks, resulting in a non-succinct garbled RAM $\hat{R}$ of size

$$|\hat{R}| = \lceil T/S \rceil \times \Phi = \text{poly}(\lambda)|R|T$$

- We note that for any input x of instance complexity T^* , the output $R(x)$ is produced after evaluating $\lceil T^*/S \rceil$ garbled blocks, taking $\text{poly}(\lambda)|R|T^*$ steps.

¹⁴It also has the additional logic for deciding whether the output configuration is a final configuration, and returns the output if so.

Modification to Step 2: As before, the second step is using obfuscation to “compress” the size of the non-succinct garbling scheme constructed in Step 1. However, if directly obfuscate the program that generates each of $\lceil T/S \rceil$ garbled blocks completely, it leads to an obfuscated program of size at least $\text{poly}(\lambda, \Phi) = \text{poly}(\lambda, |R|, S)$. In this case, the complexity of the new garbling scheme is not linear in S , and the overhead of the produced garbled RAM is at least $\text{poly}(\lambda, |R|, S)$.

Better efficiency: Instead of obfuscating the program that generates a whole garbled block, we rely on the parallel computability of $p\mathcal{GS}$ to obfuscate the decomposed small circuits that generates individual bits of each garbled block. More precisely, the program that produces the garbled blocks is

$$\hat{B}_t = \mathbb{P}^{\lambda, S, R, K_\alpha, K_\beta}(t) : \hat{B}_t = p\text{Gb}(\mathbf{key}_t, B(t, R, \mathbf{key}_{t+1}, \cdot); \beta_t) \text{ with } \mathbf{key}_j = \text{Gen}(1^\lambda; \alpha_j)$$

where all random coins $\alpha_t, \alpha_{t+1}, \beta_t$ are generated using a PRF with keys K_α, K_β . It follows directly from that $p\text{Gb}$ is $\text{poly}(\lambda)\Psi$ -parallel computable, where Ψ is the size of the program obfuscated (the augmented block in this case) that $\mathbb{P}$ is $\text{poly}(\lambda)|R|$ -parallel computable by circuits $\{\mathbb{P}_i^{\lambda, S, R, K_\alpha, K_\beta}(t)\}_{i \in \Phi}$ (recall that Φ is the size of each garbled block). Thus to obtain better efficiency, obfuscate each $\mathbb{P}_i$ independently. The new garbled program $\hat{R}$ is the array of obfuscated programs:

$$\hat{R} = (\bar{\mathbb{P}}_1, \dots, \bar{\mathbb{P}}_\Phi), \text{ where } \bar{\mathbb{P}}_i \stackrel{\$}{\leftarrow} i\mathcal{O}(1^\lambda, \mathbb{P}_i) .$$

Evaluation of the garbled RAM proceeds the same as before, except that the t^{th} garbled block is generated by evaluating the array of obfuscated programs on input t .

The security proof: Security follows the same blue-print as before—the simulated program $\tilde{R}$ is the obfuscation of a program $\mathbb{Q}$ that produces simulated garbled-blocks using the simulation procedure $p\text{Sim}\cdot\text{Gen}$ and $p\text{Sim}\cdot\text{Gb}$ of $p\mathcal{GS}$. The indistinguishability of $\tilde{R}$ and $\hat{R}$ is then established through a sequence of hybrids where the obfuscated program simulates the first j garbled-blocks, and generates the rest honestly; to stitch these two parts together, the j^{th} garbled block is produced using a program $\mathbb{R}$ that invokes $p\text{Sim}\cdot\text{Gen}$, $p\text{Sim}\cdot\text{Gb}$ and $p\text{Encode}$ of $p\mathcal{GS}$. The only difference now is that $\mathbb{Q}$ and $\mathbb{R}$ are not obfuscated directly, but rather their decomposition $\{\mathbb{Q}_i\}_{i \in [\Phi]}$ and $\{\mathbb{R}_i\}_{i \in [\Phi]}$ is obfuscated: It follows from the fact that $p\text{Sim}\cdot\text{Gen}$, and $p\text{Encode}$ are $\text{poly}(\lambda)$ -parallel computable and $p\text{Sim}\cdot\text{Gb}$ is $\text{poly}(\lambda)\Psi$ -parallel computable that both $\mathbb{Q}$ and $\mathbb{R}$ are $\text{poly}(\lambda)|R|$ -parallel computable. Then in simulation and hybrids, $\mathbb{Q}_i$ and $\mathbb{R}_i$ are obfuscated independently.

Efficiency. Since each $\mathbb{P}_i$, $\mathbb{Q}_i$ and $\mathbb{R}_i$ have size $\text{poly}(\lambda)|R|$, each obfuscated program has size $\text{poly}(\lambda, |R|)$. Therefore, the size of the new garbled RAM (and the complexity for generating it) is,

$$\text{size of garbled RAM} = \Phi \times \text{poly}(\lambda, |R|) = \text{poly}(\lambda, |R|) \times S ,$$

which is linear in the space complexity of R .

Moreover, evaluation of an input x of instance complexity T^* requires generating and evaluating $\lceil T^*/S \rceil$ garbled blocks, which takes time

$$\text{run-time of garbled RAM} = \lceil T^*/S \rceil \times \text{poly}(\lambda, |R|) \times S = \text{poly}(\lambda, |R|) \times T^* .$$

This concludes Theorem 7.

5 From Garbling to FE to Reusable Garbling

We observe that in contexts such as secure computation [GMW87] and functional encryption [SW05, O’N10, BSW12], to evaluate a function f on an input x , it suffices to evaluate the *randomized* function that computes a garbled program of f and an encoding of the input (recall that by the security of the garbling scheme this reveals no more than the output of the function). Thus, by plugging-in our construction of garbling schemes with space-dependent complexity into earlier constructions of secure computation or *randomized* functional encryption [GJKS], we directly obtain, assuming **IO** for $P/poly$ and one-way functions, a randomized functional encryption with space-dependent complexity, and secure computation protocols whose communication complexity grows polynomially with the space complexity of the program to be evaluated, but only logarithmically with the running-time.

We additionally observe that by combining our construction of functional encryption with previous results [CIJ⁺13, GKP⁺13b]—[CIJ⁺13] showed that function encryption schemes with indistinguishability based security implies ones with simulation-based security, which further implies reusable garbling schemes by [GKP⁺13b]—directly yields a construction of *reusable* succinct garbling schemes with space-dependent complexity from **IO** for $P/poly$ and one-way functions.

We emphasize that the above applications work generally for any “nice” class of algorithms; (conditions on the algorithm classes are specified in the theorem statements below). Therefore, below we show the connections between Garbling, randomized functional encryption and reusable garbling w.r.t. general classes of algorithms.¹⁵ Applications to specific models of computation can then be derived as special cases.

Below, we start with the definitions of function encryption and reusable garbling scheme, and then move to showing the connections.

5.1 Functional Encryption

We first recall definitions of public key functional encryption schemes, both the indistinguishability-based definitions and simulation-based definitions, adapting to general algorithm classes. For indistinguishability-based security, we recall the definition for randomized algorithms in [GJKS], whereas for simulation-based security, we recall the definition of [BSW12, O’N10] for deterministic Boolean algorithms. Below we first introduce the syntax, then various security notions and finally the efficiency guarantees.

Syntax. We introduce the syntax and correctness of functional encryption scheme w.r.t. classes of potentially randomized algorithms, which immediately imply the syntax and correctness for deterministic algorithms.

Definition (Functional Encryption.). *A functional encryption scheme $\mathcal{FE}$ for a class of (well-formed) (potentially randomized) algorithms $\{\mathcal{AL}_\lambda\}_{\lambda \in \mathbb{N}}$ consists of algorithms (FE.Setup, FE.Enc, FE.KeyGen, FE.Dec) where the first three are probabilistic algorithms while the last is deterministic; furthermore, they satisfy the following syntax and correctness:*

- FE.Setup(1^λ) outputs a public key MPK and a master secret key MSK .
- FE.Enc(x, MPK) outputs a ciphertext CT .
- FE.KeyGen(AL, MSK) on input an algorithm $AL \in \mathcal{AL}_\lambda$, outputs a secret key sk_{AL} .

¹⁵Since the application to MPC is straightforward, we omit the details of the proof below.

- $\text{FE.Dec}(\text{CT}, \text{sk}_{AL})$ outputs a string y .

Correctness: For every polynomial T and polynomial $n = n(\lambda)$, every sequence of algorithm tuples $\{\vec{AL} = \vec{AL}_\lambda\} \in \{\mathcal{AL}_\lambda^n\}$ and every sequence of input tuples $\{\vec{x} = \vec{x}_\lambda\}$ where $x_i \in \{0, 1\}^{\text{poly}(\lambda)}$, the following two distributions are computationally indistinguishable:

- **Real:** $\{\text{FE.Dec}(\text{CT}_i, \text{SK}_{AL_j})\}_{i \in [n], j \in [n]}$ where,
 $(\text{MPK}, \text{MSK}) \xleftarrow{\$} \text{FE.Setup}(1^\lambda)$
 $\text{CT}_i \xleftarrow{\$} \text{FE.Enc}(\text{MPK}, x_i)$ for $i \in [n]$
 $\text{sk}_{AL_j} \xleftarrow{\$} \text{FE.KeyGen}(AL_j, \text{MSK})$ for $j \in [n]$
- **Ideal:** $\{AL_j(x_i; r_{ij})\}_{i \in [n], j \in [n]}$ where,
for every $j \in [n]$, $\{r_{ij}\}_{i \in [n]}$ are sufficiently long random strings if AL_j is a randomized algorithm and are empty if AL_j is a deterministic algorithm. (In the case that x_i exceeds the input length bound of the algorithm AL_j , $|x_i| > AL_j.n$, the output is set to $\perp$).

Indistinguishability based security. We provide the definition of indistinguishability based security for classes of (potentially randomized) algorithms. Our definition is a generalization of that in [GJKS] to the case of full security.

Definition 13 ((Full) q_1 - ℓ - q_2 -IND-security). A functional encryption scheme $\mathcal{FE}$ is (ful-) q_1 - ℓ - q_2 -IND secure if for every polynomial T , the advantage of any PPT adversary $\mathcal{A}$ in the following game is negligible.

Experiment (ful-) q_1 - ℓ - q_2 -IND- $\mathcal{FE}_{\mathcal{A}}^{\mathcal{FE}}(1^\lambda, T = T(\lambda))$:

1. *Setup:* $(\text{MPK}, \text{MSK}) \xleftarrow{\$} \text{FE.Setup}(1^\lambda)$
2. *Non-Adaptive Key Queries:* $(\vec{m}_0, \vec{m}_1, \text{st}_1) \xleftarrow{\$} \mathcal{A}^{\text{FE.KeyGen}(\text{MSK}, \cdot), \mathcal{O}_1(\cdot)}(\text{MPK})$.
3. *Generate Challenge Ciphertext:* $\forall i, \text{CT}_i \xleftarrow{\$} \text{FE.Enc}(\text{MPK}, m_{b,i})$, where $b \xleftarrow{\$} \{0, 1\}$.
4. *Adaptive Key Queries:* $\alpha \xleftarrow{\$} \mathcal{A}^{\text{FE.KeyGen}(\text{MSK}, \cdot), \mathcal{O}_2(\cdot)}(\text{st}_1, \vec{\text{CT}})$.

$\mathcal{O}_1$ is a stateful oracle that on input (AL, null) , generates a secret key $\text{sk}_{AL} \xleftarrow{\$} \text{FE.KeyGen}(\text{MSK}, AL)$ and records it internally, and on input (AL, C) checks if a secret key has been generated for AL , and returns $\text{FE.Dec}(\text{sk}_{AL}, C)$ if it is the case ($\perp$ otherwise). $\mathcal{O}_2$ is identical to $\mathcal{O}_1$ except that it only decrypts ciphertext $C \neq \text{CT}_i$ for all i . The advantage of $\mathcal{A}$ is $\Pr[\alpha = b] - 1/2$.

Restriction on $\mathcal{A}$: Let S_1 and S_2 represent the sets of non-adaptive and adaptive key queries made by $\mathcal{A}$ in Step 2 and 4 respectively; let Q_1, Q_2 be the sets of key queries $\mathcal{A}$ submits to $\mathcal{O}_1$ and $\mathcal{O}_2$ respectively. $\mathcal{A}$ must follow the restriction that 0) $S_1, S_2, Q_1, Q_2 \subseteq \mathcal{AL}_\lambda^T$, 1) $|S_1| \leq q_1(\lambda)$, 2) $|S_2| \leq q_2(\lambda)$, 3) $|\vec{m}_0| = |\vec{m}_1| \leq \ell$ and for every $i \in [|\vec{m}_0|]$ $|m_{0,i}| = |m_{1,i}|$, and 4) for every i, j , let AL_j be the j^{th} query in $S_1 \cup S_2$, the following distributions are indistinguishable:

$$\begin{aligned} \{r \xleftarrow{\$} \{0, 1\}^{\text{poly}(\lambda)} : (AL_j(m_{0,i}; r), T_{AL}(m_{0,i}; r))\}_\lambda \\ \{r \xleftarrow{\$} \{0, 1\}^{\text{poly}(\lambda)} : (AL_j(m_{1,i}; r), T_{AL}(m_{1,i}; r))\}_\lambda \end{aligned}$$

Additionally, we consider the following weaker notions.

Definition 14 (Selective q_1 - ℓ - q_2 -IND-security). *A functional encryption scheme $\mathcal{FE}$ is selective q_1 - ℓ - q_2 -IND secure if for every polynomial T , the advantage of any PPT adversary $\mathcal{A}$ is negligible in the $\text{sel-}q_1$ - ℓ - q_2 -IND($1^\lambda, T$) game, which is the same as the above q_1 - ℓ - q_2 -IND game, except that $\mathcal{A}$ is required to select the challenge messages $\vec{m}_0, \vec{m}_1$ at the beginning of experiment before MPK, MSK are chosen.*

Note that in the selective game $\text{sel-}q_1$ - ℓ - q_2 -IND, it is without loss of generality to remove Step 2 in the experiment; thus, we can assume w.l.o.g. that $q_1 = 0$. Furthermore, it follows from standard hybrid argument that for any polynomials q_1, ℓ, q_2 , q_1 -1- q_2 -IND security implies q_1 - ℓ - q_2 -IND security, both in the selective and full game. Thus in the rest of the paper we use interchangeably q_1 -1- q_2 -IND security and q_1 - ℓ - q_2 -IND security.

Definition 15 (Honest-Sender (full or selective) q_1 - ℓ - q_2 -IND-Security). *We say that a functional encryption scheme is honest-sender (full or selective) q_1 - ℓ - q_2 -IND secure, if it satisfies the same security condition as in Definition 13, except that, in the experiments the oracles $\mathcal{O}_1$ and $\mathcal{O}_2$ are empty.*

When the class of algorithms are deterministic, the standard definition in the literature considers only honest-sender security. We follow this convention; when an algorithm class is deterministic only security against malicious receiver is considered.

Definition ((Honest-sender) Full or Selective IND-security). *A functional encryption scheme $\mathcal{FE}$ is (honest-sender) ful-IND-secure or sel-IND-secure, if it is (honest-sender) ful-poly-1-poly-IND-secure or sel-0-1-poly-IND-secure.*

Simulation based security. Next we proceed to define simulation based security notions. In this case, we consider only classes of algorithms $\{\mathcal{AL}_\lambda\}$ that are *deterministic*. Furthermore, we note that it is without loss of generality to consider only Boolean algorithms. This is because a functional encryption scheme for Boolean algorithms can be easily turned into a scheme for algorithms with m -bit outputs, by running the Boolean scheme for m times in parallel. Such parallel repetition leads to a scheme where the ciphertext length is linear in $n \times m$ (as well as in λ), where n is the input length. On the other hand, it was shown that simulation-based secure functional encryption must have the size of ciphertexts growing linearly with the output length (if there is any non-adaptive queries made before the challenge ciphertexts are generated). Thus, the parallel repetition is essentially optimal.

Definition. *A functional encryption scheme $\mathcal{FE}$ for a class $\{\mathcal{AL}_\lambda\}$ of deterministic Boolean algorithms with is (ful-) q_1 - ℓ - q_2 -SIM secure if for every PPT adversary $\mathcal{A}$, there exists a simulator $\text{Sim} = (\text{Sim}_1, \text{Sim}_2)$ such that the output of the following two experiments are indistinguishable for every polynomial T .*

Experiment $(\text{ful-})\mathbf{q_1-l-q_2-RealExp}_{\mathcal{A}}^{\mathcal{FE}}(1^\lambda, T = T(\lambda))$:

1. *Setup*: $(\text{MPK}, \text{MSK}) \xleftarrow{\$} \text{FE.Setup}(1^\lambda)$
2. *Non-Adaptive Key Queries*: $(\vec{m}, \mathbf{st}_1) \xleftarrow{\$} \mathcal{A}^{\text{FE.KeyGen}(\text{MSK}, \cdot)}(\text{MPK})$.
3. *Generate Challenge Ciphertext*: $\forall i, \text{CT}_i \xleftarrow{\$} \text{FE.Enc}(\text{MPK}, m_i)$, where $b \xleftarrow{\$} \{0, 1\}$.
4. *Adaptive Key Queries*: $\alpha \xleftarrow{\$} \mathcal{A}^{\text{FE.KeyGen}(\text{MSK}, \cdot)}(\mathbf{st}_1, \vec{\text{CT}})$.
5. *Output* $(\text{MPK}, \vec{m}, S_1, S_2, \alpha)$, where S_1 and S_2 are the sets of non-adaptive and adaptive key queries made by $\mathcal{A}$ in Step 2 and 4 respectively.

Experiment $(\text{ful-})\mathbf{q_1-l-q_2-IdealExp}_{\mathcal{A}}^{\mathcal{FE}}(1^\lambda, T = T(\lambda))$:

1. *Setup*: $(\text{MPK}, \text{MSK}) \xleftarrow{\$} \text{FE.Setup}(1^\lambda)$
2. *Non-Adaptive Key Queries*: $(\vec{m}, \mathbf{st}_1) \xleftarrow{\$} \mathcal{A}^{\text{FE.KeyGen}(\text{MSK}, \cdot)}(\text{MPK})$; let $\ell' = |\vec{m}|$
3. *Generate Challenge Ciphertext*: $(\vec{\text{CT}}, \mathbf{st}') \xleftarrow{\$} \text{Sim}_1(\text{MPK}, \ell', \{|m_i|\}, \mathcal{V})$
4. *Adaptive Key Queries*: $\alpha \xleftarrow{\$} \mathcal{A}^{\mathcal{O}(\cdot)}(\mathbf{st}_1, \vec{\text{CT}})$
5. *Output* $(\text{MPK}, \vec{m}, S_1, S_2, \alpha)$.

In the above experiment $\mathcal{V}$ in Step 3 contains the outputs of the algorithms in S_1 applied to $\vec{m}$, that is, $\mathcal{V} = \cup_{Q \in S_1} \mathcal{V}_Q$ with $\mathcal{V}_Q = \{Q, \text{sk}_Q, T_Q(m_i), Q(m_i)\}_{i \in [\ell']}$. Additionally, the oracle $\mathcal{O}(\cdot)$ in Step 4 is the second stage simulator, namely $\text{Sim}_2(\mathbf{st}', \text{MSK}, \cdot, \cdot)$, where the third argument is a query circuit Q and the fourth argument is $\mathcal{V}_Q$.

Restriction on $\mathcal{A}$ in the above two experiments: 0) $S_1, S_2 \subseteq \mathcal{AL}_\lambda^T$ 1) $|S_1| \leq q_1(\lambda)$, 2) $|S_2| \leq q_2(\lambda)$, 3) $|\vec{m}| \leq \ell$.

Additionally, a functional encryption scheme $\mathcal{FE}$ is (honest-sender) **selective $\mathbf{q_1-l-q_2-SIM}$ secure** if for every PPT adversary $\mathcal{A}$, the outputs of the experiments $\text{sel-}\mathbf{q_1-l-q_2-RealExp}$ and $\text{sel-}\mathbf{q_1-l-q_2-IdealExp}$ are indistinguishable for every polynomial T , where the two games are the same as the above, except that $\mathcal{A}$ is required to select the challenge messages $\vec{m}_0, \vec{m}_1$ at the beginning of experiment before MPK, MSK are chosen (and without access to the FE.KeyGen oracle).

We note that in the above definition, the second state simulator Sim_2 receive as input the instance running time $T_Q(m_i)$ for every Q and m_i . This is necessary since the efficiency guarantees below require the decryption algorithm to have instance-based efficiency.

Efficiency. Finally, We now move to define different efficiency requirement for functional encryption.

Definition (Different Levels of Efficiency of Functional Encryption Scheme). *We say that a functional encryption scheme $\mathcal{FE}$ has optimal efficiency, or I/O- / space- / linear-time dependent complexity if the following conditions hold.*

Optimal efficiency: *Algorithms FE.Setup, FE.Enc, FE.KeyGen run in time polynomial in their input lengths, that is $p_{\text{Setup}}(\lambda)$, $p_{\text{Enc}}(\lambda, |x|)$, $p_{\text{KeyGen}}(\lambda, |AL|)$, and FE.Dec runs in time $p_{\text{Dec}}(\lambda, |AL|, |x|)T_{AL}(x)$ for a polynomial p_{Dec} .*

I/O-dependent complexity: *The above condition holds, except that the running time of FE.KeyGen and FE.Dec additionally depends on $AL.n$, $AL.m$.*

Space-dependent complexity: *The above condition holds, except that the running time of FE.KeyGen and FE.Dec additionally depends on $AL.S$.*

linear-time-dependent complexity: *The above condition holds, except that the running time of FE.KeyGen and FE.Dec depends quasi-linearly on $AL.T$.*

We note that the FE.Setup and FE.Enc always runs polynomially in their input length, independent of the parameters of the algorithms that are potentially to be evaluated. Furthermore, in the case of SIM-secure functional encryptions, we only consider Boolean algorithms; thus, it is possible to have schemes with optimal efficiency, where the encryption and key generation algorithms runs in time independent of the output length.

5.2 Reusable Garbling Scheme

We recall the definition of reusable garbled circuits in [GKP⁺13b], adapted for general algorithm classes. In [GKP⁺13b], their definition considers adaptive input selection, and their construction achieves so. An alternative definition with static input selection is considered in [GHRW14] for reusable garbled RAM. The two variants corresponds tightly with fully or selectively SIM-secure functional encryption schemes. Below we define both variants.

Definition 16 (Reusable Security [GKP⁺13b]). *We say that a garbling scheme $\mathcal{GS}$ for a class of deterministic Boolean algorithms $\{\mathcal{AL}_\lambda\}_{\lambda \in \mathbb{N}}$ has reusable security with adaptive input selection, if for all PPT adversary $\mathcal{A}$ there is a simulator $\text{Sim} = (\text{Sim}\cdot\text{Gb}, \text{Sim}\cdot\text{Inp})$ the following two games are indistinguishable for all polynomial T .*

Experiment $\text{RealExp}_A^{\mathcal{GS}}(1^\lambda, T = T(\lambda))$:

1. $(AL, \text{st}_1) \xleftarrow{\$} \mathcal{A}(1^\lambda)$
2. $(\hat{AL}, \text{key}) \xleftarrow{\$} \text{Garb}(1^\lambda, AL)$
3. $\alpha \xleftarrow{\$} \mathcal{A}^{\text{Encode}(\text{key}, \cdot)}(\text{st}_1, \hat{AL})$

Experiment $\text{IdealExp}_A^{\mathcal{GS}}(1^\lambda, T = T(\lambda))$:

1. $(AL, \text{st}_1) \xleftarrow{\$} \mathcal{A}(1^\lambda)$
2. $(\tilde{AL}, \text{st}) \xleftarrow{\$} \text{Sim}\cdot\text{Gb}(1^\lambda, |AL|, (AL.n, AL.m, AL.S, AL.T))$
3. $\alpha \xleftarrow{\$} \mathcal{A}^{\mathcal{O}(\cdot)}(\text{st}_1, \tilde{AL})$

where the oracle in the third step is the second stage simulator $\text{Sim}\cdot\text{Inp}(\text{st}, \cdot, \cdot, \cdot)$ that receives as the second parameter the length of the input $1^{|x|}$, as the third parameter the instance running time $T_{AL}(x)$ and the fourth parameter the output $AL(x)$.

Restriction on $\mathcal{A}$ in the above two experiments: the algorithm AL chosen by $\mathcal{A}$ is in $\mathcal{AL}_\lambda^T$.

Furthermore, we say that $\mathcal{GS}$ has reusable security with selective input selection, if the above indistinguishability is true for all PPT adversary $\mathcal{A}$ that selects its oracle queries in the third step at the beginning of the games.

5.3 IND-secure FE for General Classes of Randomized Algorithm

We show a general method for constructing (full or selective) IND-secure functional encryption for a general class $\{\mathcal{AL}_\lambda\}$ of (potentially randomized) algorithms, from a garbling scheme $\mathcal{GS} = (\text{Garb}, \text{Encode}, \text{Eval})$ for the corresponding class of “de-randomized” algorithms, and a (full or selective respectively) IND-secure functional encryption for circuits.

Towards this, consider a randomized algorithm AL , let $\text{deRand}[AL, k]$ be the following de-randomized algorithm corresponding to AL , where $k \xleftarrow{\$} \text{PRF}\cdot\text{Gen}(1^\lambda)$ is a PRF key.

$\text{deRand}^F[AL, k](x) : \text{Run } AL(x) \text{ with pseudo-random coins } r_t = F(k, t) \text{ for step } t$

Then the high-level idea for constructing a IND-secure FE $\mathcal{FE}^A$ for $\{\mathcal{AL}_\lambda\}$ is the following: The secret key corresponding to algorithm AL will be the secret key generated using the IND-secure circuit-FE $\mathcal{FE}^C$ for the (randomized) algorithm that garbles the de-randomized C_{AL} as described above. More specifically, the circuit C_{AL} is constructed as follows:

$(\hat{\Gamma}, \hat{x}) \xleftarrow{\$} C_{AL}(x) : k \xleftarrow{\$} \text{PRF}\cdot\text{Gen}(1^\lambda), (\hat{\Gamma}, \text{key}) \xleftarrow{\$} \text{Garb}(1^\lambda, \text{deRand}^F[AL, k]), \hat{x} \xleftarrow{\$} \text{Encode}(\text{key}, x)$

Below we state our result formally. Note that since the garbling algorithm must work with algorithms of the form $\text{deRand}^F[AL, k]$, we require that it is for an algorithms class that is *closed under composition with polynomial-sized circuits*, which implies that for every $AL \in \mathcal{AL}_\lambda$, $\text{deRand}^F[AL, k]$ is in $\mathcal{AL}_\lambda$.

Proposition 2 (IND-secure FE for General Classes of Randomized Algorithms). *Let $\{\mathcal{AL}_\lambda\}$ be any class of (potentially randomized) algorithms that is closed under composition with polynomial-sized circuits. It holds that if there are*

- i) a garbling scheme $\mathcal{GS} = (\text{Garb}, \text{Encode}, \text{Eval})$ for deterministic algorithms in $\{\mathcal{AL}_\lambda\}$.
- ii) a (ful-)IND-secure (or sel-IND-secure) functional encryption scheme $\text{IND-}\mathcal{FE}^C$ for the class rCIR of polynomial-sized randomized circuits.
- **then**, there is a (ful-)IND-secure (or sel-IND-secure respectively) functional encryption scheme $\text{IND-}\mathcal{FE}^A$ for $\{\mathcal{AL}_\lambda\}$.

Furthermore, if $\mathcal{GS}$ has optimal efficiency or I/O-dependent complexity, $\text{IND-}\mathcal{FE}^A$ has I/O-dependent complexity. If $\mathcal{GS}$ has space- / linear-time- dependent complexity, so does $\text{IND-}\mathcal{FE}^A$.

Proof. Below we give the construction and proof. Given an IND-secure circuit FE $\mathcal{FE}^C = (\text{FE.Setup}, \text{FE.KeyGen}, \text{FE.Enc}, \text{FE.Dec})$, our IND-secure FE is constructed as follows: It has the same setup and encryption algorithm. The key generation and decryption algorithms invokes the algorithms FE.KeyGen and FE.Dec as sub-routines. To generate a key for algorithm AL , $\mathcal{FE}^A$ generates a key $\text{sk}_{C_{AL}}$ using $\mathcal{FE}^C$ for the circuit C_{AL} as described above and returns $\text{sk}_{AL} = \text{sk}_{C_{AL}}$. To decrypt a ciphertext c of value x , it invokes the decryption algorithm $\text{FE.Dec}(\text{sk}_{C_{AL}}, c)$ to obtain a pair $(\hat{\Gamma}, \hat{x})$, and then evaluates the garbled pair to obtain an output y .

The IND-security of $\mathcal{FE}^A$ reduces to the IND-security of $\mathcal{FE}^C$. This follows as an adversary $\mathcal{A}$ attacking the former can be transformed into an adversary $\mathcal{A}'$ attacking the latter. More specifically, in an $\text{IND}_{\mathcal{A}'}^{\mathcal{FE}^C}$ game, the adversary $\mathcal{A}'$ can internally emulate a game $\text{IND}_{\mathcal{A}}^{\mathcal{FE}^A}$ for $\mathcal{A}$: For every key query AL from $\mathcal{A}$, $\mathcal{A}'$ translates AL to a key query C_{AL} to its own key generation oracle, and for every decryption query c from $\mathcal{A}$, $\mathcal{A}'$ simply relays c to its own decryption oracles, and upon receiving an output $(\hat{\Gamma}, \hat{x})$, it evaluates the garbled pair to obtain y and feeds $\mathcal{A}$ with y .

It is easy to see that $\mathcal{A}$ emulates the view of $\mathcal{A}'$ perfectly. The only thing we need to establish is that whenever $\mathcal{A}$ sends a legitimate key query AL —that is, for the two challenge messages x_1 and x_2 , the outputs and running time of AL on x_1 and x_2 are indistinguishable—the translated key queries C_{AL} is also legitimate—that is, its outputs on x_1 and x_2 are indistinguishable. Below we argue this.

It follows from the pseudo-randomness of PRF and the security of the garbling scheme that for any polynomial T , any T -time randomized algorithm AL and any two inputs x_1, x_2 such that $AL(x_1; r), T_{AL}(x_1, r)$ for random r is indistinguishable from $AL(x_2; r), T_{AL}(x_2, r)$ for random r , we have that the outputs of C_{AL} on input x_1 and x_2 are also indistinguishable. More precisely, for every polynomial T ,

$$\begin{aligned} \forall \quad & \{AL\}_\lambda \in \{\mathcal{AL}_\lambda^T\}, \{x_1\}_\lambda, \{x_2\}_\lambda, \\ \text{if} \quad & \left\{ r \xleftarrow{\$} \{0, 1\}^{AL.T} : AL(x_1; r), T_{AL}(x_1, r) \right\}_\lambda \approx \left\{ r \xleftarrow{\$} \{0, 1\}^{AL.T} : AL(x_2; r), T_{AL}(x_2, r) \right\}_\lambda, \\ \text{then,} \quad & \{C_{AL}(x_1)\}_\lambda \approx \{C_{AL}(x_2)\}_\lambda \end{aligned}$$

This follows since for each $i \in \{1, 2\}$, it follows from the pseudo-randomness of the PRF that the output and running time of $\text{deRand}^F[AL, k]$ on x_1 and x_2 are indistinguishable, when the PRF key is chosen at random, that is,

$$\begin{aligned} & \left\{ k \xleftarrow{\$} \text{PRF.Gen}(1^\lambda) : \text{deRand}^F[AL, k](x_1), T_{\text{deRand}[AL, k]}(x_1) \right\}_\lambda \\ & \approx \left\{ k \xleftarrow{\$} \text{PRF.Gen}(1^\lambda) : \text{deRand}^F[AL, k](x_2), T_{\text{deRand}[AL, k]}(x_2) \right\}_\lambda, \end{aligned}$$

Then it follows from the security of the garbling scheme that the output of $C_{AL}(x_1)$, which is a freshly generated garbled pair $\{\hat{\Gamma}, \hat{x}_1\}$, can be simulated by random variables in the first ensemble

above, and that of $C_{AL}(x_2)$ can be simulated by variables in the second ensembles. Therefore, the indistinguishability of $C_{AL}(x_1)$ and $C_{AL}(x_2)$ holds.

This concludes the security of $\mathcal{FE}^A$. $\square$

Combining with known garbling schemes for circuits [Yao86], and TM and RAM with space-dependent complexity (our construction), the above lemma immediately implies the following theorem:

Theorem 8. *The following statements hold:*

- Assume the existence of randomized functional encryption for NC^1 , there is a randomized functional encryption for P/poly .
- Assume the existence of randomized functional encryption for P/poly , there is a randomized functional encryption scheme for TM and RAM with space-dependent complexity.

5.4 From IND security to SIM security.

The work by De Caro et. al. [CIJ⁺13] showed a tight connection between functional encryption with indistinguishability-based security and that with simulation-based security. More precisely, they provide a generic transformation that turns any (full or selective) $\text{q-}\ell\text{-poly-IND}$ secure functional encryption scheme $\text{IND-}\mathcal{FE}$ for polynomial-sized circuits to a (full or selective) $\text{q-}\ell\text{-poly-SIM}$ functional encryption scheme $\text{SIM-}\mathcal{FE}$ for polynomial-sized circuits, assuming one-way functions.

With a closer examination of their transformation, it is easy to see that their transformation works for general classes of algorithms (beyond circuits). Below we state the generic transformation theorem implicit in their security proof.

Proposition 3 (Generic Transformation from FE with IND-Security to SIM-Security (Implicit in [CIJ⁺13])). *For every class $\{\mathcal{AL}_\lambda\}$ of deterministic Boolean algorithms that are closed under composition with polynomial-sized circuits, the following holds: Assuming the existence of one-way functions, for every polynomial q and ℓ ,*

- *if there is a (ful-) $\text{q-}\ell\text{-poly-IND-secure}$ (or $\text{sel-}\text{q-}\ell\text{-poly-IND-secure}$) functional encryption scheme $\text{IND-}\mathcal{FE}$ for the class of algorithms $\{\mathcal{AL}_\lambda\}$.*
- *then there is a (ful-) $\text{q-}\ell\text{-poly-SIM-secure}$ (or $\text{sel-}\text{q-}\ell\text{-poly-SIM-secure}$ respectively) functional encryption scheme $\text{SIM-}\mathcal{FE}$ for $\{\mathcal{AL}_\lambda\}$,*

Furthermore, the following efficiency preservation holds.

- *if $\text{IND-}\mathcal{FE}$ has optimal efficiency, or space- / linear-time-dependent complexity, so does $\text{SIM-}\mathcal{FE}$, and*
- *if $\text{IND-}\mathcal{FE}$ has succinct ciphertexts, so does $\text{SIM-}\mathcal{FE}$.*

Note that the fact that the resulting SIM-secure functional encryption scheme can have optimal efficiency (if the underlying IND-secure functional encryption has) does not contradict with the lower bound that the ciphertext and key sizes must scale with the output length, because we consider only Boolean algorithms. SIM-secure functional encryption scheme for multi-bit algorithms can be constructed via parallel repetition of a scheme for Boolean algorithms; in this case, the ciphertext and key sizes do scale with the output length.

5.5 From SIM-secure FE to Reusable Garbling Scheme

The work by Goldwasser et. al. [GKP⁺13b] give a transformation from a (ful-)1-1-0-SIM-secure functional encryption scheme for circuits to a fully secure reusable garbling scheme. It is straightforward to see that their construction works for general classes of algorithms and when considering reusable garbling schemes with only static input selection, only sel-1-1-0-SIM-secure functional encryption is needed.

Proposition 4 (Generic Transformation from 1-1-0-SIM-Secure FE to Reusable Garbling Scheme (Implicit in [GKP⁺13b])). *For every class $\{\mathcal{AL}_\lambda\}$ of well-formed deterministic Boolean algorithms that are closed under composition with polynomial-sized circuits, the following holds: Assuming the existence of one-way functions,*

- *if there is a (ful-)1-1-0-SIM-secure (or sel-1-1-0-SIM-secure) functional encryption scheme $\text{SIM-}\mathcal{FE}$ for the class of algorithms $\{\mathcal{AL}_\lambda\}$,*
- *then there is a reusable garbling scheme $\mathcal{GS}$ for $\{\mathcal{AL}_\lambda\}$ with adaptive (or static) input selection.*

Furthermore, the following efficiency preservation holds.

- *if $\text{SIM-}\mathcal{FE}$ has optimal efficiency or I/O- /space- / linear-time-dependent complexity, so does $\mathcal{GS}$, and*
- *if $\text{SIM-}\mathcal{FE}$ has succinct ciphertexts, $\mathcal{GS}$ has succinct input encodings.*

Combining Proposition 2, 3 and 4, with our construction of garbling schemes with space-dependent complexity, we obtain the following.

Theorem 9. *Assume the existence of $\mathbf{iO}$ for P/poly and one-way function, there is re-usable garbling scheme for TM and RAM with space dependent complexity.*

6 Garbling v.s. $\mathbf{iO}$

In this section we show a connection between (succinct) $\mathbf{iO}$ for classes of algorithms $\mathcal{C}$ and (succinct) garbling schemes for the same classes $\mathcal{C}$. Roughly speaking,

- assuming the existence of (succinct) $\mathbf{iO}$ for any “nice” class $\mathcal{C}$ of algorithms and one-way functions, there exists a (succinct) garbling scheme for $\mathcal{C}$;
- assuming the existence of $\mathbf{iO}$ for P/poly and a (succinct) garbling scheme for any “nice” class $\mathcal{C}$ of algorithms, both with sub-exponential security, there exists a (succinct) $\mathbf{iO}$ for $\mathcal{C}$.

6.1 From Garbling to $\mathbf{iO}$

We present a generic transformation from a garbling scheme for an algorithm class $\{\mathcal{AL}_\lambda\}$ to an indistinguishability obfuscator for $\{\mathcal{AL}_\lambda\}$, assuming sub-exponentially indistinguishability obfuscators for circuits. We require that the algorithm class to have the property that for any $\lambda < \lambda' \in \mathbb{N}$, it holds that every algorithm $AL \in \mathcal{AL}_\lambda$ is also contained in $\mathcal{AL}_{\lambda'}$ —we say that such a class is “monotonically increasing”. For instance, the class of Turing machines TM and RAM machines RAM are all monotonically increasing.

Proposition 5. Let $\{\mathcal{AL}_\lambda\}$ be any monotonically increasing class of deterministic algorithms. It holds that if there are

- i) a sub-exponentially indistinguishable $\mathbf{iO}$, $i\mathcal{O}^C$, for circuits, and
- ii) a sub-exponentially indistinguishable garbling scheme $\mathcal{GS}$ for $\{\mathcal{AL}_\lambda\}$.
- **then**, there is an indistinguishability obfuscator $i\mathcal{O}^A$ for $\{\mathcal{AL}_\lambda\}$.

Furthermore, the following efficiency preservation holds.

- if $\mathcal{GS}$ has optimal efficiency or I/O-dependent complexity, $i\mathcal{O}^A$ has I/O-dependent complexity.
- If $\mathcal{GS}$ has space-dependent complexity, so does $i\mathcal{O}^A$.
- If $\mathcal{GS}$ and $i\mathcal{O}^C$ have linear-time-dependent complexity, so does $i\mathcal{O}^A$.

Proof of Proposition 5. This result relies on a notion from the recent work by Canetti, Lin, Tessaro and Vaikuntanathan [CLTV14] which they refer to as *probabilistic iO*. In [CLTV14], they show that assuming *sub-exponentially indistinguishable IO* for circuits and *sub-exponentially secure puncturable PRF*, the following natural way for obfuscating probabilistic circuits does achieve a limited notion of indistinguishability-based security. Below, we first recall their result and show how to apply it to obtain $\mathbf{iO}$ from garbling.

Probabilistic iO. Consider the following natural way of obfuscating a *probabilistic* circuit C .

$$\hat{C} \xleftarrow{\$} \text{piO}(1^\lambda, C) : k \xleftarrow{\$} \text{PRF-Gen}(1^{\lambda'}); C'(x) = C(x; F(k, x)); \hat{C} \xleftarrow{\$} i\mathcal{O}(1^{\lambda'}, C'), \text{ with } \lambda' = \text{poly}(\lambda, C.n)$$

The work of [CLTV14] showed that when the underlying $\mathbf{iO}$ and PRF are all sub-exponentially indistinguishable, the above algorithm piO ensures the indistinguishability of the obfuscation of two strongly indistinguishable circuits. More precisely, two circuits (C_1, C_2) with $n = C_1.n = C_2.n$ are strongly indistinguishable w.r.t. auxiliary input z if for every input $x \in \{0, 1\}^n$, outputs of $C_1(x)$ and $C_2(x)$ are $\text{negl}(\lambda)2^{-n}$ indistinguishable given z .

Lemma 1 (pIO for Circuits [CLTV14]). *Consider any sequence of probabilistic circuit pairs and auxiliary input $\{C_\lambda^1, C_\lambda^2, z_\lambda\}$, such that, for every non-uniform PPT $\mathcal{R}$, and every $\lambda \in \mathbb{N}$, $C^1 = C_\lambda^1$, $C^2 = C_\lambda^2$, $z = z_\lambda$,*

$$\forall x \in \{0, 1\}^n, \text{ where } n = C^1.n = C^2.n$$

$$\left| \Pr[y \xleftarrow{\$} C_1(x) : \mathcal{R}(C_1, C_2, x, y, z) = 1] - \Pr[y \xleftarrow{\$} C_2(x) : \mathcal{R}(C_1, C_2, x, y, z) = 1] \right| \leq \text{negl}(\lambda)2^{-n}$$

Assuming sub-exponentially indistinguishable $\mathbf{iO}$ for circuits $i\mathcal{O}^C$, and sub-exponentially indistinguishable OWF. The algorithm piO described above ensures that,

$$\left\{ C_1, C_2, \text{piO}(1^\lambda, C_1), z \right\}_\lambda \approx \left\{ C_1, C_2, \text{piO}(1^\lambda, C_2), z \right\}_\lambda$$

For completeness, we include a proof sketch of the lemma.

Proof Sketch: The proof of the lemma essentially relies on complexity leveling. To see the proof, first consider a simpler case, where the two circuits C_1 and C_2 have identical implementation on all but one input x^* , and the outputs on x^* , $C_1(x^*)$ and $C_2(x^*)$, are indistinguishable. In this case, we show that $\text{piO}(1^\lambda, C_1) \approx \text{piO}(1^\lambda, C_2)$. Recall that $\hat{C}_b \xleftarrow{\$} \text{piO}(1^\lambda, C_b)$ is the $\mathbf{iO}$ obfuscation of

program C'_b that evaluates C_b with pseudo-random coins generated using a hardwired PRF key k . Thus, it follows from the security of **iO** that $\hat{C}_b$ is indistinguishable to $i\mathcal{O}(C''_b)$ where C''_b has a punctured key $k(x^*)$ and $C_b(x^*; \mathbf{F}(k, x^*))$ hardwired in. Then it follows from the pseudo-randomness of puncturable PRF and the indistinguishability of $C_1(x^*)$ and $C_2(x^*)$ that $i\mathcal{O}(C''_0)$ and $i\mathcal{O}(C''_1)$ are indistinguishable. Therefore, by a hybrid argument, we have that $pi\mathcal{O}(1^\lambda, C_1) \approx pi\mathcal{O}(1^\lambda, C_2)$.

Now consider the case where C_1 and C_2 are completely different, but their output distributions are $\text{negl}(\lambda)2^{-n}$ -indistinguishable. To show that their **pIO** obfuscation are indistinguishable, we just need to consider an exponential, 2^n , number of hybrids, where in each hybrid, the implementation for one input x^* is changed from using C_1 to C_2 . By the same argument as before, neighboring hybrids have a distinguishing gap $O(\text{negl}(\lambda)2^{-n})$; thus by an exponential hybrid argument, the overall distinguishing gap is bounded by $\text{negl}(\lambda)$. This concludes the lemma.

Construction of **iO** for General Algorithms. Using the lemma from [CLTV14], we now prove Proposition 5.

Given $2^{-\lambda^\varepsilon}$ -indistinguishable **iO** $i\mathcal{O}^C$ and garbling scheme $\mathcal{GS}$, let $pi\mathcal{O}$ be the obfuscator for probabilistic circuits from $i\mathcal{O}^C$ as described above. Let **RE** be the following “randomized encoding” algorithm:

$$(\hat{AL}, \hat{x}) \xleftarrow{\$} \text{RE}(1^\lambda, AL, x), \text{ where } (\hat{AL}, \text{key}) \xleftarrow{\$} \text{Garb}(1^{\lambda'}, AL, x), \hat{x} = \text{Encode}(\text{key}, x), \lambda' = (\lambda + AL.n)^{1/\varepsilon}$$

Note that it is due to the monotonically increasing property of the algorithm class that we can invoke **Garb** with a bigger security parameter λ' for $AL \in \mathcal{AL}_\lambda$. Then our **iO** for the general algorithm class proceeds as,

$$\hat{AL} \xleftarrow{\$} i\mathcal{O}^A(1^\lambda, AL) \text{ where } \hat{AL} \xleftarrow{\$} pi\mathcal{O}(\lambda, \text{RE}(1^\lambda, AL, \cdot))$$

It follows from the correctness of $\mathcal{GS}$ and $i\mathcal{O}^C$ underlying $pi\mathcal{O}$ that $i\mathcal{O}^A$ has correctness.

Security. Next, we argue about the security of $i\mathcal{O}^A$ by contra-position. Fix a polynomial T , a *non-uniform* PPT samplable distribution $\mathcal{D}$ over the support $\{\mathcal{AL}_\lambda^T \times \mathcal{AL}_\lambda^T \times \{0, 1\}^{\text{poly}(\lambda)}\}$, such that, with overwhelming probability, $(AL_1, AL_2, z) \leftarrow \mathcal{D}(1^\lambda)$ satisfies that AL_1 and AL_2 are functionally equivalent and has matching parameters. Suppose that $i\mathcal{O}^A$ is insecure, that is, there is a non-uniform PPT $\mathcal{A}$ that distinguishes the following ensembles with inverse polynomial probability $1/p(\lambda)$.

$$\left\{ (AL_1, AL_2, z) \xleftarrow{\$} \mathcal{D}(1^\lambda) : (i\mathcal{O}^A(1^\lambda, AL_1), z) \right\}_\lambda$$

$$\left\{ (AL_1, AL_2, z) \xleftarrow{\$} \mathcal{D}(1^\lambda) : (i\mathcal{O}^A(1^\lambda, AL_2), z) \right\}_\lambda$$

Since $\mathcal{D}$ satisfies that with overwhelming probability, AL_1, AL_2 sampled from it have the same functionality and matching parameters, there exists one such sequence $\{(AL_{1,\lambda}, AL_{2,\lambda}, z_\lambda)\}$ w.r.t. which $\mathcal{A}$ distinguishes obfuscation of $AL_{1,\lambda}$ and $AL_{2,\lambda}$ given z_λ with probability $1/2p(\lambda)$. By construction of $i\mathcal{O}^A$, this implies that $\mathcal{A}$ distinguishes the following ensembles with probability $1/2p(\lambda)$.

$$\left\{ (pi\mathcal{O}(1^\lambda, \text{RE}(1^\lambda, AL_{1,\lambda}, \cdot)), z_\lambda) \right\}_\lambda \tag{7}$$

$$\left\{ (pi\mathcal{O}(1^\lambda, \text{RE}(1^\lambda, AL_{2,\lambda}, \cdot)), z_\lambda) \right\}_\lambda \tag{8}$$

We show this $\mathcal{A}$ contradicts with Lemma 1. It follows from $2^{-\lambda^\varepsilon}$ -indistinguishability of $\mathcal{GS}$ and the fact that algorithm RE invokes the garbling algorithm with security parameter $\lambda' = (\lambda + n)^{1/\varepsilon}$, for every $\lambda \in \mathbb{N}$, and $x_\lambda \in \{0, 1\}^{AL_{1,n}}$, the following ensembles are $\text{negl}(\lambda)2^{-n}$ -indistinguishable.

$$\left\{ C_1, C_2, x_\lambda, \text{RE}(1^\lambda, AL_{1,\lambda}, x_\lambda), z_\lambda \right\}_\lambda$$

$$\left\{ C_1, C_2, x_\lambda, \text{RE}(1^\lambda, AL_{2,\lambda}, x_\lambda), z_\lambda \right\}_\lambda$$

where $C_b = \text{RE}(1^\lambda, AL_{b,\lambda}, \cdot)$. Thus it follows from Lemma 1 that ensembles (7) and (8) are indistinguishable. This gives a contradiction with the fact that $\mathcal{A}$ distinguishes them and hence, the security of $i\mathcal{O}^A$ holds.

Efficiency. Finally, we analyze the efficiency of $i\mathcal{O}^A$. It is easy to see that $pi\mathcal{O}(1^\lambda, C)$ runs in polynomial time $p_{pIO}(\lambda', |C|)$ where the polynomial p_{pIO} depends on the running time of the underlying $\mathbf{iO}$ and PRF; moreover, if the underlying $\mathbf{iO}$ has linear-time-dependent complexity, p_{pIO} also depends quasi-linearly in $|C|$. Let $p_{\text{RE}}(\lambda', |AL|, n, m, S, T)$ be the running time of $\text{RE}(1^\lambda, AL, x)$ (depending on the efficiency of $\mathcal{GS}$, the polynomial p_{RE} depends on a subset of the parameters). Overall, the running time of $i\mathcal{O}^A(1^\lambda, AL)$ is

$$p_{pIO}(\lambda', p_{\text{RE}}(\lambda', |AL|, n, m, S, T)) \text{ where } \lambda' = \text{poly}(\lambda, n)$$

Therefore,

- If $\mathcal{GS}$ has optimal efficiency (that is, p_{RE} depends only on m) or I/O-dependent complexity (that is, p_{RE} does not depend on S, T), $i\mathcal{O}^A$ has I/O-dependent complexity.
- If $\mathcal{GS}$ has space-dependent complexity (that is, p_{RE} depend on T), so does $i\mathcal{O}^A$.
- If $\mathcal{GS}$ and the underlying $\mathbf{iO}$ has linear-time-dependent complexity (that is, p_{RE} depends quasi-linearly on T and so does p_{pIO} on $|C|$), so does $i\mathcal{O}^A$.

Finally, we note that combining the proposition with constructions of garbling schemes for TM and RAM in Section 3 and 4, we directly obtain $\mathbf{iO}$ for TM and RAM with space-dependent complexity.

Theorem 10. *Assume a sub-exponentially indistinguishable $\mathbf{iO}$ for circuits and sub-exponentially secure OWF. There is an indistinguishability obfuscator for TM and RAM with space-dependent complexity.*

Regarding Evaluation Efficiency. Evaluating the $\mathbf{iO}$ for TM and RAM obtained in Theorem 10 on input x , involves evaluating the obfuscated program on x once to obtain a garbled pair $(\hat{AL}, \hat{x})$, and then evaluate them. Therefore, overall, evaluation takes time $T_{AL}(x) \times \text{poly}(\lambda, |AL|, AL.S)$. When the space is large, the overhead on computation time is large. We can then improve the evaluation efficiency (at the price of losing succinctness of the garbled RAM) by combining proposition with the construction of garbling RAM by [LO13, GHL⁺14]. Since their garbled RAM has size and evaluation time quasi-linear in the time complexity (i.e., $\text{poly}(\lambda, |AL|, AL.m)T$), we can obtain the following:

Theorem 11. *Assume a sub-exponentially indistinguishable $\mathbf{iO}$ for circuits and sub-exponentially secure OWF. There is an indistinguishability obfuscator for RAM with input and output lengths bounded by a-priori fixed polynomials, and the indistinguishability obfuscator has linear-time dependent complexity.*

We note that when combining the theorem with the garbling scheme for RAM of [LO13, GHL⁺14] in a straightforward way, it actually yields only an **iO** for RAM with complexity (in terms of both size and evaluation time) polynomial in the time complexity T of the RAM machine being obfuscated. However, we can apply the same trick as described in [GHRW14] to improve the efficiency of the **iO** for RAM to depend only quasi-linearly on T . In the work of [GHRW14], they noticed that in the construction of [LO13, GHL⁺14], each bit in a garbled RAM and an input encoding can be generated using a small circuit of size $\tilde{O}(|R| + n + m) \times \text{poly}(\lambda, \log T)$, where R is the RAM machine under consideration, $n = R.n$, $m = R.m$ and $T = R.T$. Thus, to achieve linear-time-dependent complexity, we can simply modify our construction above as follows. Instead of directly obfuscating the whole randomized encryption algorithm RE , which has size $S = \tilde{O}(|R| + n + m) \times \text{poly}(\lambda, \log T) \times T$ and leads to a $\text{poly}(S)$ size obfuscated circuit, do the following: Decompose RE into a set of S small circuits $\{\text{RE}_i\}$ each of which computes the i^{th} bit in the randomized encoding, and then obfuscate each RE_i separately. Since each RE_i is small, the final obfuscation only depends on T quasi-linearly; more precisely, the complexity is $\text{poly}(\lambda, |R|, n, m, \log T) \times T$.

6.2 From iO to Garbling

We here show how to transform **iO** for a class C into garbling scheme for the same class with the same efficiency.

Proposition 6. *Let $\{\mathcal{AL}_\lambda\}$ be any class of deterministic algorithms. It holds that if there is an **iO** iO for $\{\mathcal{AL}_\lambda\}$, then there is a garbling scheme $\mathcal{GS}$ for this class. Furthermore, the following efficiency preservation holds.*

- *if iO has optimal efficiency or input- / I/O- dependent complexity, $\mathcal{GS}$ has I/O-dependent complexity.*
- *If iO has space- / linear-time- dependent complexity, so does $\mathcal{GS}$.*

We observe that combining the construction of **iO** for Turing machines with input-dependent complexity by [BCP14, ABG⁺13] with the theorem, we obtain a Garbling scheme for Turing machine with only I/O dependent complexity.

Corollary 1. *Assume the existence of **iO** for Turing machines with input-dependent complexity. There is a garbling scheme for Turing machines with I/O-dependent complexity.*

Proof of Proposition 6. The construction is quite straight-forward and illustrates the difference between obfuscation and garbling schemes. The key generation is simply the key generation algorithm for Lamport's one-time signature scheme [Lam79] with the tweak that instead of using a one-way function (as in Lamport's construction), we use a length doubling PRG—the public key for signing messages of length n consist of $2n$ images $\{c_i^b = f(r_i^b)\}_{i \in [n], b \in \{0,1\}}$ of a PRG f , and the secret key is the set of pre-images $\{r_i^b\}_{i \in [n], b \in \{0,1\}}$; both the public and secret keys are output as part of the garbling key. The encoding of an input x is a signature of x (i.e., $(r_1^{x_1}, \dots, r_n^{x_n})$, and the encoding of an algorithm AL is the obfuscation of a program $\Pi[AL]$ that on input n (presumed) pre-images $\vec{r}$ determines whether there exist an input x such that $f(r_i) = c_i^{x_i}$ and if so runs $AL(x)$; otherwise it simply outputs $\perp$; it is important that the computation performed to determine the input x takes a number of steps t_0 that is independent of the input.

To show that this construction is a secure garbling we need to exhibit a simulator that given just the output $y = AL(x)$ of the program AL on input x and the number of steps T^* taken by $AL(x)$ (as well as other parameters (n, m, S, T) of AL) can simulate the encoded input and program.

To simulate the encoded input, we simply output n random pre-images $\vec{r}$, and to simulate the the encoded program, we simply obfuscate the program $\tilde{\Pi}$ that on input $\vec{r}$ outputs y while taking T^* steps, and on any other input outputs $\perp$ while taking t_0 steps. (The bound parameters of $\tilde{\Pi}$ are set to (n, m, S, T) .)

To show indistinguishability of the simulation, we consider a hybrid experiment which proceeds just as the real one except that the key generation algorithm is modified so that (with overwhelming probability) there only exists a signature for the message x —we simply let $c_i^{1-x_i}$ be chosen as a uniform random string (instead of picking it in the image of f). By the security of the PRG this experiment is indistinguishable from the real one; furthermore (with overwhelming probability) the program being obfuscated in this experiment is functionally equivalent and has the same running-time as $\tilde{\Pi}$ for all inputs, and thus by security of the TM indistinguishability obfuscator this experiment is indistinguishable from the simulated one. $\square$

References

- [ABG⁺13] Prabhakaran Ananth, Dan Boneh, Sanjam Garg, Amit Sahai, and Mark Zhandry. Differing-inputs obfuscation and applications. 2013.
- [AIK04] Benny Applebaum, Yuval Ishai, and Eyal Kushilevitz. Cryptography in nc^0 . In *FOCS*, pages 166–175, 2004.
- [BCP14] Elette Boyle, Kai-Min Chung, and Rafael Pass. On extractability obfuscation. In *TCC*, pages 52–73, 2014.
- [BGI⁺01] Boaz Barak, Oded Goldreich, Russell Impagliazzo, Steven Rudich, Amit Sahai, Salil Vadhan, and Ke Yang. On the (im)possibility of obfuscating programs. In *Advances in Cryptology CRYPTO 2001*, pages 1–18. Springer, 2001.
- [BGI14] Elette Boyle, Shafi Goldwasser, and Ioana Ivan. Functional signatures and pseudorandom functions. In *PKC*, pages 501–519, 2014.
- [BGK⁺13] Boaz Barak, Sanjam Garg, Yael Tauman Kalai, Omer Paneth, and Amit Sahai. Protecting obfuscation against algebraic attacks. In *EuroCrypt’14*, 2013.
- [BMR90] Donald Beaver, Silvio Micali, and Phillip Rogaway. The round complexity of secure protocols (extended abstract). In *STOC*, pages 503–513, 1990.
- [BP13] Elette Boyle and Rafael Pass. Limits of extractability assumptions with distributional auxiliary input. 2013.
- [BR14] Zvika Brakerski and Guy N. Rothblum. Virtual black-box obfuscation for all circuits via generic graded encoding. In *TCC*, pages 1–25, 2014.
- [BSW12] Dan Boneh, Amit Sahai, and Brent Waters. Functional encryption: a new vision for public-key cryptography. *Commun. ACM*, 55(11):56–64, 2012.
- [BW13] Dan Boneh and Brent Waters. Constrained pseudorandom functions and their applications. In *ASIACRYPT (2)*, pages 280–300, 2013.

- [CIJ⁺13] Angelo De Caro, Vincenzo Iovino, Abhishek Jain, Adam O'Neill, Omer Paneth, and Giuseppe Persiano. On the achievability of simulation-based security for functional encryption. In *Advances in Cryptology - CRYPTO 2013 - 33rd Annual Cryptology Conference, Santa Barbara, CA, USA, August 18-22, 2013. Proceedings, Part II*, pages 519–535, 2013.
- [CLP13] Kai-Min Chung, Huijia Lin, and Rafael Pass. Constant-round concurrent zero knowledge from p-certificates. In *54th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2013, 26-29 October, 2013, Berkeley, CA, USA*, pages 50–59, 2013.
- [CLTV14] Ran Canetti, Huijia Lin, Stefano Tessaro, and Vinod Vaikuntanathan. Obfuscation of probabilistic functionalities and applications. Manuscript, 2014.
- [GGH13a] Sanjam Garg, Craig Gentry, and Shai Halevi. Candidate multilinear maps from ideal lattices. In *Advances in Cryptology–EUROCRYPT 2013*, pages 1–17. Springer, 2013.
- [GGH⁺13b] Sanjam Garg, Craig Gentry, Shai Halevi, Mariana Raykova, Amit Sahai, and Brent Waters. Candidate indistinguishability obfuscation and functional encryption for all circuits. *Proc. of FOCS 2013*, 2013.
- [GGM86] Oded Goldreich, Shafi Goldwasser, and Silvio Micali. How to construct random functions. *J. ACM*, 33(4):792–807, 1986.
- [GGP10] Rosario Gennaro, Craig Gentry, and Bryan Parno. Non-interactive verifiable computing: Outsourcing computation to untrusted workers. In *CRYPTO*, pages 465–482, 2010.
- [GHL⁺14] Craig Gentry, Shai Halevi, Steve Lu, Rafail Ostrovsky, Mariana Raykova, and Daniel Wichs. Garbled RAM revisited. In *Advances in Cryptology - EUROCRYPT 2014 - 33rd Annual International Conference on the Theory and Applications of Cryptographic Techniques, Copenhagen, Denmark, May 11-15, 2014. Proceedings*, pages 405–422, 2014.
- [GHRW14] Gentry, Halevi, Raykova, and Wichs. Outsourcing private ram computation. *Proc. of FOCS 2014*, 2014.
- [GJKS] Vipul Goyal, Abhishek Jain, Venkata Koppula, and Amit Sahai. Functional encryption for randomized functionalities.
- [GKP⁺13a] Shafi Goldwasser, Yael Tauman Kalai, Raluca A. Popa, Vinod Vaikuntanathan, and Nikolai Zeldovich. How to run turing machines on encrypted data. In *CRYPTO (2)*, pages 536–553, 2013.
- [GKP⁺13b] Shafi Goldwasser, Yael Tauman Kalai, Raluca A. Popa, Vinod Vaikuntanathan, and Nikolai Zeldovich. Reusable garbled circuits and succinct functional encryption. In *Symposium on Theory of Computing Conference, STOC'13, Palo Alto, CA, USA, June 1-4, 2013*, pages 555–564, 2013.
- [GLSW14] Craig Gentry, Allison Lewko, Amit Sahai, and Brent Waters. Indistinguishability obfuscation from the multilinear subgroup elimination assumption. Cryptology ePrint Archive, Report 2014/309, 2014.

- [GMW87] Oded Goldreich, Silvio Micali, and Avi Wigderson. How to play any mental game or a completeness theorem for protocols with honest majority. In *STOC*, pages 218–229, 1987.
- [IK02] Yuval Ishai and Eyal Kushilevitz. Perfect constant-round secure computation via perfect randomizing polynomials. In *ICALP*, pages 244–256, 2002.
- [KPTZ13] Aggelos Kiayias, Stavros Papadopoulos, Nikos Triandopoulos, and Thomas Zacharias. Delegatable pseudorandom functions and applications. In *CCS*, pages 669–684, 2013.
- [Lam79] Leslie Lamport. Constructing digital signatures from a one-way function. SRI Technical Report, 1979.
- [LO13] Steve Lu and Rafail Ostrovsky. How to garble RAM programs. In *Advances in Cryptology - EUROCRYPT 2013, 32nd Annual International Conference on the Theory and Applications of Cryptographic Techniques, Athens, Greece, May 26-30, 2013. Proceedings*, pages 719–734, 2013.
- [Mic00] Silvio Micali. Computationally sound proofs. *SIAM J. Comput.*, 30(4):1253–1298, 2000.
- [O’N10] Adam O’Neill. Definitional issues in functional encryption. Cryptology ePrint Archive, Report 2010/556, 2010. <http://eprint.iacr.org/>.
- [PST14] Rafael Pass, Karn Seth, and Sidharth Telang. Indistinguishability obfuscation from semantically-secure multilinear encodings. In *CRYPTO’14*, 2014.
- [SW05] Amit Sahai and Brent Waters. Fuzzy identity-based encryption. In *EUROCRYPT*, pages 457–473, 2005.
- [SW14] Amit Sahai and Brent Waters. How to use indistinguishability obfuscation: Deniable encryption, and more. *Proc. of STOC 2014*, 2014.
- [Yao86] Andrew Chi-Chih Yao. How to generate and exchange secrets (extended abstract). In *FOCS*, pages 162–167, 1986.

Constant-Round Non-Malleable Commitments from Any One-Way Function

Huijia Lin*

Rafael Pass†

September 1, 2011‡

Abstract

We show *unconditionally* that the existence of commitment schemes implies the existence of *constant-round* non-malleable commitments; earlier protocols required additional assumptions such as collision resistant hash functions or subexponential one-way functions.

Our protocol also satisfies the stronger notions of concurrent non-malleability and robustness. As a corollary, we establish that constant-round non-malleable zero-knowledge arguments for **NP** can be based on one-way functions and constant-round secure multi-party computation can be based on enhanced trapdoor permutations; also here, earlier protocols additionally required either collision-resistant hash functions or subexponential one-way functions.

*Cornell University, E-Mail: huijia@cs.cornell.edu.

†Cornell University, E-Mail: rafael@cs.cornell.edu. Pass is supported in part by a Alfred P. Sloan Fellowship, Microsoft New Faculty Fellowship, NSF CAREER Award CCF-0746990, AFOSR YIP Award FA9550-10-1-0093, and DARPA and AFRL under contract GG11413-137380. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the Defense Advanced Research Projects Agency or the US government.

‡A preliminary version of this paper appeared in STOC 2011.

1 Introduction

Commitment schemes are one of the most fundamental cryptographic building blocks. Often described as the “digital” analogue of sealed envelopes, commitment schemes enable a *sender* to commit itself to a value while keeping it secret from the *receiver*. This property is called *hiding*. Furthermore, the commitment is *binding*, and thus in a later stage when the commitment is opened, it is guaranteed that the “opening” can yield only a single value determined in the committing stage. Their applications range from coin flipping [?] to the secure computation of any efficiently computable function [?, ?]. In light of their importance, commitment schemes have received a considerable amount of attention. This has resulted in a fairly comprehensive understanding of the hardness assumptions under which they can be realized; in particular, by the results of Naor [?] and Håstad et al [?], the existence of one-way functions implies the existence of two-round commitments.

For many applications, however, the most basic security guarantees of commitments are not sufficient. For instance, the basic definition of commitments does not rule out an attack where an adversary, upon seeing a commitment to a specific value v , is able to commit to a related value (say, $v - 1$), even though it does not know the actual value of v . This kind of attack might have devastating consequences if the underlying application relies on the *independence* of committed values (e.g., consider a case in which the commitment scheme is used for securely implementing a contract bidding mechanism). Indeed, for the general task of secure multi-party computation [?], such independence is crucial. The state of affairs is even worsened by the fact that many of the known commitment schemes are actually susceptible to this kind of attack. In order to address the above concerns, Dolev, Dwork and Naor (DDN) introduced the concept of *non-malleable commitments* [?]. Loosely speaking, a commitment scheme is said to be non-malleable if it is infeasible for an adversary to “maul” a commitment to a value v into a commitment to a related value $\tilde{v}$.

More precisely, we consider a *man-in-the-middle* (MIM) attacker that participates in two concurrent execution of a commitment scheme $\langle C, R \rangle$; in the “left” execution it interacts with an honest committer (running C); in the “right” execution it interacts with an honest receiver (running R). Additionally, we assume that the players have n -bit identities (where n is polynomially related to the security parameter), and that the commitment protocol depends only on the identity of the committer; we sometimes refer to this as the identity of the interaction. Intuitively, $\langle C, R \rangle$ being non-malleable means that if the identity of the right interaction is different than the identity of the left interaction (i.e., A does not use the same identity as the left committer), the value A commits to on the right does not depend on the value it receives a commitment to on the left; this is formalized by requiring that for any two values v_1, v_2 , the values A commits to after receiving left commitments to v_1 or v_2 are indistinguishable.

The first non-malleable commitment protocol was constructed by Dolev, Dwork and Naor [?] in 1991. The security of their protocol relies on the minimal assumption of one-way functions and requires $\Omega(\log n)$ rounds of interaction, where $n \in N$ is the length of party identities. Non-malleable commitments have since been extensively studied in the literature; the main question has been to determine the number of communication rounds needed for non-malleable commitments. Let us briefly survey some of this literature.

1.1 The State of the Art of Non-malleable Commitments

As mentioned, the original work by DDN assumes only one-way functions, and considers the “plain” model of execution; that is, there is no trusted infrastructure. DiCrenenzo, Ishai and Ostrovsky [?] and follow-up works in e.g., [?, ?, ?, ?, ?] showed how to improve the round-complexity of the

DDN construction when assuming the existence of some trusted infrastructure (e.g. a common random string); in such models *non-interactive* (i.e., single message) non-malleable commitments based on only one-way function are known [?]. The first improvement to the round-complexity of the DDN construction without any trusted infrastructure came more than a decade later. Following the ground-breaking work by Barak on *non-black-box simulation* [?], in 2002, Barak [?] presented a constant-round protocol for non-malleable commitments; the security of this protocol however relies on the existence of trapdoor permutations and hash functions that are collision-resistant against circuits of sub-exponential size. A few years later, Pass and Rosen [?] (relying on a technique from [?]), showed that collision resistant hash functions secure against polynomially-sized circuits are sufficient to obtain a constant-round protocol. Next, Pandey, Pass and Vaikuntanathan [?] provided a construction of a non-interactive non-malleable commitment based on a new hardness assumption with a strong non-malleability flavour; in contrast to the earlier constant-round constructions, their protocol has a black-box proof of security.

But, despite two decades of research, there have been no improvements over the original DDN construction, when only assuming the existence of *one-way functions*, leaving open the following question.

Does there exist a sub-logarithmic non-malleable commitment scheme, assuming only one-way functions?

1.2 Settling the Round-complexity of Non-Malleable Commitments

In this work, we settle the round-complexity of non-malleable commitments: we present a constant-round protocol in the “plain” model that is based on the assumption of one-way functions, and has a black-box proof of security. Since the existence of commitment schemes already implies the existence of one-way functions (cl. [?]), we have:

Theorem 1. *Assume the existence of a commitment scheme. Then, there exists a constant-round non-malleable commitment scheme with a black-box proof of security.*

Concurrent Non-malleability: As mentioned, the original notion of non-malleability considers an MIM attacker participating in a single execution on the left and a single execution on the right. Already the original DDN paper suggested that a stronger notion of non-malleability—*concurrent non-malleability*—where the MIM may participate in an unbounded number of executions on both the left and the right, is desirable. Pass and Rosen [?] provided the first construction of a concurrently non-malleable commitment scheme; their scheme only has a constant number of rounds but relies on the existence of claw-free permutations (and non-black-box techniques). Subsequently, Lin, Pass and Venkatasubramanian [?] provided an $O(n)$ -round construction based on one-way functions. As we show, our protocol is also concurrently non-malleable.

Robust Non-malleability: In this work we also introduce a new notion of non-malleability: *robust non-malleability*. Roughly speaking, whereas traditional non-malleability considers a scenario where a MIM participates in the *same* commitment protocol on the left and the right, *r*-robustness considers a notion of non-malleability for commitments where the MIM attacker participates in an arbitrary *r*-round protocol on the left, and the commitment protocol on the right. Robustness is useful when using non-malleable commitments as subprotocols within larger protocols (see Section 1.3). As we show, for any constant *r*, our protocol can be made *r*-robust while still remaining constant-round.

Thus summarizing the above discussion, we have:

Theorem 2. *Assume the existence of a commitment scheme. Then, for any constant r , there exists a constant-round commitment scheme that is r -robust concurrently non-malleable with a black-box proof of security.*

1.3 Applications to Secure Computation

As mentioned, “independence” of inputs is crucial for secure multi-party computation protocols. Indeed, there has been a tight interplay between work on the round-complexity of multi-party computation (MPC) and work on non-malleable commitments.

Goldreich, Micali and Wigderson’s [?] original work on secure multi-party computation showed a $\Omega(m)$ -round multi-party computation protocol based on the existence of enhanced trapdoor permutations (TDPs), where m is the number of players in the execution; implicit in their work is a $O(n)$ -round non-malleable commitment for the special case of so-called “synchronizing” adversaries that have identities of length $\log n$. Subsequent works improved the round-complexity by making stronger assumptions. Katz, Ostrovsky, and Smith [?], following the work by Chor and Rabin [?], obtained a $O(\log m)$ -round MPC protocol assuming TDPs and dense-crypto systems by relying on the non-malleable commitments from [?]. By additionally assuming the existence of hash-function collision-resistant against circuits of sub-exponential size (and non-black-box techniques), they also obtained a $O(1)$ -round MPC protocol by instead relying on the non-malleable commitment from [?]. More recently, Pass [?], showed the existence of a $O(1)$ -rounds MPC protocol assuming only TDPs and (standard) collision resistant hash functions (but still using non-black box techniques); this technique in turned was used in the non-malleable commitment of [?].

The implicit connection between the round-complexity of non-malleable commitments and secure multi-party was formalized by Lin, Pass and Venkitasubramanian in [?]: they show that *the existence of k -round 4-robust non-malleable commitments and the existence of TDPs implies the existence of $O(k)$ -round secure multi-party computation.*

Combining the result of [?] with Theorem 2, we get that secure multi-party computations can be performed in a constant number of round based on only TDPs.

Theorem 3. *Assume the existence of enhanced trapdoor permutations. Then there exists a constant-round protocol for secure multi-party computation.*

1.4 Applications to Non-malleable ZK

Non-malleable zero-knowledge [?] consider the execution of zero-knowledge protocols in the presence of a MIM attacker. Roughly speaking, a zero-knowledge protocol is non-malleable if the MIM attacker can only provide convincing right-interaction proofs of statements that it could have proved without participating in the left interaction. The recent result of [?] shows that *the existence of k -round 4-robust non-malleable commitments implies the existence of $O(k)$ -round non-malleable zero-knowledge arguments for NP*. By combining their results with Theorem 2 we directly have that the existence of one-way functions implies the existence a constant-round zero-knowledge argument for NP.

Theorem 4. *Assume the existence of one-way functions. Then there exists a constant-round non-malleable zero-knowledge argument for NP with a black-box proof of security.*

1.5 A New Technique: “Message-scheduling in the head”

The main idea underlying all non-malleable commitment schemes is to “encode” the identity of the committer into the protocol. At the very least, this ensure that unless the attacker copies the

identity of the left committer, the attacker cannot simply forward messages between the left and the right executions. But we also need to ensure that the attacker cannot in a clever way maul the messages it receives on the left so they become useful on the right. For instance, in the original DDN construction, the identity is encoded into the scheduling of messages in the protocol; on a very high-level (and oversimplifying), the idea is to ensure that at some point in the execution, the MIM must “speak” while only receiving “useless” messages. The problem with this approach is that it requires a high round-complexity.

We will revisit the DDN approach. The main idea behind our scheme is to perform the message scheduling “in the head”. A bit more precisely, our protocol follows the “simulation-soundness” paradigm of Sahai [?], first used in the context of CCA-secure encryption, and next used by Pass and Rosen [?] in the context of non-malleable commitments; that is, the main component of our construction is a method for enabling us to “simulate” the left interaction, while ensure that the right interaction remains “sounds”. Towards this, we embedd a “trapdoor” into the protocol which depends on the identity of the interaction; proving simulation-soundness then essentially amounts to showing that there exists a way to recover the trapdoor for the left intraction, while ensuring that the adversary does not recover the trapdoor for the right interaction (as long as the right interaction has a different identity than the left interaction).

The idea is to have a protocol where the trapdoor can be recovered by “rewinding” some specific messages in the protocol—called “slots”—in *a specific order which depends on the identity of the interaction*. Furthermore, the protocol should have the property that if this specific rewinding order is not the rewinding order actually used, then a trapdoor cannot be recovered. So, if we rewind the left interaction according to the rewinding order corresponding to the identity of the left interaction, this will still not enable the adversary to recover the trapdoor corresponding to the right interaction (unless the identity of the right interaction is the same as the identity of the left interaction). In our particular instantiation of this idea, the trapdoor will be a “signature-chain” (i.e., a signature on a signature on a signature, etc.) of length n (i.e., the identity length) using different keys; the choice of the keys in the signature chain are determined by the identity of the interaction. Next, the protocol will have a “slot” for each of the keys where the receiver is willing to sign a *single* message for the committer using the key corresponding to the slot. The key point is that the simulator is able to rewind the slots in an appropriate order to recover a signature-chain corresponing to the identity of the left interaction; but the rewindings will still not enable the adversary to recover a signature-chain corresponding to any other identity.

1.6 Historical Notes

This paper is a combined version of [?] and [?]. In [?], we first showed the existence of a $O(1)^{\log^* n}$ -round protocol that is based on the existence of one-way functions and uses a black-box proof of security. On a high-level, the main technique of [?] was a method for *amplifying non-malleability*: that is, we presented a method for transforming a non-malleable commitment scheme that handles identities of length t into one that handles identities of length $O(2^t)$. The $O(1)^{\log^* n}$ -round protocol was finally obtained starting off with the protocol of DDN for constant length identities and next iteratively amplifying it. The notion of robust non-malleability was also first defined in [?] and was an intergral part of the amplification procedure: in fact, our amplification procedure could only be applied to robust non-malleable commitments.

In [?], we additionally pointed out that amplification procedure also yield a natural route towards constant-round non-malleable commitments: it suffices to come with a constant-round protocol that handles identities of length $\log^{(k)} n = \log \log \dots \log n$, where k is a constant; any such protocol can be amplified to a full-fledged non-malleable commitment while still remaining constant round. Subsequent work by Pass and Wee [?] obtained a constant-round protocol based on sub-

exponentially hard one-way functions (again using a black-box proof of security), by following this paradigm: subexponential one-way functions were used to construct a constant-round non-malleable commitment for “small” identities, and the protocol can then be amplified into a full-fledged one. An elegant work by Wee [?] later simplified and improved the amplification procedure of [?], leading to a protocol using $O(\log^* n)$ -rounds, based on one-way functions. Finally, independently of [?], a beautiful work by Goyal also obtains a constant-round non-malleable commitment based on one-way functions; the construction of [?] also follows the above amplification paradigm by Goyal instead of a constant-round robust non-malleable commitment protocol for small identities based on one-way functions.

The construction from [?] is direct: we no longer require amplification; instead we directly construct a full-fledged robust non-malleable protocol. Nevertheless, some of the ideas used in the amplification procedure are helpful when analyzing our protocol.

1.7 Outline

In Section 2, we provide some preliminaries. In Section 3, we provide an overview of our protocol construction and its security proof. In Section 4 we provide some formalizations and results about “signature-chains”. Our protocol (which relies on the notion of a signature chain) is presented in Section 5. We provide the proof of (stand-alone) non-malleability in Section 6; in Section 7 and 8, we demonstrate that our protocol is also concurrent non-malleable, and can be made r -robust for any constant r .

2 Preliminaries

Let N denote the set of all positive integers. For any integer $n \in N$, let $[n]$ denote the set $\{1, 2, \dots, n\}$. We denote by $\{0, 1\}^n$ the set of binary strings of length n , and $\{0, 1, 2\}^n$ the set of trinary strings of length n . Given a binary (or trinary) string ψ of length n , we denote by $[\psi]_1^i$ the prefix of ψ of length i . We denote by $\mathcal{PPT}$ probabilistic polynomial time Turing machines. We assume familiarity with interactive Turing machines, denoted ITM, interactive protocols, and computational indistinguishability; the formal definitions of interactive protocols and computational indistinguishability are provided in Appendix A. Given a pair of ITMs, A and B , we denote by $\langle A(x), B(y) \rangle(z)$ the random variable representing the (local) output of B , on common input z and private input y , when interacting with A with private input x , when the random tape of each machine is uniformly and independently chosen.

2.1 Signature Schemes

We focus on *fixed-length signature schemes* $\Pi = (\text{Gen}, \text{Sign}, \text{Ver})$, that is, the signing algorithm Sign on input 1^n , a public key pk and a message $m \in \{0, 1\}^*$, always outputs a signature of length n . We refer the reader to [?] for a formal definition. Such signature schemes can be constructed relying on universal one-way hash functions [?], which in turn can be based on any one-way function [?]. Below, a signature scheme always refers to a fixed-length signature scheme.

2.2 Commitment Schemes

Commitment schemes are used to enable a party, known as the *sender*, to commit itself to a value while keeping it secret from the *receiver* (this property is called **hiding**). Furthermore, the commitment is **binding**, and thus in a later stage when the commitment is opened, it is guaranteed that the “opening” can yield only a single value determined in the committing phase. In this work,

we consider commitment schemes that are **statistically-binding**, namely while the hiding property only holds against computationally bounded (non-uniform) adversaries, the binding property is required to hold against unbounded adversaries. We refer the reader to [?] for a formal definition.

Two-round (i.e., a single message from the receiver followed by a single message from the committer) commitment schemes are known to exist based on the minimal assumption of one-way functions [?, ?]. In the sequel of the paper, a commitment scheme always refers to a statistically-binding commitment.

Tag-based Commitment Scheme. Following [?, ?], we consider *tag-based commitment schemes* where, in addition to the security parameter, the committer and the receiver also receive a “tag”—a.k.a. the identity— id as common input.

2.3 Concurrent Non-Malleable Commitments

We recall the definition of concurrent non-malleability from [?]. For convenience, we use a slightly different presentation (based on indistinguishability rather than simulation); equivalence follows using a standard argument (c.f. [?, ?]). Let $\langle C, R \rangle$ be a tag-based commitment scheme, and let $n \in N$ be a security parameter. Consider a man-in-the-middle adversary A (as shown in figure 1) that, on inputs n and z (where z is received as an auxiliary input), participates in m left and right interactions simultaneously. In the left interactions the man-in-the-middle adversary A interacts with C , receiving commitments to values $v_1, \dots, v_m$, using identities of length n , $\text{id}_1, \dots, \text{id}_m \in \{0, 1\}^n$, of its choice. In the right interactions A interacts with R attempting to commit to a sequence of related values $\tilde{v}_1, \dots, \tilde{v}_m$, again using identities of length n $\tilde{\text{id}}_1, \dots, \tilde{\text{id}}_m$ of its choice. If any of the right commitments are invalid, or undefined, its value is set to $\perp$. For any i such that $\tilde{\text{id}}_i = \text{id}_j$ for some j , set $\tilde{v}_i = \perp$ —i.e., any commitment where the adversary uses the same identity as one of the left interactions is considered invalid. Let $\text{mim}_{\langle C, R \rangle}^A(v_1, \dots, v_m, z)$ denote a random variable that describes the values $\tilde{v}_1, \dots, \tilde{v}_m$ and the view of A , in the above experiment.

Definition 1. A commitment scheme $\langle C, R \rangle$ is said to be *concurrent non-malleable* (with respect to itself) if for every polynomial $p(\cdot)$, and every *PPT* man-in-the-middle adversary A that participates in at most $m = p(n)$ concurrent executions, the following ensembles are computationally indistinguishable.

$$\left\{ \text{mim}_{\langle C, R \rangle}^A(v_1, \dots, v_m, z) \right\}_{n \in N, v_1, \dots, v_m \in \{0, 1\}^n, v'_1, \dots, v'_m \in \{0, 1\}^n, z \in \{0, 1\}^*}$$

$$\left\{ \text{mim}_{\langle C, R \rangle}^A(v'_1, \dots, v'_m, z) \right\}_{n \in N, v_1, \dots, v_m \in \{0, 1\}^n, v'_1, \dots, v'_m \in \{0, 1\}^n, z \in \{0, 1\}^*}$$

We also consider relaxed notions of concurrent non-malleability: one-one, one-many, and many-one secure non-malleable commitments (See Figure 2 below.) In a one-one (a.k.a., a stand-alone secure) non-malleable commitment, we consider only adversaries A that participate in one left and one right interaction; in one-many, A participates in one left and many right, and in many-one, A participates in many left and one right.

As shown in [?], any protocol that is one-many non-malleable is also concurrent non-malleable.

Proposition 1 ([?]). *Let $\langle C, R \rangle$ be a one-many concurrent non-malleable commitment. Then, $\langle C, R \rangle$ is also a concurrent non-malleable commitment.*

2.4 Robustness: Non-Malleability w.r.t. k -round Protocols

The concept of non-malleability is traditionally only considered in a setting where a man-in-the-middle adversary is participating in two (or more) executions of the *same* protocol. We here consider a new notion of non-malleability with respect to arbitrary k -round protocols.

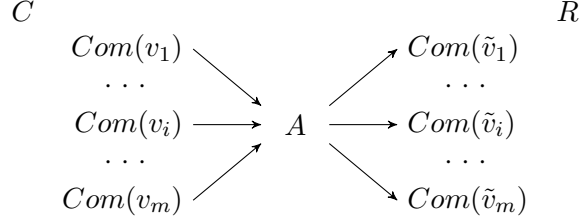


Figure 1: A concurrent man-in-the-middle adversary.

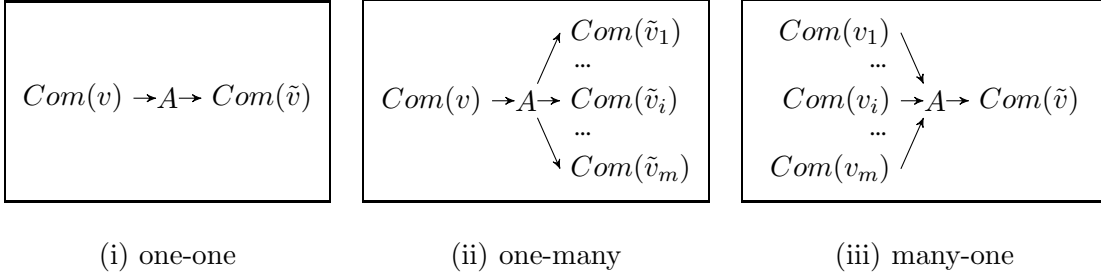


Figure 2: Restricted man-in-the-middle adversaries.

Consider a one-many man-in-the-middle adversary A (as shown in figure 3) that participates in one left interaction—communicating with a machine B —and many right interactions—acting as a committer using the commitment scheme $\langle C, R \rangle$. As in the standard definition of non-malleability, A can adaptively choose the identities in the right interactions. We denote by $\text{mim}_{\langle C, R \rangle}^{B, A}(y, z)$ the random variable consisting of the view of $A(z)$ in a man-in-the-middle execution when communicating with $B(y)$ on the left and honest receivers on the right, combined with the values $A(z)$ commits to on the right. Intuitively, we say that $\langle C, R \rangle$ is one-many non-malleable w.r.t B if $\text{mim}_{\langle C, R \rangle}^{B, A}(y_1, z)$ and $\text{mim}_{\langle C, R \rangle}^{B, A}(y_2, z)$ are indistinguishable, whenever interactions with $B(y_1)$ and $B(y_2)$ cannot be distinguished.

Definition 2. Let $\langle C, R \rangle$ be a commitment scheme, and B a PPT ITM. We say the commitment scheme $\langle C, R \rangle$ is one-many non-malleable w.r.t. B , if for every two sequences $\{y_n^1\}_{n \in N}$ and $\{y_n^2\}_{n \in N}$, $y_n^1, y_n^2 \in \{0, 1\}^n$, such that, for all PPT ITM $\tilde{A}$, it holds that

$$\left\{ \langle B(y_n^1), \tilde{A}(z) \rangle(1^n) \right\}_{n \in N, z \in \{0, 1\}^*} \approx \left\{ \langle B(y_n^2), \tilde{A}(z) \rangle(1^n) \right\}_{n \in N, z \in \{0, 1\}^*}$$

then it also holds that, for every PPT one-many man-in-the-middle adversary A ,

$$\left\{ \text{mim}_{\langle C, R \rangle}^{B, A}(y_n^1, z) \right\}_{n \in N, z \in \{0, 1\}^*} \approx \left\{ \text{mim}_{\langle C, R \rangle}^{B, A}(y_n^2, z) \right\}_{n \in N, z \in \{0, 1\}^*}$$

We say that $\langle C, R \rangle$ is one-many k -robust if $\langle C, R \rangle$ is one-many non-malleable w.r.t. any machine B that interacts with the man-in-the-middle adversary in k rounds.

3 Proof Overview

To explain the main ideas behind our construction, we here focus on outlining the construction of a constant-round non-malleable commitment scheme that is secure for *synchronizing* and *non-aborting* adversaries; we next comment on how to deal with general adversaries. An adversary is

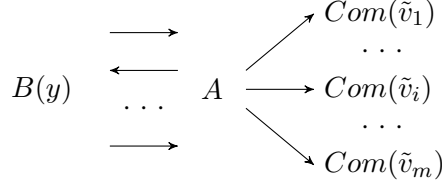


Figure 3: A concurrent man-in-the-middle adversary with respect to protocol B on input y .

said to be synchronizing if it “aligns” the left and the right executions; that is, whenever it receives message i on the left, it directly sends message i on the right, and vice versa. An adversary is said to be non-aborting if it never sends any invalid messages in the left interaction (where it is acting as a receiver); it might still send invalid messages on the right.

As mentioned in the introduction, the idea is to have a protocol with an “identity-based trapdoor” embedded into it. The trapdoor will be a “signature-chain” using a sequence of keys that are determined by the identity of the protocol. More precisely, we say that $(\sigma_0, \sigma_1, \dots, \sigma_n)$ is a *plain signature chain*¹ with respect to the signature scheme Π , the verification keys vk_0, vk_1 and the pattern $\psi \in \{0, 1\}^n$ if $\sigma_0 = 0$ and for all $0 \leq i < n$, σ_{i+1} is a signature on the message (i, σ_i) with respect to the key $vk_{\psi_{i+1}}$. For convenience of notation, for the remainder of this section we fix a particular signature scheme Π ; all signatures we use are with respect to this particular scheme.

The following simple claim regarding signature chains will be useful. Consider a “signature game” where an adversary A gets access to two randomly chosen verification keys vk_0, vk_1 and additionally has access to signature oracles with respect to vk_0 and vk_1 ; let φ denote the “access pattern” of the adversary to the signature oracle (that is, if the i ’th oracle call is to the signature oracle w.r.t. vk_b , then $\varphi_i = b$). The claim now is that, with overwhelming probability, if in the signature game, A manages to output a plain signature chain with respect to vk_0, vk_1 and pattern ψ , then ψ is a substring of φ .

The protocol for committing to a string v with identity id proceeds as follows:

- **Slot 1:** The receiver R generates a key-pair (sk_0, vk_0) for the signature scheme Π , and sends vk_0 to the committer C . C next send a random message r_0 to R who signs r_0 and then returns the signature to C .
- **Slot 2:** R generates another key-pair (sk_1, vk_1) and sends vk_0 to the committer C . As in Slot 1, C next send a random message r_1 to R who signs r_1 and then returns the signature to C .
- **Commit phase:** C commits to v using a standard statistically binding commitment.
- **Proof phase:** C gives R a “special-purpose”² witness indistinguishable argument of knowledge of the fact that it either knows the value committed to in the commit phase, or that it knows a plain signature chain with respect to vk_0, vk_1 and id .

We now turn to argue that this protocol is non malleable with respect to non-aborting and synchronizing adversaries. For simplicity, we here focus only on one-one (i.e., stand-alone) non-malleability (but the same proof actually also works for concurrent non-malleability). Consider a

¹We use the name “plain signature chain” (instead of just “signature chain”), since the actual signature chains we will use in the final construction will be a bit more complicated.

²We will shortly explain what makes this proof special.

man-in-the-middle adversary A that uses identity id on the left and identity $\tilde{\text{id}} \neq \text{id}$ on the right, and receives a commitment to the value v on the left. We will argue that no matter what the value of v is, the value it commits to on the right will be indistinguishable. Towards this goal, consider a hybrid experiment where the left interaction is simulated by acting honestly in Slot 1 and 2, next committing to 0, and finally using a “fake-witness”—namely a signature chain—in the proof phase; the simulator obtains this fake witness by simply rewinding Slot 1 and 2 (that is, to rewinding slot b , we restore the state of A after vk_b has been sent, and send a new message to be signed) in the appropriate order to obtain a signature chain with respect to id (note that since A is non-aborting, each time the simulator asks it to sign a message, it does). To show the above claim, we now argue that no matter what the value of v is, the value A commits to on the right in the real execution (when receiving a commitment to v), is indistinguishable from the value it commits to on the right when the left interaction instead is simulated.

The key-point of the proof is the claim that even in the simulation, A cannot use a fake-witness in the right interaction. This follows from the fact that since A is synchronizing, when we rewind Slot 1 and 2 on the left, the same slots are rewound on the right *in exactly the same order*. Thus, by the signature-game claim, if A manages to get a signature chain it must be a subset of the pattern 01id (the reason we need to append 01 is that A gets two signatures in the honest emulation of Slot 1 and 2, already before we start the rewindings). So, if we appropriately restrict the identity set (for instance, by requiring that all identities start with 10) then the only valid identity that is a substring of 01id is id , and thus $\tilde{\text{id}} = \text{id}$, which is a contradiction.

To argue that the value committed to on the right does not change when we move from the real interaction to the simulation, consider an intermediary hybrid where we only change the witness used in the proof phase (but keep the value committed to in the commit phase to v). Note that since the adversary is synchronizing, the proof phase of the left interaction appears completely after the commitment (in Stage 2) in the right interaction. Therefore, the right value does not change at all when switching the the witness used in the proof phase on the left.

Finally, we simply have to argue that the value on the right does not change once we change the value committed to in the commit phase on the left. By the hiding property of the left commitment, the view of the adversary does not change when the left committed value switches. But since the value committed to on the right cannot be efficiently recovered, this does not directly imply that the committed value also is indistinguishable. To resolve this problem, we rely on the argument of knowledge property of the proof phase: A witness on the right can be extracted efficiently from the proof phase. Since the witness used in the right interaction cannot be a fake witness (by the key-claim above), it must be the value committed to in the commit phase, so indistinguishability of the committed value follows from the hiding property of the the left commitment.

Dealing with aborting adversaries: When considering aborting adversaries, we run into two obstacles:

- The adversary might notice that the simulator is feeding it signature chains to sign (instead of random messages) and thus decide to abort the left execution. We handle this by adapting the definition of a signature chain: instead of requiring the chain to be “a signature on a signature on a signature... etc”, we require a signature-chain to be a signature on “a *commitment* of a signature on a commitment of a signature... etc”. And next, in the protocol, we let C send commitments to 0 instead of random strings. To be able to establish an analog of the above signature-game claim, we additionally require C to give a zero-knowledge argument of knowledge of the value it committed to before R agrees to sign it.
- Another problem is that A might abort the left execution with some probability p . This means that we might have to rewind the left execution many times (roughly $1/p$ times) before getting

the signature we are looking for. As a consequence, the "access pattern" on the right will be a substring of $0\text{id}_1^*\text{id}_2^*\dots\text{id}_n^*$. To get around this problem, we add an additional slot (and a corresponding signature key). Next, we require that the signature-chain corresponding to the identity id to be with respect to the pattern $2\text{id}_12\text{id}_22\text{id}_3\dots2\text{id}_n$.

Dealing with non-synchronizing adversaries: As is usually the case, synchronizing adversaries are the "hardest" to deal with. To prove security against non-synchronizing adversaries, we follow basically the same argument: First, if A is not synchronizing there exists some slot that is never rewound and so if the identity of the right interaction contains at least two 0's and two 1's, we can still establish the above key-claim. Next, to argue that the committed value on the right does not change, we consider again the intermediary hybrid above. However, when the adversary is not synchronizing, it may choose to interleave messages in the proof phase of the left interaction and the commitment of the right interaction, and thus the right committed value may change when the witness on the left changes. To overcome this problem, we again rely on the argument of knowledge property of the proof phase to extract a witness from the right interaction. Since the witness cannot be a signature-chain (by the key-claim), it must be the committed value; then the indistinguishability of the committed value follows from the witness-indistinguishability of the left proof phase. However, one problem is that extraction on the right may rewind the left proof phase and thus break the witness indistinguishability property. One way of resolving this problem would be to (in analogy with [?]) have the proof phase be statistically witness indistinguishable; but this requires additional assumptions (to keep it constant-round). Instead, we here introduce a different technique to overcome this problem: we let the proof phase consist of *multiple sequentially ordered* witness indistinguishable special-sound proofs.³ This allows us to change the witness in each of the proofs, one by one, while ensuring that the witness on the right can be extracted from some other proof, without rewinding the left proof where the witness currently is being changed.

4 Signature Chains and Games

Let $\Pi = (\text{Gen}, \text{Sign}, \text{Ver})$ be a fixed-length signature scheme, and com a statistically-binding commitment scheme. For simplicity of notation, we keep these schemes fixed, and provide our definitions and protocols with respect to those particular schemes. Furthermore, for simplicity of exposition, we assume that com that is non-interactive; however, all of our definitions and protocols can be easily modified to work with any two-round statistically-binding commitment schemes; see Remark 5 for further details.

We now turn to formally defining the notion of a *signature-chain* and then proceed to defining *signature-games*.

Definition 3 (Signature-Chain). *Let $\ell \in \mathbb{N}$, $\psi \in \{0, 1, 2\}^\ell$ and $vk_0, vk_1, vk_2 \in \{0, 1\}^*$ be three verification keys for the signature scheme Π . We say that a triplet $\delta = (\bar{\sigma}, \bar{c}, \bar{r})$ is a **signature-chain** w.r.t. keys vk_0, vk_1, vk_2 and pattern ψ , if $\bar{\sigma}$, $\bar{c}$, and $\bar{r}$ are vectors of length ℓ satisfying the following properties.*

- *For all $i \in [\ell]$, $\bar{\sigma}_i$ is valid signature of the message $\bar{c}_i$ under key vk_{ψ_i} , i.e., $\text{Ver}(vk_{\psi_i}, \bar{\sigma}_i, \bar{c}_i) = 1$.*
- *For all $1 < i \leq \ell$, $\bar{c}_i$ is a commitment to the tuple $(i-1, \bar{\sigma}_{i-1})$ using com and randomness $\bar{r}_i$; and $\bar{c}_1$ is a commitment to 0^m using com and randomness $\bar{r}_1$, where $m = \log \ell + n$.*

³This method was originally used by us in the amplification procedure of [?]; this "trick" is also a central component enabling the works of [?, ?].

We say that a signature-chain $\delta = (\bar{\sigma}, \bar{c}, \bar{r})$ has length ℓ if $|\bar{\sigma}| = \ell$.

We proceed to define a signature-game $SG^{A,\ell}(n, z)$, where A on input $1^n, z$ interacts with a *Challenger* in the following three stages:

Stage 1: the Challenger samples three pairs of signing and verification keys at random, $(sk_b, vk_b) \leftarrow \text{Gen}(1^n)$, where $b \in \{0, 1, 2\}$, and sends A the verification keys, vk_0, vk_1 , and vk_2 .

Stage 2: A interacts with the Challenger in a sequence of iterations for as long as it wishes. Iteration i proceeds as follows:

- A sends the Challenger a tuple (φ_i, c) , where $\varphi_i \in \{0, 1, 2\}$, followed by a 5-round ZKAOK proof of the statement that c is a valid commitment of com .
- if the proof is convncing, the Challenger signs the commitment c using the signing key s_{φ_i} and returns the signature to A ; otherwise, it aborts the iteration (without giving back a signature).

Stage 3: Finally, A outputs the tuple (δ, ψ) .

We call the sequence $\varphi = \varphi_1, \varphi_2, \dots$ of signing request, the “access pattern” of A . We say that the output of A is *well-formed* if δ is a length $l(n)$ signature-chain with respect to vk_0, vk_1, vk_2 and ψ . Finally, we say that A *wins* if its output is well-formed at ψ is not a substring of its access pattern φ (and *loses* otherwise).

Lemma 1. *For every PPT adversary A and every polynomial ℓ , there exists a negligible function μ , such that for every $n \in N, z \in \{0, 1\}^*$, the probability that A wins in $SG^{A,\ell}(n, z)$ is at most $\mu(n)$.*

Proof. Consider any adversary A , polynomial ℓ , $n \in N$, and $z \in \{0, 1\}^*$. Without loss of generality, we can assume that A always outputs tuples $(\delta = (\bar{\sigma}, \bar{c}, \bar{r}), \psi)$ such that $|\bar{\sigma}| = |\bar{c}| = |\bar{r}| = \psi = l(n)$ (since whenever it doesn't it loses). For each $i \in [l(n)]$, define the random variable $\mathcal{I}_i$ to be the index of the *first* iteration (in Stage 2 of the game $SG^{A,\ell}(n, z)$) in which A queries the Challenger for a signature of the commitment $\bar{c}_i$ under key v_{ψ_i} ; if A never queries the Challenger for a signature of $\bar{c}_i$, $\mathcal{I}_i$ is set to $\perp$.

Note that if the output of A is well-formed, it contains a signature-chain $\delta = (\bar{\sigma}, \bar{c}, \bar{r})$ w.r.t. pattern ψ , such that for every i , $\bar{\sigma}_i$ is a valid signature of $\bar{c}_i$ under key v_{ψ_i} . It thus follows from the unforgibility of the signature scheme that, except with negligible probability, for each i , A must have queried $\bar{c}_i$ for a signature of v_{ψ_i} in some iteration. We thus have the following claim.

Claim 1. *For every PPT adversary A and polynomial ℓ , there exists a negligible function μ_1 , such that for all $n \in N, z \in \{0, 1\}^*$, the probability that the output of A in $SG^{A,\ell}(n, z)$ is of A is well-formed and there exists an $i \in [l(n)]$ such that $\mathcal{I}_i = \perp$, is smaller than $\mu_1(n)$.*

We also have the following claim.

Claim 2. *For every PPT adversary A and polynomial ℓ , there exists a negligible function μ_2 , such that, for all $n \in N, z \in \{0, 1\}^*$, the probability that the output of A in $SG^{A,\ell}(n, z)$ is well-formed and there exists an $i \in [l(n) - 1]$ such that $\mathcal{I}_i \neq \perp, \mathcal{I}_{i+1} \neq \perp$ and $\mathcal{I}_i \geq \mathcal{I}_{i+1}$, is smaller than $\mu_2(n)$.*

Before proceeding to the proof of Claim 2, we let us first prove Lemma 1 using Claim 1 and 2. It follows from the two claims that, except with negligible probability, *either* the output of A is not well-formed, *or* the output is well-formed and for all i , $\mathcal{I}_i \neq \perp$ and $\mathcal{I}_i < \mathcal{I}_{i+1}$. In the former case, the adversary loses the game. In the latter case, as $\mathcal{I}_i \neq \perp$ for all i , A must have asked for a signature using key v_{ψ_i} in the $\mathcal{I}_i^{\text{th}}$ iteration, which means $\varphi_{\mathcal{I}_i} = \psi_i$. Furthermore, as $\mathcal{I}_i < \mathcal{I}_{i+1}$ for all i , it follows that ψ is a substring of φ . Therefore, A loses in this case as well. Thus, except with negligible probability, A loses.

Proof of Claim 2. First notice that it follows from the (statistical) binding property of **com**, that except with negligible probability⁴, if the output $(\delta = (\bar{\sigma}, \bar{c}, \bar{r}), \psi)$ of A is well-formed, then for all i , $\bar{c}_i \neq \bar{c}_{i+1}$, since $\bar{c}_i, \bar{c}_{i+1}$ are respectively commitments to tuples of the form $(i, \cdot)$ and $(i+1, \cdot)$. It follows that, except with negligible probability, if the output of A is well-formed, there doesn't exist some i such that $\mathcal{I}_i, \mathcal{I}_{i+1} \neq \perp$ but $\mathcal{I}_i = \mathcal{I}_{i+1}$. Thus, it suffices to bound the probability that the output of A is well formed and there exists some i such that $\mathcal{I}_i, \mathcal{I}_{i+1} \neq \perp$ and $\mathcal{I}_i > \mathcal{I}_{i+1}$.

Towards this, assume for contradiction that there exists an adversary A and a polynomial ℓ , such that there exists a function $i : N \rightarrow N$ and a polynomial p , such that for infinitely many $n \in N, z \in \{0, 1\}^*$, the probability that the output of A in the game $\text{SG}^{A, \ell}(n, z)$ is well-formed, $\mathcal{I}_i, \mathcal{I}_{i+1} \neq \perp$, and $\mathcal{I}_i > \mathcal{I}_{i+1}$ for $i = i(n)$, is at least $1/p(n)$. We can construct a machine B that violate the unforgibility of the signature scheme Π .

B , on input $1^n, z$ and a randomly generated verification key vk , has access to the signing oracle corresponding to vk , and tries to forge a signature (of vk) as follows: it internally emulates an execution of the signature game $\text{SG}^{A, \ell}(n, z)$ with A honestly, with the following exceptions:

- In Stage 1, it picks an index $t \in \{0, 1, 2\}$ at random and forwards the verification key vk to the adversary as the t^{th} verification key.
- In Stage 2, whenever A requests a signature of a message m under key vk , it obtains such a signature from the signing oracle and forwards it to A .

Furthermore, it guesses that $\mathcal{I}_i = u$ and $\mathcal{I}_{i+1} = k$, for random $u > k$. Then, in the k^{th} iteration (in Stage 2 of $\text{SG}^{A, \ell}(n, z)$), after receiving a request from A to sign the commitment c , it extracts out the value (j, σ^*) committed to in c from the ZKAOK that A provides following the signing request. Later, in the u^{th} iteration, when A submits a query c^* to the Challenger, it checks whether σ^* is a valid signature of c^* under key vk . If so, it halts and outputs the message-signature pair (c^*, σ^*) ; otherwise, it halts and outputs fail.

By construction, B emulates the view of A in the signature game $\text{SG}^{A, \ell}(n, z)$ perfectly before it halts. Therefore, by our hypothesis, with probability at least $1/p(n)$, in emulation by B , A would query for the first time the commitments $\bar{c}_i$ and $\bar{c}_{i+1}$ in iterations $\mathcal{I}_i$ and $\mathcal{I}_{i+1}$ respectively, such that $\mathcal{I}_{i+1} < \mathcal{I}_i$ and $\bar{c}_{i+1}$ is a commitment to a tuple $(i+1, \bar{\sigma}_{i+1})$, where $\bar{\sigma}_{i+1}$ is a signature of $\bar{c}_i$ under the verification key v_{ψ_i} . Let $M(n)$ be the maximum number of iterations in the game; M is polynomially bounded since the running-time of A is. With probability at least $\frac{1}{q(n)} = \frac{1}{3M(n)^2 p(n)}$, it holds that (1) the above event occurs in the emulation by B and (2) B correctly guesses the values of $\mathcal{I}_i, \mathcal{I}_{i+1}$ and v_{ψ_i} . In this case, except with negligible probability, the committed value σ^* that B extracts out from the ZKAOK following $c = \bar{c}_{i+1}$ contains a valid signature of $\bar{c}_i$, which is queried for the first time in the u^{th} iteration for a signature using key vk . Hence B will output a valid message-signature pair $(\bar{c}_i, \sigma^*)$ for vk , without querying the signing oracle $\bar{c}_i$ (since once the query $\bar{c}_i$ is submitted for the first time in iteration u , B halts immediately and outputs the pair); this violates the unforgibility of the signature scheme Π . $\square$

$\square$

5 The Protocol

Let $\Pi = (\text{Gen}, \text{Sign}, \text{Ver})$ be the fixed-length signature scheme, and **com** the non-interactive statistically-binding commitment scheme considered in the last section. To simplify the presentation of the proof, we assume that both Π and **com** can be “broken”—i.e., signatures can be

⁴Since we assume that **com** is non-interactive, we actually have perfect binding, but given that we want an analysis that works also for two-round commitments, we here directly consider the more general case of statistical binding.

generated for *any* message, and the value committed to can be recovered for *any* commitment—in time $2^{n/2}$ where n is the security parameter; this is without loss of generality since we can always appropriately “scale-down” the security parameter in Π and com (and make sure that com commits to values “bit-by-bit”). To further simplify the presentation, we provide the construction of a non-malleable commitment $\langle C, R \rangle$ that works assuming player identities are ℓ -bit binary strings that contains at least two 0-bits and two 1-bits; any such scheme can trivially be turned into one that works for arbitrary identities (by simply appending two 0’s and two 1’s to the identity).

To commit to a value v , the Committer and the Receiver of $\langle C, R \rangle$, on common input a security parameter 1^n (in unary) and an identity $\text{id} \in D^\ell$, proceed in the following three stages:

Stage 1: The receiver interacts with the Committer in three iterations, where iteration $i \in \{0, 1, 2\}$ proceeds in the following steps:

1. The Receiver generates a pair of signing and verification keys, $(s_i, v_i) \leftarrow \text{Gen}(1^n)$, of the signature scheme Π , and sends the verification key v_i .
2. The Committer commits to 0^m , where $m = \log \ell + n$, using com . Let c_i be the commitment sent to the Receiver.
3. The Committer proves that c_i is a valid com commitment using a 5-round ZKAOK protocol.
4. The Receiver signs the commitment c_i using the signing key s_i , and sends the generated signature θ_i to the Committer.

Stage 2: The Committer commits to the value v using com . Let c' be the commitment generated.

Stage 3: The Committer proves that

- *either* c' is a valid com commitment,
- *or* there exists a signature-chain δ w.r.t. v_0, v_1, v_2 and pattern $\text{pattern}(\text{id})$, where the function $\text{pattern} : \{0, 1\}^* \rightarrow \{0, 1, 2\}^*$ maps a (binary) identity id of length ℓ to a trinary string of length 2ℓ as follows:

$$\text{pattern}(\text{id}) = 2, \text{id}_1, 2, \dots, \text{id}_i, 2, \dots, \text{id}_\ell$$

This statement is proved using $k + 5$ sequential invocations of a 4-round $\mathcal{WZ}$ special sound proof system, where k is the number of messages in Stage 1 of the protocol; we additionally require that the length of the “challenge” in each special-sound proof is n .

We refer to the last three steps of an iteration in Stage 1 as a *slot*, which *opens* when the Committer send the com commitment to 0^m , and *closes* when the Receiver returns a signature to the commitment. We call the slot in iteration i , the i ’th slot.

It is easy to see that the protocol $\langle C, R \rangle$ consists of a constant number of messages. Furthermore, it follows using standard techniques that $\langle C, R \rangle$ is a valid commitment scheme.

Proposition 2. $\langle C, R \rangle$ is a commitment scheme.

Proof. We show that the $\langle C, R \rangle$ scheme satisfies the binding and hiding properties.

Binding: The binding property follows directly from the statistically binding property of com used in Stage 2.

Hiding: The hiding property essentially follows from the hiding property of com and the fact that Stage 3 of the protocol is $\mathcal{WT}$ (since $\mathcal{WT}$ proofs are closed under concurrent composition [?]). For completeness, we provide the proof. We show that any adversary R^* that violates the hiding property of $\langle C, R \rangle$ can be used to violate the hiding property of com . More precisely, given any adversary R^* , such that, for infinitely many $n \in N$, and $v_1, v_2 \in \{0, 1\}^n$, R^* distinguishes commitments to v_1 and v_2 made using $\langle C, R \rangle$, we construct a machine R' that distinguishes commitments to v_1 and v_2 made using com . Note that the execution of a commitment of $\langle C, R \rangle$ to v_1 proceeds identically as that of a commitment to v_2 before the Stage 2 commitment of com is sent. Then by our hypothesis, there must exist a partial joint view ρ of the committer and R^* that determines the execution of the commitment before Stage 2, such that, conditioned on ρ occurring, R^* distinguishes commitments to v_1 and v_2 . Let δ be a valid signature-chain corresponding to the transcript of Stage 1 in ρ . R' on auxiliary input ρ and δ proceeds as follows: it internally incorporates R^* , and feed R^* its part of view in ρ ; it then forwards the external commitment made using com to R^* in Stage 2; in Stage 3, it gives $\mathcal{WT}$ proofs using δ as a “fake witness”. Finally, it outputs whatever R^* outputs. From the $\mathcal{WT}$ property of Stage 3, it follows that R' distinguishes the commitment made using com , if R^* distinguishes the commitment made using $\langle C, R \rangle$ conditioned on ρ occurring. □

Both the definition of signature-games and our non-malleable commitment protocols makes use of a *non-interactive* statistically-binding commitment scheme com . Both can be easily modified to work also with any two-round statistically binding commitment schemes $\overline{\text{com}}$. In both cases, we the first message r of a commitment of $\overline{\text{com}}$ is sent at the beginning of the execution, and then the rest of the execution proceeds just as if $\overline{\text{com}}$ had been non-interactive. (Additionally, in the last stage of the protocol $\langle C, R \rangle$, the sender proves that either the Stage 2 message is the second message of a valid $\overline{\text{com}}$ commitment with first message r , or it knows a signature-chain $\delta = (\bar{\sigma}, \bar{c}, \bar{r})$, such that, δ is well-formed, except that, for all i , $\bar{c}_i$ is the second message of a $\overline{\text{com}}$ commitment to $\bar{\sigma}_{i-1}$, generated in responding to the first message r using randomness $\bar{r}_i$). Exactly the same proof as in Section 4 and Section 6 still go through using these modified construction, since commitments of $\overline{\text{com}}$ are hiding, no matter what the first message is, and even if the first message is reused.

6 Proof of Non-malleability

In this section, we show that $\langle C, R \rangle$ is stand-alone non-malleable. In Sections 8 and 7, we extend the proof to show that $\langle C, R \rangle$ is also robust and concurrent non-malleable.

Theorem 5. $\langle C, R \rangle$ is (one-one) non-malleable.

Proof. The goal is to show that for every one-one man-in-the-middle adversary A that participates in one left and one right execution, the following ensembles are indistinguishable:

$$\left\{ \text{mim}_{\langle C, R \rangle}^A(v_1, z) \right\}_{n \in N, v_1, v_2 \in \{0, 1\}^n, z \in \{0, 1\}^*}$$

$$\left\{ \text{mim}_{\langle C, R \rangle}^A(v_2, z) \right\}_{n \in N, v_1, v_2 \in \{0, 1\}^n, z \in \{0, 1\}^*}$$

Towards this, we define a series of hybrid experiments $H_0, \dots, H_{k+6}$. In each of these experiments, we show that the view of A , combined with the value that A commits to on the right, are indistinguishable. Let $\text{hyb}_i(v, z)$ denote the random variable describing the view of $A(z)$, combined with

the value it commits to in the right interaction in hybrid H_i (as usual, the committed value is replaced with $\perp$ if the right interaction fails or if A has copied the identity of the left interaction).

Hybrid H_0 : In H_0 we first perfectly emulate a real execution of $\text{mim}_{(C,R)}^A(v, z)$ —we call this the *Main Execution*—and next, if A successfully completed Stage 1 in the Main Execution, we try extract a “fake-witnesses” (i.e., a signature-chain) for the left interaction. More precisely, let $\text{id}_l, v_0, v_1, v_2$, respectively be the identity and the verification keys of the left interaction in the Main Execution, and let $\psi = \text{pattern}(\text{id}_l)$; the *Extraction Procedure* now proceeds in $|\psi| = 2\ell$ iterations described below.

Iteration 1: If A successfully completes Stage 1 of the left interaction in the Main Execution, it must have provided three valid signatures $\theta_0, \theta_1, \theta_2$ of commitments to 0^m , where $m = \log \ell + n$, under keys v_0, v_1, v_2 respectively. Since a signature-chain with pattern ψ starts off with a signature $\bar{\sigma}_1$ of a commitment to 0^m under key $v_{\psi_1} = v_2$, the procedure simply sets $\bar{\sigma}_1 = \theta_2$, $\bar{c}_1$ to be the transcript of the commitment to 0^m generated in iteration 2 (in Stage 1) of the left interaction, and $\bar{r}_1$ to be the randomness used in the commitment.

Iteration $i + 1$: Assume that at the end of the i^{th} iteration, for $i \in [2\ell - 1]$, the procedure has obtained a signature-chain δ_i of length i w.r.t. (keys v_0, v_1, v_2 and) pattern $[\psi]_1^i$, containing signatures $\bar{\sigma}_1, \dots, \bar{\sigma}_i$. Then, in iteration $i + 1$, we obtain a signature-chain δ_{i+1} of length $i + 1$, w.r.t. pattern $[\psi]_1^{i+1}$ by rewinding the appropriate slot in Stage 1 of the left interaction. More precisely, the procedure repeatedly rewinds A from where the slot ψ_{i+1} opens on the left in the Main Execution, and commits to the tuple $(i, \bar{\sigma}_i)$ (instead of 0^m) in the rewindings, until this left-slot closes successfully (i.e., A returns a valid signature on the commitment under key $v_{\psi_{i+1}}$). In each of these rewindings, the right executions are emulated using fresh randomness; in particular, this means that whenever a rewinding goes beyond the point when a verification key is sent in the right interaction, in each such rewinding a fresh verification key is picked. Then the extraction procedure simply sets $\bar{\sigma}_{i+1}$ to be this signature, and again sets $\bar{c}_{i+1}$ and $\bar{r}_{i+1}$ to be the commitment and randomness used.

If the extraction procedure takes more than $2^{n/2}$ steps, it is “cut-off”; in this case, a signature chain can be recovered in time $\text{poly}(2^{n/2})$ by our assumption on the signature scheme II. The extraction procedure thus always terminates and always recovers a valid signature chain for the left interaction.

Since the view of A in the Main Execution in H_0 is perfectly emulated as in $\text{mim}_{(C,R)}^A(v, z)$, we trivially have that the view and value A commits to in H_0 is identically distributed to that in the real execution.

Claim 3. *For every PPT adversary A , it holds that:*

$$\left\{ \text{mim}_{(C,R)}^A(v, z) \right\}_{n \in N, v \in \{0,1\}^n, z \in \{0,1\}^*} = \left\{ \text{hyb}_0(v, z) \right\}_{n \in N, v \in \{0,1\}^n, z \in \{0,1\}^*}$$

Hybrid H_1 to H_{k+5} : In hybrids H_1 to H_{k+5} , we change the witness used in the $k + 5$ *WISSP* proofs in Stage 3 of the left interaction. More specifically, experiment H_i proceeds identically to H_{i-1} , except that in the first i proofs in Stage 3 of the left interaction, we prove that there exists a signature-chain w.r.t. v_0, v_1, v_2 and pattern $\text{pattern}(\text{id}_l)$, by using the extracted signature-chain δ as a “fake-witness”. We show that the view and value committed to on the right interaction in H_{i-1} and H_i are indistinguishable.

Proposition 3. *For every PPT adversary A , and every function $i : N \rightarrow N$, it holds that:*

$$\left\{ \text{hyb}_{i(n)-1}(v, z) \right\}_{n \in N, v \in \{0,1\}^n, z \in \{0,1\}^*} \approx \left\{ \text{hyb}_{i(n)}(v, z) \right\}_{n \in N, v \in \{0,1\}^n, z \in \{0,1\}^*}$$

Towards this, we reduce the indistinguishability of $\{\text{hyb}_{i(n)-1}(v, z)\}$ and $\{\text{hyb}_{i(n)}(v, z)\}$ to the witness indistinguishability of the Stage 3. More specifically, consider some adversary A , a function i , and a polynomial p , such that (for infinitely many $n \in N$, inputs $v \in \{0,1\}^n$ and $z \in \{0,1\}^*$), $\text{hyb}_{i(n)-1}(v, z)$ and $\text{hyb}_{i(n)}(v, z)$ are distinguishable with probability $1/p(n)$. We show that there exists a PPT machine B that can violate the $\mathcal{WI}$ property of the $\mathcal{WISSP}$ protocol $\langle P, V \rangle$ used in Stage 3 of the protocol.

Description of B

Input: B receives a security parameter 1^n and v and z as auxiliary input.

Procedure: B externally interacts with a prover P of the $\mathcal{WISSP}$ protocol $\langle P, V \rangle$, receiving a proof of a statment x using witness w_0 or w_1 , where x , w_0 and w_1 are chosen by B . Internally, it proceeds in the following three phases:

Simulation Phase: B internally emulates an execution of the experiment $\text{hyb}_i(v, z)$ with A , with the exception that messages in the i^{th} left-proof of the Main Execution are forwarded externally to P . More precisely, at the beginning of the i^{th} left-proof, B sends the external prover P the statement x of the i^{th} proof, together with the “real witness” $w_0 = (v, r)$ (the decommitment of the Stage 2 commitment of the left interaction) and the “fake witness” $w_1 = \delta$ (the signature-chain of the left interaction extracted from A); B next forwards the proof of x generated by P (using either w_0 or w_1) to A as the i^{th} left-proof. Let Δ be the simulated view of A in the Main Execution.

Rewinding Phase: If the right interaction is successful and has a different identity from the left interaction in Δ , B extracts the value committed to in this interaction as follow:

- Find the first $\mathcal{WISSP}$ proof $(\alpha_1, \alpha_2, \beta, \gamma)$ in Δ , such that, during its the execution, no messages belonging to Stage 1 or the i^{th} proof of the left interaction are exchanged. (Such a $\mathcal{WISSP}$ proof must exist since there are $k + 5$ $\mathcal{WISSP}$ proofs, whereas only $k + 4$ messages in Stage 1 and the i^{th} proof of the left interaction.)
- Rewinds the proof by sending new random challenges β' until a second transcript $(\alpha_1, \alpha_2, \beta', \gamma')$ is obtained.
In the rewindings, emulate the left and right interaction for A in identically the same way as in the Main Execution, except that, whenever A expects a new message in Stage 1 or the i^{th} proof of the left interaction, cancel the execution and start a new rewinding again.
- If $\beta_\rho \neq \beta'_\rho$, extract witness w from $(\alpha_1, \alpha_2, \beta, \gamma)$ and $(\alpha_1, \alpha_2, \beta', \gamma')$. Otherwise halt and output fail_1 .
- If $w = (v, r)$ is valid decommitment for the right interaction, then set $\hat{v} = v$. Otherwise halt and output fail_2 .

Output Phase: If the right interaction that is not convincing or the identity of the right interaction is the same as the left interaction, set $\hat{v} = \perp$. Output $\hat{v}$ and Δ .

Figure 4: The construction of B

On a high-level, the machine B , on common input 1^n and auxiliary input v, z , externally interacts with an honest prover P and receives a left-interaction Stage 3 proof, generated using either the real witness w_0 —the decommitment of the Stage 2 commitment in the left interaction—or the fake witness w_1 —a signature chain for the left interaction. Internally, B emulates an execution of either hyb^{i-1} or hyb^i with A (depending on the witness used in the external proof), except that, messages in the i^{th} proof in Stage 3 of the left interactions are forwarded externally. Furthermore, if the right interaction is successful and has a different identity from the left, B attempts to extract the value committed to on the right by repeatedly rewinding the WISSP proofs in Stage 3 of the right interaction by sending new challenge messages in this proof. Since the i^{th} left-proof is forwarded externally, the rewinding has to be done in a manner that does not “affect” the i^{th} left-proof. Roughly speaking, this is possible since there are more WISSP proofs in Stage 3 of the right interaction, than the number of messages in the i^{th} left-proof. Therefore, in the right interaction, there exist some WISSP proofs that does not interleave with any messages in the i^{th} left-proof, and B can use rewindings to extract a witness *without rewinding* the left-proof. Our actual rewinding strategy also avoids rewinding Stage 1 of the left interaction, so that the fake-witness δ of the left interaction remains a valid signature chain also in the rewindings, and thus can be reused to simulate the left interaction also in the rewindings. This is again possible since there are more right-proofs than the number of messages in Stage 1 and the i^{th} proof in the left interaction. To slightly simplify the analysis, we additionally “cut-off” the rewindings if B takes more than $2^{n/2}$ steps and simply recover the value committed to in time $\text{poly}(2^{n/2})$; recall that this is possible due to our assumption on com .

If during the rewindings, B sends the same challenge message twice, it aborts outputting fail_1 . Additionally, if the witness extracted from the right interaction is not a valid decommitment (it could also be a fake-witness), B aborts outputting fail_2 . Otherwise, B outputs the emulated view of A , together with the value committed to in the right interaction.,

See Figure 4 for a formal description of B . Below, in Lemma 2, we show that the running-time of machine B is “bounded”, in the sense that the probability that B runs for super-polynomial time is negligible.

Lemma 2. *There exists a polynomial function T , such that for every polynomial function q , every $b \in \{0, 1\}$, every sufficiently large $n \in N$, and inputs $v \in \{0, 1\}^n$ and $z \in \{0, 1\}^*$, the probability that machine B runs for more than $q(n)T(n)$ steps in an execution of the experiment $\text{STA}_b(\langle P, V \rangle, B, v, z)$ is smaller than $1/q(n)$.*

Roughly speaking, the Lemma is proven by first bounding the running-time of a “hypothetical procedure” which perform all the same rewindings, but otherwise acts honestly (i.e., always commits to 0^m in Stage 1, and always uses the honest witness in Stage 3); it follows using a simple “ $p \times 1/p$ ” argument (similar to those in [?]) that the expected running-time of this procedure is polynomial. Next we show that, with high probability, the running-time of the actual procedure is not too far off. We note that due to reasons similar to those in [?] we are not able to bound the expected running-time of B . Additionally, it seems unclear if the methods of [?] could be applicable to obtains a simulation with an expected polynomial running-time. Fortunately, in our application, since we do not actually per se care about the running time of the simulation (but only care about breaking some specific security property, namely witness indistinguishability) our weaker bound suffices.

A formal proof of Lemma 2 can be found in Section 6.1.

The following lemma is the core of our analysis.

Lemma 3. *The following holds.*

$$\begin{aligned} \{\text{STA}_0(\langle P, V \rangle, B, v, z)\}_{n \in N, v \in \{0,1\}^n, z \in \{0,1\}^*} &\approx \{\text{hyb}^{i-1}(v, z)\}_{n \in N, v \in \{0,1\}^n, z \in \{0,1\}^*} \\ \{\text{STA}_1(\langle P, V \rangle, B, v, z)\}_{n \in N, v \in \{0,1\}^n, z \in \{0,1\}^*} &\approx \{\text{hyb}^i(v, z)\}_{n \in N, v \in \{0,1\}^n, z \in \{0,1\}^*} \end{aligned}$$

Before proceeding to the proof of 3, let us see how Lemma 3 and 2 together violate the $\mathcal{WI}$ property of the Stage 3 proofs. Recall that by our assumption, $\text{hyb}^i(v, z)$ and $\text{hyb}^{i-1}(v, z)$ can be distinguished with probability $1/p(n)$; by Lemma 3, $\text{STA}_0(\langle P, V \rangle, B, v, z)$ and $\text{STA}_1(\langle P, V \rangle, B, v, z)$ can thus be distinguished with probability at least, say, $3/4p(n)$. By Lemma 2, the probability that B runs for more than, say, $4p(n)T(n)$ steps in either experiment is at most $1/4p(n)$. Therefore, by the union bound, the outputs of B (in STA_0 and STA_1) are still distinguishable with probability at least $1/4p(n)$, even if we cut-off the execution of B after $4p(n)T(n)$ steps (and output $\perp$ if B fails to complete), which is a contradiction.

Let us now turn to proving Lemma 3.

Proof. (of Lemma 3) By construction, B perfectly emulates the view of A in $\text{hyb}^{i-1}(v, z)$ when receiving an external proof generated using the real witness w_0 , and that in $\text{hyb}^i(v, z)$ when receiving a proof generated using the fake witness w_1 . Therefore, to show Lemma 3, it suffices to show that B (almost) always extracts a valid decommitment for the right interaction if it is successful and has a different identity from the left interaction (recall that by statistical binding of $\langle C, R \rangle$, the committed value is unique with overwhelming probability). In other words, showing Lemma 3 amounts to showing that the probability that B outputs fail_1 or fail_2 is negligible.

Claim 4. *There exists a negligible function μ , such that for every $b \in \{0, 1\}$, every sufficiently large $n \in N$, and inputs $v \in \{0, 1\}^n$ and $z \in \{0, 1\}^*$, the probability that B outputs fail_1 in $\text{STA}_b(\langle P, V \rangle, B, v, z)$ is smaller than $\mu(n)$.*

Proof. Recall that B outputs fail_1 only if in some rewinding it picks the same challenge β' as the challenge β used in the same proof in the Main Execution. Since the number of rewindings by B is bounded by $2^{n/2}$ and the length of each challenge is n , by the union bound, the probability that this happens is negligible. $\square$

Claim 5. *There exists a negligible function μ , such that, for every $b \in \{0, 1\}$, every sufficiently large $n \in N$, and inputs $v \in \{0, 1\}^n$ and $z \in \{0, 1\}^*$, the probability that B outputs fail_2 in $\text{STA}_b(\langle P, V \rangle, B, v, z)$ is smaller than $\mu(n)$.*

Proof. Assume for contradiction that there exists a polynomial $g(n)$, such that, with probability $1/g(n)$, B extracts an invalid decommitment from the right interaction. Towards reaching a contradiction, we consider another machine B' , which proceeds identically to B except that it cuts-off the execution after $g(n)T(n)$ steps (and outputs $\perp$ in this case). It follows from Lemma 2 that the probability that B runs for more than $g(n)T(n)$ steps is at most $1/2g(n)$. Therefore, the probability that B' extracts out an invalid decommitment from the right interaction k is at least $1/2g(n)$. Furthermore, by the special-soundness property of the right-proofs, if the witness is not a valid decommitment, it must be a signature-chain δ w.r.t. the right-interaction keys v'_0, v'_1, v'_2 and pattern $\text{pattern}(\text{id}_r)$. Consider the following two possible adversarial schedulings w.r.t. the left and the k^{th} right interactions in the Main Execution:

Scheduling 1: A “aligns” the slots in the left and right interactions *one by one*: a right-slot is said to be *aligned* with a left-slot if (1) its corresponding verification key is sent before the left-slot opens, and (2) its opening message (i.e., the commitment from A) is sent after the left-slot opens; see Figure 5 (i).

Scheduling 2: A does not align the slots in the left and right interactions; see Figure 5 (ii). This means that there exists some right-interaction slot that is not aligned with *any* left-interaction slot.

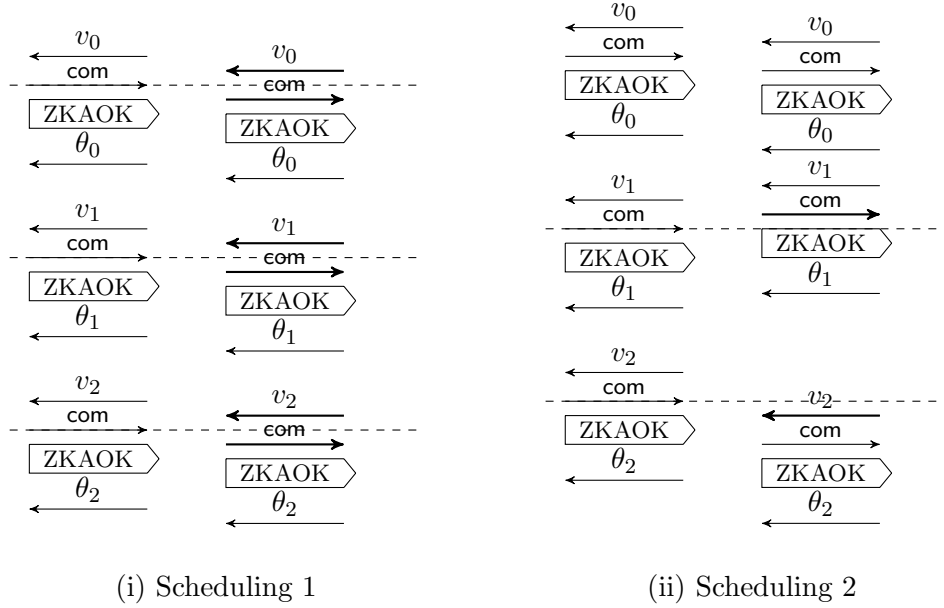


Figure 5: The two schedulings of the messages in Stage 1 of the left and right interactions.

Since Scheduling 1 and 2 are the only two possible schedulings, by our hypothesis, at least one of the following two conditions holds.

Condition 1: The probability that Scheduling 1 occurs in the Main Execution and that B' extracts an invalid decommitment from the right interaction is non-negligible.

Condition 2: The probability that Scheduling 2 occurs in the Main Execution and that B' extracts an invalid decommitment from the right interaction is non-negligible.

We show that neither condition can hold.

Assume Condition 1 holds. We reach a contradiction by constructing a machine C that externally participates in the signature game, while internally emulating an execution of $\text{STA}_b(\langle P, V \rangle, B', v, z)$ except that messages in Stage 1 of the right interaction are emulated by forwarding the appropriate messages from the signature games to A . More precisely, C forwards the three verification keys vk_0, vk_1, vk_2 in the signature game to A as the verification keys in Stage 1 of the right interaction in the Main Execution. If Schedule 1 does not occur in the Main Execution, C simply aborts. Otherwise, whenever during some rewinding, A requests another signature in one of the slots on the Main Execution (and thus using one of vk_0, vk_1, vk_2), C obtains such a signature by accessing the appropriate signature oracle in the game and forwards it to A . Recall that whenever we rewind beyond the point where a

verification key is sent, a new verification key is generated by B and thus B can obtain the appropriate signatures without querying the oracle. Since the left and right slots in the Main Execution are aligned one by one, we have that whenever the t^{th} left-slot is rewound, the adversary A may only request new signatures using key v'_t on the right. It follows that the “access-pattern” of the signatures requested is a substring of

$$\varphi = 012\|(\text{id}_l)_1^*, 2^*, \dots, (\text{id}_l)_i^*, 2^*, \dots, (\text{id}_l)_\ell^*$$

So, whenever B' extracts out a signature-chain δ w.r.t. (keys v'_0, v'_1, v'_2 and pattern $\text{pattern}(\text{id}_r)$), C wins in the signature game since $\text{pattern}(\text{id}_r)$ is not a substring of φ (as $\text{id}_r \neq \text{id}_l$). Since the running-time of C is polynomial this contradicts Lemma 1.

Assume Condition 2 holds. We construct a machine C' just as in the previous case, except that C' abort whenever Schedule 2 does not happen in the Main Execution. When Scheduling 2 does occurs in the Main Execution, there exists a right-slot t that is not aligned with any left-slots; in other words, in all the rewindings where A gets to request a new signature in Slot t on the right, the rewinding goes beyond the point where the verification key for slot t is sent (and so new keys gets generated in each rewinding) and thus the t' th oracle is *never* used during the extraction phase. It follows that the access pattern in the signature game has a single character t , but the signature extracted is with respect to a pattern with two of each character. So, as in Condition 1, whenever B' extracts out a signature-chain δ , C' wins in the signature game. There is just one slight complication with the implementation of C' : in the rewindings, B might rewind A in the middle of one of the ZKAOK in Stage 1, and since the ZKAOKs are not public-coin, we might not be able to emulate the continuation of the verifier strategy for this protocol. Note, however, that Lemma 1 still holds even if consider a slight variant of the signature game where after each ZKAOK the verifier reveals all of its random coins; this follows since this adjusted protocol would still be an ZKAOK and Lemma 1 no mayyer what ZKAOK we use in the signature game. C' can now easily be implemented as an adversary for this modified signature game.

□

□

Hybrid H_{k+6} : Hybrid H_{k+6} proceeds identically to H_{k+5} except that the Stage 2 commitment of the left execution is emulated by committing to 0^n . It follows using the same argument as in hybrids H_i , for $i \in [k+5]$, that the value committed in the right interaction can be extracted without rewinding Stage 2 of the left interaction. It then follows from the hiding property of the Stage 2 commitment that the combined view and values committed to by A in H_{k+5} are indistinguishable from that in H_{k+6} .

It follows by a hybrid argument that,

$$\left\{ \text{mim}_{\langle C, R \rangle}^A(v, z) \right\}_{n \in N, v \in \{0,1\}^n, z \in \{0,1\}^*} \approx \left\{ \text{hyb}_{k+6}(v, z) \right\}_{n \in N, v \in \{0,1\}^n, z \in \{0,1\}^*}$$

Since the above holds for every value v , we have

$$\left\{ \text{mim}_{\langle C, R \rangle}^A(v_1, z) \right\}_{n \in N, v_1, v_2 \in \{0,1\}^n, z \in \{0,1\}^*} \approx \left\{ \text{hyb}_{k+6}(v_1, z) \right\}_{n \in N, v_1, v_2 \in \{0,1\}^n, z \in \{0,1\}^*}, \text{ and}$$

$$\left\{ \text{mim}_{\langle C, R \rangle}^A(v_2, z) \right\}_{n \in N, v_1, v_2 \in \{0,1\}^n, z \in \{0,1\}^*} \approx \left\{ \text{hyb}_{k+6}(v_2, z) \right\}_{n \in N, v_1, v_2 \in \{0,1\}^n, z \in \{0,1\}^*}$$

Finally, since by the definition of hyb_{k+6} , it holds that for every v_1, v_2 and z , $\text{hyb}_{k+6}(v_1, z) = \text{hyb}_{k+6}(v_2, z)$, we conclude that,

$$\left\{ \text{mim}_{\langle C, R \rangle}^A(v_1, z) \right\}_{n \in N, v_1, v_2 \in \{0,1\}^n, z \in \{0,1\}^*} \approx \left\{ \text{mim}_{\langle C, R \rangle}^A(v_2, z) \right\}_{n \in N, v_1, v_2 \in \{0,1\}^n, z \in \{0,1\}^*}$$

□

6.1 Proof of Lemma 2

Proof of Lemma 2. The running-time of B consists of three parts:

Part 1—Time spent simulating the Main Execution: Since A runs in strict polynomial time, the time $T_1(n)$ that B spends in the Main Execution polynomially bounded.

Part 2—Time spent extracting the left “fake-witness”: We show that there exists a polynomial $T_2(n)$ such that for every polynomial q_2 , (every $b \in \{0,1\}$, every sufficiently large $n \in N$, and inputs $v \in \{0,1\}^n$, $z \in \{0,1\}^*$), the probability that B spends more than $q_2(n)T_2(n)$ steps extracting the left “fake-witness” in $\text{STA}_b(\langle P, V \rangle, B, v, z)$ is smaller than $1/q_2(n)$.

Part 3—Time spent extracting the committed value on the right: We show that there exists a polynomial $T_3(n)$ such that for every polynomial q_3 , the probability that B spends more than $q_3(n)T_3(n)$ steps extracting the right committed value in $\text{STA}_b(\langle P, V \rangle, B, v, z)$ is smaller than $1/q_3(n)$.

So given an arbitrary polynomial q , we get by the union bound that, the probability B spends more than $2q(n)T_2(n)$ step in part 2, or more than $2q(n)T_3(n)$ steps in part 3, is smaller than $1/q(n)$. We conclude that there exists some sufficiently big polynomial $T(n) \geq T_1(n) + 2T_2(n) + 2T_3(n)$ such that for every polynomial q , the probability that B takes more than $q(n)T(n)$ steps is smaller than $1/q(n)$.

Analysis of Part 2: Recall that in an execution of $\text{STA}_b(\langle P, V \rangle, B, v, z)$, the extraction of the left “fake-witness” proceeds in 2ℓ iterations. The running-time of the first iteration is clearly polynomial; we proceed to analyze the time spent in the remainder of the iterations. Recall that in an iteration $i > 1$, B takes the signature-chain $\delta_{i-1} = ([\bar{\sigma}]_1^{i-1}, [\bar{c}]_1^{i-1}, [\bar{r}]_1^{i-1})$ of length $i-1$, w.r.t. (keys v_0, v_1, v_2 and) pattern $[\psi]_1^{i-1}$, (where $\psi = \text{pattern}(\text{id}_I)$) obtained in the previous iteration, and extends it to a signature-chain $\bar{\sigma}_i$ of length i w.r.t. pattern $[\psi]_1^i$. This is done by repeatedly rewinding A from the start of the left-slot ψ_i and committing to $(i-1, \bar{\sigma}_{i-1})$ in the rewindings, until A closes this left-slot successfully. (Below we assume for simplicity that the extraction procedure is never cut-off and may run for more than $2^{n/2}$ steps, since this only increases the running time). Towards bounding the running-time of this extraction procedure, we first consider a *hypothetical procedure*, which proceeds almost the same as the actual extraction procedure, except that in the rewindings in iteration $i > 1$, instead of committing to $(i-1, \bar{\sigma}_{i-1})$, it commits to 0^m . In other words, the hypothetical procedure simulates the view of A in the rewindings using identically the same distribution as in the Main Execution. We show that the expected running-time of this hypothetical procedure is *poly*(n); we next bound the running-time of the actual extraction procedure.

Running-time Analysis of the Hypothetical Procedure: Let $\psi = \text{pattern}(\text{id}_I)$ be the pattern of the “fake-witness” of the left intearction. In iteration $i > 1$, the hypothetical extraction procedure repeatedly rewinds the left-slot ψ_i ; let $\mathcal{T}^i$ be the random variable that describes the time spent in

rewinding the left-slot ψ_i in iteration $i > 1$. We show that $E[T^i] \leq \text{poly}(n)$ and then by linearity of expectation, we conclude that the expected running-time of the hypothetical procedure is

$$\sum_{i=2}^{2\ell} E[T^i] \leq \sum_{i=2}^{2\ell} \text{poly}(n) \leq \text{poly}(n),$$

since the number of iterations is $\text{poly}(n)$.

Let us turn to bounding $E[T^i]$. Let Γ_{ψ_i} denote the set of prefixes ρ —i.e., partial transcripts of the Main Execution—from where the left-slot ψ_i opens. Given a prefix $\rho \in \Gamma_{\psi_i}$, we introduce the following notations:

- let $\Pr[\rho]$ denote the probability that ρ occurs as a prefix in the Main Execution;
- let p_ρ denote the probability that, conditioned on the prefix ρ occurring (in the Main Execution), the left-slot ψ_i closes successfully in the Main Execution.

Take any ρ from Γ_{ψ_i} . We claim that conditioned on ρ occurring, the expected value of T^i —denoted $E[T^i|\rho]$ —is $\text{poly}(n)$. This follows since, first, the hypothetical procedure starts rewinding the left-slot ψ_i in iteration i only if this slot closes successfully in the Main Execution; hence, (conditioned on ρ occurring,) the probability the left-slot ψ_i is rewound is at most p_ρ . Secondly, once it starts rewinding the left-slot ψ_i , it continues until the slot closes successfully again; since the hypothetical procedure proceeds identically in the rewindings as in the Main Execution, the probability that the left-slot ψ_i closes successfully in any rewinding is also p_ρ , and thus, (conditioned on ρ occurring,) the expected number of rewindings performed before this happens is $1/p_\rho$. Therefore, the overall expected number of rewindings from ρ is $p_\rho \times \frac{1}{p_\rho} = 1$. As each rewinding takes at most $\text{poly}(n)$ steps, we conclude that $E[T^i|\rho] \leq \text{poly}(n)$. Thus,

$$E[T^i] = \sum_{\rho \in \Gamma_{\psi_i}} E[T^i|\rho] \Pr[\rho] \leq \text{poly}(n) \times \sum_{\rho \in \Gamma_{\psi_i}} \Pr[\rho] \leq \text{poly}(n)$$

Running-time Analysis of the Actual Extraction Procedure: Given that the expected running time of the hypothetical procedure is bounded by a polynomial $\tilde{T}(n)$, it follows using the Markov inequality that, for every polynomial q_2 , (every b , every $n \in N$, and inputs v, z), the probability that the hypothetical procedure takes more than $q_2(n)\tilde{T}(n)/2$ steps is smaller than $2/q_2(n)$. Then we claim that the probability that actual extraction procedure takes more than $q_2(n)\tilde{T}(n)/2$ steps is smaller than $1/q_2(n)$. This follows since the only difference between the hypothetical and the actual extraction procedures is that, in the former the rewindings are simulated by committing to 0^m using `com`, whereas in the latter rewindings are simulated by committing to a tuple that contains a signature. Since the ZKAOK proof following the commitment is never rewound, it follows directly from the hiding property of `com` and the zero knowledge property of the ZKAOK proof that, the probability that the actual extraction procedure runs for more than $q_2(n)\tilde{T}(n)/2$ steps differs from that of the hypothetical procedure by at most a negligible amount. Thus, for sufficiently large n , we have that the probability B spends more than $T_2(n) = q_2(n)\tilde{T}(n)/2$ steps is smaller than $1/q_2(n)$.

Analysis of Part 3: We show that the time that B spends in the Rewinding Phase is bounded by a polynomial $T_3(n)$ in expectation. It then follows by the Markov inequality that, for every polynomial q_3 , the probability that B takes more than $q_3(n)T_3(n)$ steps is smaller than $1/q_3(n)$.

It follows from the same argument as in the above “running-time analysis of the hypothetical procedure” that to bound the expected time spent extracting the right committed value (also here, we consider the running-time without cut-offs), it suffices to bound the expected time spent in

rewinding each right *WISSP* proof, since the total number of right-proofs is $\text{poly}(n)$. Then recall that a right-proof is rewound only if the proof completes successfully in the Main Execution, without interleaving with any message in Stage 1 or the i^{th} proof of the left interaction. On the other hand, once the rewinding starts, it continues until this right-proof completes successfully again, while cancelling every rewinding in which the proof interleaves with any message in Stage 1 or the i^{th} proof of the left interaction. Furthermore, as every rewinding is simulated exactly the same as in the Main Execution, it follows using the same “ p times $1/p$ argument” as in the analysis of part 2 that the expected number of rewindings for every right-proof is 1, and hence the expected time spent in extracting the right committed value is bounded by a polynomial $T_3(n)$. $\square$

7 Proof of Concurrent Non-Malleability

Let us turn to proving that $\langle C, R \rangle$ is also concurrently non-malleable. Recall that by Proposition 1, to show concurrent non-malleability, it suffices to prove that $\langle C, R \rangle$ is one-many non-malleable; that is, for every one-many man-in-the-middle adversary A , that participates in one left and many right interactions, the view of A and the values it commits to on the right are indistinguishable, no matter what value it is receiving a commitment to on the left. Towards this, we consider the same hybrid experiments H_0 to H_{k+6} as in the proof of stand-alone non-malleability. It follows from almost the same proof as before that the view of A and the values it commits to on the right are indistinguishable in sequential hybrids, except that, in hybrids H_1 to H_{k+6} , we (or more precisely, the simulator B) now need to extract out the values that A commits to in *all* the right interactions (recall that the proof relies on the fact that the value that A commits to in the right interaction can be extracted “efficiently”, to show the indistinguishability of hybrid H_i and H_{i+1} for $1 \leq i \leq k+6$). This is easy to achieve, since we can simply extract the values that A commits to in each right interaction *one by one*, after the Main Execution completes. More precisely, in the Rewinding Phase, for *every* successful right interaction that has a different identity from the left interaction in the Main Execution, B finds a *WISSP* proof in Stage 3 of this right interaction that does not interleave with any message in Stage 1 and the i^{th} proof (or Stage 2 for hybrid H_{k+6}) of that left interaction, and repeatedly rewinds the proof until a second transcript is obtained; it then computes a witness, if the two transcripts are different. Since there are only polynomial number of right interactions, it follows using almost the same proof of Lemma 2 that the running time of B is “bounded”, and further using exactly the same proof of Lemma 3 that, except with negligible probability, the witnesses that B extracts out are indeed the values committed to in the right interactions. Thus by the *WI* property of the Stage 3 proofs (or the hiding property of Stage 2 resp.), the view and the values committed to by A are indistinguishable in hybrids H_i and H_{i+1} for $1 \leq i \leq k+4$ (or in H_{k+5} and H_{k+6} resp.). We thus have:

Theorem 6. $\langle C, R \rangle$ is concurrent non-malleable.

8 Proof of Robust Non-Malleability

In this section, we show that, for any $r \in N$, $\langle C, R \rangle$ can be easily modified into a $O(r)$ -round (concurrent) non-malleable commitment scheme $\langle \tilde{C}, \tilde{R} \rangle$ that is additionally one-many r -robust.

It is shown in [?] that one-many r -robust commitment schemes are easy to construct: any commitment scheme that is “extractable” and has more than r “rewinding slots” is directly one-many non-malleable w.r.t. r -round protocols. Therefore, to make our constant-round non-malleable commitment scheme $\langle C, R \rangle$ one-many r -robust, we simply add more *WISSP* proofs in Stage 3 of the

protocol. More precisely, the commitment scheme $\langle C_r, R_r \rangle$ proceeds identitically to $\langle C, R \rangle$, except that in Stage 3 of the protocol, the Committer $\tilde{C}$ needs to provide $\max(r+1, l)$ *WLSSP* proofs (of the statement that either the Stage 2 message is a valid commitment or that it knows a “trapdoor”), where l is the number of *WLSSP* proofs in Stage 3 of the original protocol $\langle C, R \rangle$. It follows using the same proof as in [?] that $\langle C_r, R_r \rangle$ is one-many r -robust. Roughly speaking, the main idea of the proof is to reduce the one-many r -robustness to the indistinguishability of the interaction with machine $B(y_n^1)$ or $B(y_n^2)$, by extracting the value committed to in the right interactions from the *WLSSP* proofs in Stage 3 of the protocol, *without rewinding the left interactions*. This is achievable, (similar to the proof of the indistinguishability of Hybrid H_i and H_{i+1} in Section 6,) as there are more *WLSSP* proofs in Stage 3 than the nubmer of messages in the left interaction, and one can always find a *WLSSP* proof that does not interleave with the left interaction and extract a witness from this proof, without rewinding the left interactions. The witness extracted must be a valid decommitment, as otherwise, by the special-soundness of the proof, it must be a valid signature-chain, which violates the soundness of the signature-game (since the adversary here is never rewound and obtains only three signatures during the straight-line execution of the right interaction). Therefore, we conclude that $\langle C_r, R_r \rangle$ is one-many r -robust. It follows using the same proof in Section 6 that $\langle C_r, R_r \rangle$ is stand-alone non-malleable; and it further follows using the same proof as in Section 7 that it is, in fact, also concurrent non-malleable.

Lemma 4. *For every $r \in N$, the protocol $\langle C_r, R_r \rangle$ has $O(r)$ -round, and is concurrently non-malleable and one-many r -robust.*

Theorem 2 follows directly from Lemma 4. Furthermore, for $r < l$, the protocol $\langle C_r, R_r \rangle$ is the same as $\langle C, R \rangle$; thus,

Corollary 1. *For any $r < l$, $\langle C, R \rangle$ is concurrently non-malleable and one-many r -robust.*

9 Acknowledgements

We are very grateful to Boaz Barak, Ran Canetti, Danny Dolev, Cynthia Dwork, Johan Håstad, Oded Goldreich, Shafi Goldwasser, Silvio Micali, Moni Naor, Tal Rabin, Alon Rosen, Amit Sahai, Wei-lung Tseng, Muthuramakrishnan Venkatasubramanian and Hoeteck Wee for many enlightening discussions about non malleability over the years. We are particularly grateful to Oded Goldreich for encouraging us to include a self-contained description of a non-malleable commitment for constant length identities into the journal version of [?]; the ideas in this paper came out of thinking about how to simplify the presentation of this (weak) primitive. The second author is also indebted to Alon Rosen for introducing him to the area of non malleability, and for many fruitful discussions about it.

A General Definitions

A.1 Witness Relations

We recall the definition of a witness relation for a $\mathcal{NP}$ language [?].

Definition 4 (Witness relation). *A witness relation for a language $L \in \mathcal{NP}$ is a binary relation R_L that is polynomially bounded, polynomial time recognizable and characterizes L by $L = \{x : \exists y \text{ s.t. } (x, y) \in R_L\}$*

We say that y is a witness for the membership $x \in L$ if $(x, y) \in R_L$. We will also let $R_L(x)$ denote the set of witnesses for the membership $x \in L$, i.e., $R_L(x) = \{y : (x, y) \in R_L\}$. In the following, we assume a fixed witness relation R_L for each language $L \in \mathcal{NP}$.

A.2 Indistinguishability

Definition 5 (Computational Indistinguishability). *Let Y be a countable set. Two ensembles $\{A_{n,y}\}_{n \in N, y \in Y}$ and $\{B_{n,y}\}_{n \in N, y \in Y}$ are said to be **computationally indistinguishable** (denoted by $\{A_{n,y}\}_{n \in N, y \in Y} \approx \{B_{n,y}\}_{n \in N, y \in Y}$), if for every PPT “distinguishing” machine D , there exists a negligible function $\nu(\cdot)$ so that for every $n \in N, y \in Y$:*

$$|\Pr[a \leftarrow A_{n,y} : D(1^n, y, a) = 1] - \Pr[b \leftarrow B_{n,y} : D(1^n, y, b) = 1]| < \nu(n)$$

A.3 Interactive Proofs

We use the standard definitions of interactive proofs (and interactive Turing machines) [?]. Given a pair of interactive Turing machines, P and V , we denote by $\langle P(w), V \rangle(x)$ the random variable representing the (local) output of V , on common input x , when interacting with machine P with private input w , when the random input to each machine is uniformly and independently chosen.

Definition 6 (Interactive Proof System). *A pair of interactive machines $\langle P, V \rangle$ is called an **interactive proof system** for a language L if there is a negligible function $\nu(\cdot)$ such that the following two conditions hold :*

- **Completeness:** *For every $x \in L$, and every $w \in R_L(x)$, $\Pr[\langle P(w), V \rangle(x) = 1] = 1$*
- **Soundness:** *For every $x \in \{0, 1\}^n - L$, and every interactive machine B , $\Pr[\langle B, V \rangle(x) = 1] \leq \nu(n)$*

*In case that the soundness condition is required to hold only with respect to a computationally bounded prover, the pair $\langle P, V \rangle$ is called an **interactive argument system**.*

A.4 Zero-Knowledge

We recall the standard definition of $\mathcal{ZK}$ proofs. Loosely speaking, an interactive proof is said to be **zero-knowledge** ($\mathcal{ZK}$) if a verifier V learns nothing beyond the validity of the assertion being proved, it could not have generated on its own. As “feasible” computation in general is defined though the notion of probabilistic polynomial-time, this notion is formalized by requiring that the output of every (possibly malicious) verifier interacting with the honest prover P can be “simulated” by a probabilistic expected polynomial-time machine S (a.k.a. the *simulator*). The idea behind this definition is that whatever V^* might have learned from interacting with P , he could have learned by himself by running the simulator S .

The notion of $\mathcal{ZK}$ was introduced and formalized by Goldwasser, Micali and Rackoff in [?]. We present their definition below.

Definition 7 ($\mathcal{ZK}$). *Let L be a language in $\mathbf{NP}$, R_L a witness relation for L , (P, V) an interactive proof (argument) system for L . We say that (P, V) is **statistical/computational $\mathcal{ZK}$** , if for every probabilistic polynomial-time interactive machine V there exists a probabilistic algorithm S whose expected running-time is polynomial in the length of its first input, such that the following ensembles are statistically close/computationally indistinguishable over L .*

- $\left\{ \langle P(y), V(z) \rangle(x) \right\}_{n \in N, x \in \{0, 1\}^n \cap L, y \in R_L(x), z \in \{0, 1\}^*}$
- $\left\{ S(x, z) \right\}_{n \in N, x \in \{0, 1\}^n \cap L, y \in R_L(x), z \in \{0, 1\}^*}$

where $\langle P(y), V(z) \rangle(x)$ denotes the view of V in interaction with P on common input x and private inputs y and z respectively.

A.5 Witness Indistinguishability

An interactive proof (or argument) is said to be *witness indistinguishable* ($\mathcal{WI}$) if the verifier's output is “computationally independent” of the witness used by the prover for proving the statement. In this context, we focus on languages $L \in \mathcal{NP}$ with a corresponding witness relation R_L . Namely, we consider interactions in which, on common input x , the prover is given a witness in $R_L(x)$. By saying that the output is computationally independent of the witness, we mean that for any two possible $\mathbf{NP}$ -witnesses that could be used by the prover to prove the statement $x \in L$, the corresponding outputs are computationally indistinguishable.

Definition 8 (Witness-indistinguishability). *Let $\langle P, V \rangle$ be an interactive proof (or argument) system for a language $L \in \mathcal{NP}$. We say that $\langle P, V \rangle$ is witness-indistinguishable for R_L , if for every probabilistic polynomial-time interactive machine V^* and for every two sequences $\{w_{n,x}^1\}_{n \in N, x \in L}$ and $\{w_{n,x}^2\}_{n \in N, x \in L}$, such that $w_{n,x}^1, w_{n,x}^2 \in R_L(x)$ for every $x \in L \cap \{0, 1\}^n$, the following probability ensembles are computationally indistinguishable over $n \in N$.*

- $\{\langle P(w_{n,x}^1), V^*(z) \rangle(x)\}_{n \in N, x \in L \cap \{0, 1\}^n, z \in \{0, 1\}^*}$
- $\{\langle P(w_{n,x}^2), V^*(z) \rangle(x)\}_{n \in N, x \in L \cap \{0, 1\}^n, z \in \{0, 1\}^*}$

A.6 Proofs (Arguments) of Knowledge

Loosely speaking, an interactive proof is a proof of knowledge if the prover convinces the verifier that it *possesses*, or can *feasibly compute*, a witness for the statement proved. The notion of a proof of knowledge is essentially formalized as follows: an interactive proof of $x \in L$ is a proof of knowledge if there exists a probabilistic expected polynomial-time *extractor* machine E , such that for any prover P , E on input the description of P and any statement $x \in L$ readily outputs a valid witness for $x \in L$ if P succeeds in convincing the Verifier that $x \in L$. Formally,

[Proof of knowledge [?]] Let (P, V) be an interactive proof system for the language L . We say that (P, V) is a *proof of knowledge* for the witness relation R_L for the language L if there exists a probabilistic expected polynomial-time machine E , called the extractor, and a negligible function $\nu(n)$ such that for every machine P^* , every statement $x \in \{0, 1\}^n$, every random tape $r \in \{0, 1\}^*$ and every auxiliary input $z \in \{0, 1\}^*$,

$$\Pr [\langle P_r'(z), V \rangle(x) = 1] \leq \Pr [E^{P_r'(x,z)}(x) \in R_L(x)] + \nu(n)$$

consider $\mathcal{PPT}$ provers. An interactive argument system $\langle P, V \rangle$ is an *argument of knowledge* if the above condition holds w.r.t. probabilistic polynomial-time provers.

Special-sound $\mathcal{WI}$ proofs A 4-round public-coin interactive proof for the language $L \in \mathcal{NP}$ with witness relation R_L is *special-sound* with respect to R_L , if for any two transcripts $(\delta, \alpha, \beta, \gamma)$ and $(\delta', \alpha', \beta', \gamma')$ such that the initial two messages, δ, δ' and α, α' , are the same but the challenges β, β' are different, there is a deterministic procedure to extract the witness from the two transcripts and runs in polynomial time. Special-sound $\mathcal{WI}$ proofs for languages in $\mathcal{NP}$ can be based on the existence of 2-round commitment schemes, which in turn can be based on one-way functions [?, ?, ?, ?].

Adaptively Secure Puncturable Pseudorandom Functions in the Standard Model

Susan Hohenberger*
Johns Hopkins University
susan@cs.jhu.edu

Venkata Koppula
University of Texas at Austin
kvenkata@cs.utexas.edu

Brent Waters†
University of Texas at Austin
bwaters@cs.utexas.edu

June 13, 2015

Abstract

We study the adaptive security of constrained PRFs in the standard model. We initiate our exploration with puncturable PRFs. A puncturable PRF family is a special class of constrained PRFs, where the constrained key is associated with an element x' in the input domain. The key allows evaluation at all points $x \neq x'$.

We show how to build puncturable PRFs with adaptive security proofs in the standard model that involve only polynomial loss to the underlying assumptions. Prior work had either super-polynomial loss or applied the random oracle heuristic. Our construction uses indistinguishability obfuscation and DDH-hard algebraic groups of composite order.

More generally, one can consider a t -puncturable PRF: PRFs that can be punctured at any set of inputs S , provided the size of S is less than a fixed polynomial. We additionally show how to transform any (single) puncturable PRF family to a t -puncturable PRF family, using indistinguishability obfuscation.

*Supported by the National Science Foundation (NSF) CNS-1154035 and CNS-1228443; the Defense Advanced Research Projects Agency (DARPA) and the Air Force Research Laboratory (AFRL) under contract FA8750-11-C-0080, the Office of Naval Research under contract N00014-14-1-0333, and a Microsoft Faculty Fellowship.

†Supported by NSF CNS-0915361 and CNS-0952692, CNS-1228599 DARPA through the U.S. Office of Naval Research under Contract N00014-11-1-0382, Google Faculty Research award, the Alfred P. Sloan Fellowship, Microsoft Faculty Fellowship, and Packard Foundation Fellowship.

1 Introduction

Pseudorandom functions (PRFs) are one of the fundamental building blocks in modern cryptography. A PRF system consists of a keyed function F and a set of keys $\mathcal{K}$ such that for a randomly chosen key $k \in \mathcal{K}$, the output of the function $F(k, x)$ for any input x in the input space “looks” random to a computationally bounded adversary, even when given polynomially many evaluations of $F(k, \cdot)$. Recently, the concept of *constrained pseudorandom functions*¹ was proposed in the concurrent works of Boneh and Waters [BW13], Boyle, Goldwasser and Ivan [BGI13] and Kiayias, Papadopoulos, Triandopoulos and Zacharias [KPTZ13]. A constrained PRF system is associated with a family of boolean functions $\mathcal{F} = \{f\}$. As in standard PRFs, there exists a set of *master keys* $\mathcal{K}$ that can be used to evaluate the PRF F . However, given a master key k , it is also possible to derive a *constrained* key k_f associated with a function $f \in \mathcal{F}$. This constrained key k_f can be used to evaluate the function $F(k, \cdot)$ at all inputs x such that $f(x) = 1$. Intuitively, we would want that even if an adversary has k_f , the PRF evaluation at an input x not accepted by f looks random. Security is captured by an *adaptive* game between a PRF challenger and an adversary. The adversary is allowed to make multiple constrained key or point evaluation queries before committing to a challenge x^* not equal to any of the evaluation queries or accepted by any of the functions for which he obtained a constrained key.² The challenger either sends the PRF evaluation at x^* or an output chosen uniformly at random from the PRF range space, and the adversary wins if he can distinguish between these two cases.

Since their inception, constrained PRFs have found multiple applications. For example, Boneh and Waters [BW13] gave applications of broadcast encryption with optimal ciphertext length, identity-based key exchange, and policy-based key distribution. Sahai and Waters [SW14] used constrained PRFs as a central ingredient in their punctured programming methodology for building cryptosystems using indistinguishable obfuscation. Boneh and Zhandry [BZ14] likewise applied constrained PRFs for realizing multi-party key exchange and broadcast systems.

Adaptive Security in Constrained PRFs In their initial work, Boneh and Waters [BW13] showed constructions of constrained PRFs for different function families, including one for the class of all polynomial circuits (based on multilinear maps). However, all their constructions offer *selective security* - a weaker notion where the adversary must commit to the challenge input x^* *before* making any evaluation/constrained key queries.³ Using complexity leveraging, one can obtain adaptive security by guessing the challenge input x^* before any queries are made. However, this results in exponential security loss. The works of [BGI13, KPTZ13] similarly dealt with selective security.

Recently, Fuchsbauer, Konstantinov, Pietrzak and Rao [FKPR14] showed adaptive security for *prefix-fixing* constrained PRFs, but with quasi-polynomial security loss. Also recently, Hofheinz [Hof14] presented a novel construction that achieves adaptive security for *bit-fixing* constrained PRFs, but in the *random oracle model*.

While selective security has been sufficient for some applications of constrained PRFs, including many recent proofs leveraging the punctured programming [SW14] methodology (e.g., [SW14, HSW14, BZ14, BCPR13]), there are applications that demand adaptive security, where the security game allows the adversary to query the PRF on many inputs before deciding on the point to puncture. For instance, [BZ14] give a construction for multiparty key exchange that is semi-statically secure, and this construction requires adaptively secure constrained PRFs for circuits. We anticipate that the further realization of adaptively secure PRFs will introduce further applications of them.

Our Objective and Results Our goal is to study adaptive security of constrained PRFs in the standard model. We initiate this exploration with *puncturable PRFs*, first explicitly introduced in [SW14] as a specialization of constrained PRFs. A puncturable PRF family is a special class of constrained PRFs, where the constrained key is associated with an element x' in the input domain. The key allows evaluation at all points

¹These were alternatively called *functional PRFs* [BGI13] and *delegatable PRFs* [KPTZ13].

²This definition can be extended to handle multiple challenge points. See Section 3 for details.

³The prefix construction of [BGI13] and [KPTZ13] were also selective.

$x \neq x'$. As noted by [BW13, BGI13, KPTZ13], the GGM tree-based construction of PRFs from one-way functions (OWFs) [GGM84] can be modified to construct a puncturable PRF.⁴ A selective proof of security follows via a hybrid argument, where the reduction algorithm uses the pre-determined challenge query x^* to “plant” its OWF challenge. However, such a technique does not seem powerful enough to obtain adaptive security with only a polynomial-factor security loss. The difficulty in proving adaptive security arises due to the fact that the reduction algorithm must respond to the evaluation queries, and then output a punctured key that is consistent with the evaluations. This means that the reduction algorithm must be able to evaluate the PRF at a large set S (so that all evaluation queries lie in S with non-negligible probability). However, S cannot be very large, otherwise the challenge x^* will lie in S , in which case the reduction algorithm cannot use the adversary’s output.

In this work, we show new techniques for constructing adaptively-secure puncturable PRFs in the standard model. A central contribution is to overcome the conflict above, by allowing the reduction algorithm to commit to the evaluation queries, and at the same time, ensuring that the PRF output at the challenge point is unencumbered by the commitment.

Our main idea is to execute a delayed commitment to part of the PRF by partitioning. Initially, in our construction all points are tied to a single (Naor-Reingold [NR04] style) PRF. To prove security we begin by using the admissible hash function of Boneh and Boyen [BB04]. We partition the inputs into two distinct sets. The *evaluable set* which contains about $(1 - 1/q)$ fraction of inputs, and a *challenge set* which contains about $1/q$ fraction of inputs, where q is the number of point evaluation queries made by the attacker. Via a set of hybrid steps using the computational assumptions of indistinguishability obfuscation and subgroup hiding we modify the construction such that we use one Naor-Reingold PRF function to evaluate points in the evaluable set and a completely independent Naor-Reingold PRF to evaluate points in the challenge set.

After this separation has been achieved, there is a clearer path for our proof of security. At this point the reduction algorithm will create one PRF itself and use it to answer any attacker point query in the evaluable set. If it is asked for a point x in the challenge set, it will simply abort. (The admissible hash function ensures that we get through without abort with some non-negligible probability.) Eventually, the attacker will ask for a punctured key on x^* , which defines x^* as the challenge input. Up until this point the reduction algorithm has made no commitments on what the second challenge PRF is. It then constructs the punctured key using the a freshly chosen PRF for the challenge inputs. However, when constructing this second PRF it now knows what the challenge x^* actually is and can fall back on selective techniques for completing the proof.

At a lower level our core PRF will be the Naor-Reingold PRF [NR04], but based in composite-order groups. Let $\mathbb{G}$ be a group of order $N = pq$, where p and q are primes. The master key consists of a group element $v \in \mathbb{G}$ and $2n$ exponents $d_{i,b} \in \mathbb{Z}_N$ (for $i = 1$ to n and $b \in [0, 1]$). The PRF F takes as input a key $k = (v, \{d_{i,b}\})$, an ℓ -bit input x , uses a public admissible hash function $h : \{0, 1\}^\ell \rightarrow \{0, 1\}^n$ to compute $h(x) = b_1 \dots b_n$ and outputs $v^{\prod_{j=1}^n d_{j,b_j}}$. A punctured key corresponding to x' derived from master key k is the obfuscation of a program P which has k, x' hardwired and outputs $F(k, x)$ on input $x \neq x'$, else it outputs $\perp$.

We will use a parameterized problem (in composite groups) to perform some of the separation step. Our assumption is that given $g, g^a, \dots, g^{a^{n-1}}$ for randomly chosen $g \in \mathbb{G}$ and $a \in \mathbb{Z}_N^*$ it is hard to distinguish g^{a^n} from a random group element. While it is somewhat undesirable to base security on a parameterized assumption, we are able to use the recent results of Chase and Meiklejohn [CM14] to reduce this to the subgroup decision problem in DDH hard composite order groups.

t -puncturable PRFs We also show how to construct t -puncturable PRFs: PRFs that can be punctured at any set of inputs S , provided $|S| \leq t$ (where $t(\cdot)$ is a fixed polynomial). We show how to transform any (single) puncturable PRF family to a t -puncturable PRF family, using indistinguishability obfuscation. In the security game for t -puncturable PRFs, the adversary is allowed to query for multiple t -punctured keys, each corresponding to a set S of size at most t . Finally, the adversary sends a challenge input x^* that lies

⁴In fact, the GGM PRF construction can be used to construct prefix-fixing constrained PRFs.

in all the sets queried, and receives either the PRF evaluation at x^* or a uniformly random element of the range space.

In the construction, the setup and evaluation algorithm for the t -puncturable PRF are the same as those for the puncturable PRF. In order to puncture a key k at set S , the puncturing algorithm outputs the obfuscation of a program P that takes as input x , checks that $x \notin S$, and outputs $F(k, x)$.

For the proof of security, we observe that when the first t -punctured key query S_1 is made by the adversary, the challenger can guess the challenge $\tilde{x} \in S_1$. If this guess is incorrect, then the challenger simply aborts (which results in a $1/t$ factor security loss). However, if the guess is correct, then the challenger can now use the punctured key $K_{\tilde{x}}$ for all future evaluation/ t -punctured key queries. From the security of puncturable PRFs, it follows that even after receiving evaluation/ t -punctured key queries, the challenger will not be able to distinguish between $F(k, \tilde{x})$ and a random element in the range space.

We detail this transformation and its proof in Section 5. We also believe that we can use a similar approach to directly modify our main construction to handle multiple punctured points, however, we choose to focus on the generic transformation.

Related Works Two recent works have explored the problem of adaptive security of constrained PRFs. Fuchsbauer, Konstantinov, Pietrzak and Rao [FKPR14] study the adaptive security of the GGM construction for prefix-free constrained PRFs. They show an interesting reduction to OWFs that suffers only a quasi-polynomial factor $q^{O(\log n)}$ loss, where q is the number of queries made by the adversary, and n is the length of the input. This beats the straightforward conversion from selective to adaptive security, which results in $O(2^n)$ security loss.

Hofheinz [Hof14] shows a construction for bit-fixing constrained PRFs that is adaptively secure, assuming indistinguishability obfuscation and multilinear maps in the *random oracle model*. It also makes novel use of the random oracle for dynamically defining the challenge space based on the output of h . It is currently unclear whether such ideas could be adapted to the standard model.

Fuchsbauer et al. also show a negative result for the Boneh-Waters [BW13] construction of bit-fixing constrained PRFs. They show that any *simple* reduction from a static assumption to the adaptive security of the Boneh-Waters [BW13] bit-fixing constrained PRF construction must have an exponential factor security loss. More abstractly, using their techniques, one can show that any bit-fixing scheme that has the following properties will face this obstacle: (a) *fingerprinting queries* - By querying for a set of constrained keys, the adversary can obtain a *fingerprint* of the master key. (b) *checkability* - It is possible to efficiently check that any future evaluation/constrained key queries are consistent with the fingerprint. While these properties capture certain constructions, small perturbations to them could potentially circumvent checkability.

Partitioning type proofs have been used in several applications including identity-based encryption [BB04, Wat05, ABB10, HK12], verifiable random functions [HW10], and proofs of certain signature schemes [FHPS13, HSW13, HSW14]. We believe ours is the first to use partitioning for a delayed commitment to parameters. We note that our delayed technique is somewhat reminiscent to that of Lewko and Waters [LW12].

Recently, there has been a push to prove security for indistinguishability obfuscation from basic multilinear map assumptions. The recent work of Gentry, Lewko, Sahai and Waters [GLSW14] is a step in this direction, but itself requires the use of complexity leveraging. In the future work, we might hope for such reductions with just polynomial loss — perhaps for special cases of functionality. And thus give an end-to-end polynomial loss proof of puncturable PRFs from multilinear maps assumptions.

Two works have explored the notion of constrained verifiable random functions (VRFs). Fuchsbauer [?] and Chandran, Raghuraman and Vinayagamurthy [?] show constructions of selectively secure constrained VRFs for the class of all polynomial sized circuits. The construction in [?] is also delegatable.

Future Directions A natural question is to construct adaptively-secure constrained PRFs for larger classes of functions in the standard model. Given the existing results of [FKPR14] and [Hof14], both directions seem possible. While the techniques of [Hof14] are intricately tied to the random oracle model, it is plausible there could be constructions in the standard model that evade the negative result of [FKPR14]. On the other

hand, maybe the negative result of [FKPR14] (which is specific to the [BW13] construction) can be extended to show a similar lower bound for all constructions of constrained PRFs with respect to function family $\mathcal{F}$.

2 Preliminaries

First, we recall the notion of *admissible hash functions* due to Boneh and Boyen [BB04]. Here we state a simplified definition from [HSW14].

Definition 2.1. Let l, n and θ be efficiently computable univariate polynomials. Let $h : \{0, 1\}^{l(\lambda)} \rightarrow \{0, 1\}^{n(\lambda)}$ be an efficiently computable function and AdmSample a PPT algorithm that takes as input 1^λ and an integer q , and outputs $u \in \{0, 1, \perp\}^{n(\lambda)}$. For any $u \in \{0, 1, \perp\}^{n(\lambda)}$, define $P_u : \{0, 1\}^{l(\lambda)} \rightarrow \{0, 1\}$ as follows: $P_u(x) = 0$ if for all $1 \leq j \leq n(\lambda)$, $h(x)_j \neq u_j$, else $P_u(x) = 1$.

We say that $(h, \text{AdmSample})$ is θ -admissible if the following condition holds:

For any efficiently computable polynomial Q , for all $x_1, \dots, x_{Q(\lambda)}, x^* \in \{0, 1\}^{l(\lambda)}$, where $x^* \notin \{x_i\}_i$,

$$\Pr[(\forall i \leq Q(\lambda), P_u(x_i) = 1) \wedge P_u(x^*) = 0] \geq \frac{1}{\theta(Q(\lambda))}$$

where the probability is taken over $u \leftarrow \text{AdmSample}(1^\lambda, Q(\lambda))$.

Theorem 2.1 (Admissible Hash Function Family [BB04], simplified proof in [FHPS13]). For any efficiently computable polynomial l , there exist efficiently computable polynomials n, θ such that there exist θ -admissible function families mapping l bits to n bits.

Next, we recall the definition of indistinguishability obfuscation from [GGH⁺13, SW14]. Let PPT denote probabilistic polynomial time.

Definition 2.2. (Indistinguishability Obfuscation) A uniform PPT machine $i\mathcal{O}$ is called an indistinguishability obfuscator for a circuit class $\{\mathcal{C}_\lambda\}$ if it satisfies the following conditions:

- (Preserving Functionality) For all security parameters $\lambda \in \mathbb{N}$, for all $C \in \mathcal{C}_\lambda$, for all inputs x , we have that $C'(x) = C(x)$ where $C' \leftarrow i\mathcal{O}(\lambda, C)$.
- (Indistinguishability of Obfuscation) For any (not necessarily uniform) PPT distinguisher $\mathcal{B} = (\text{Samp}, \mathcal{D})$, there exists a negligible function $\text{negl}(\cdot)$ such that the following holds: if for all security parameters $\lambda \in \mathbb{N}$, $\Pr[\forall x, C_0(x) = C_1(x) : (C_0; C_1; \sigma) \leftarrow \text{Samp}(1^\lambda)] > 1 - \text{negl}(\lambda)$, then

$$\begin{aligned} &|\Pr[\mathcal{D}(\sigma, i\mathcal{O}(\lambda, C_0)) = 1 : (C_0; C_1; \sigma) \leftarrow \text{Samp}(1^\lambda)] - \\ &\Pr[\mathcal{D}(\sigma, i\mathcal{O}(\lambda, C_1)) = 1 : (C_0; C_1; \sigma) \leftarrow \text{Samp}(1^\lambda)]| \leq \text{negl}(\lambda). \end{aligned}$$

In a recent work, [GGH⁺13] showed how indistinguishability obfuscators can be constructed for the circuit class P/poly . We remark that $(\text{Samp}, \mathcal{D})$ are two algorithms that pass state, which can be viewed equivalently as a single stateful algorithm $\mathcal{B}$. In our proofs we employ the latter approach, although here we state the definition as it appears in prior work.

2.1 Assumptions

Let $\mathcal{G}$ be a PPT group generator algorithm that takes as input the security parameter 1^λ and outputs $(N, p, q, \mathbb{G}, \mathbb{G}_p, \mathbb{G}_q, g_1, g_2)$ where $p, q \in \Theta(2^\lambda)$ are primes, $N = pq$, $\mathbb{G}$ is a group of order N , $\mathbb{G}_p$ and $\mathbb{G}_q$ are subgroups of $\mathbb{G}$ of order p and q respectively, and g_1 and g_2 are generators of $\mathbb{G}_p$ and $\mathbb{G}_q$ respectively.

Assumption 1 (Subgroup Hiding for Composite Order DDH-Hard Groups). Let $(N, p, q, \mathbb{G}, \mathbb{G}_p, \mathbb{G}_q, g_1, g_2) \leftarrow \mathcal{G}(1^\lambda)$ and $b \leftarrow \{0, 1\}$. Let $T \leftarrow \mathbb{G}$ if $b = 0$, else $T \leftarrow \mathbb{G}_p$. The advantage of algorithm $\mathcal{A}$ in solving Assumption 1 is defined as

$$\text{Adv}_{\mathcal{A}}^{\text{SGH}} = \left| \Pr[b \leftarrow \mathcal{A}(N, \mathbb{G}, \mathbb{G}_p, \mathbb{G}_q, g_1, g_2, T)] - \frac{1}{2} \right|$$

We say that Assumption 1 holds if for all PPT $\mathcal{A}$, $\text{Adv}_{\mathcal{A}}^{\text{SGH}}$ is negligible in λ .

Note that the adversary $\mathcal{A}$ gets generators for both subgroups $\mathbb{G}_p$ and $\mathbb{G}_q$. This is in contrast to bilinear groups, where, if given generators for both subgroups, the adversary can use the pairing to distinguish a random group element from a random subgroup element.

Analogously, we assume that no PPT adversary can distinguish between a random element of $\mathbb{G}$ and a random element of $\mathbb{G}_q$ with non-negligible advantage. This is essentially Assumption 1, where prime q is chosen instead of p , and $\mathbb{G}_q$ is chosen instead of $\mathbb{G}_p$.

Assumption 2. This assumption is parameterized with an integer $n \in \mathbb{Z}$. Let $(N, p, q, \mathbb{G}, \mathbb{G}_p, \mathbb{G}_q, g_1, g_2) \leftarrow \mathcal{G}(1^\lambda)$, $g \leftarrow \mathbb{G}$, $a \leftarrow \mathbb{Z}_N^*$ and $b \leftarrow \{0, 1\}$. Let $D = (N, \mathbb{G}, \mathbb{G}_p, \mathbb{G}_q, g_1, g_2, g, g^a, \dots, g^{a^{n-1}})$. Let $T = g^{a^n}$ if $b = 0$, else $T \leftarrow \mathbb{G}$. The advantage of algorithm $\mathcal{A}$ in solving Assumption 2 is defined as

$$\text{Adv}_{\mathcal{A}} = \left| \Pr[b \leftarrow \mathcal{A}(D, T)] - \frac{1}{2} \right|$$

We say that Assumption 2 holds if for all PPT $\mathcal{A}$, $\text{Adv}_{\mathcal{A}}$ is negligible in λ .

We will use Assumption 2 for clarity in certain parts of our proof, but we do not give it a name because it is implied by other named assumptions. First, Assumption 2 is implied by the n -Power Decisional Diffie-Hellman Assumption [GJM02]. Second, it is also implied by the non-parameterized Assumption 1. The recent results of Chase and Meiklejohn [CM14] essentially show this latter implication, but that work focuses on the target groups of bilinear maps, whereas our algebraic focus does not involve bilinear maps. For completeness, we give a proof of this implication in Appendix B.

3 Constrained Pseudorandom Functions

In this section, we define the syntax and security properties of a constrained pseudorandom function family. This definition is similar to the one in Boneh-Waters [BW13], except that the keys are constrained with respect to a circuit family instead of a set system.

Let $\mathcal{K}$ denote the key space, $\mathcal{X}$ the input domain and $\mathcal{Y}$ the range space. The PRF is a function $F : \mathcal{K} \times \mathcal{X} \rightarrow \mathcal{Y}$ that can be computed by a deterministic polynomial time algorithm. We will assume there is a Setup algorithm $F.\text{setup}$ that takes the security parameter λ as input and outputs a random secret key $k \in \mathcal{K}$.

A PRF $F : \mathcal{K} \times \mathcal{X} \rightarrow \mathcal{Y}$ is said to be *constrained* with respect to a circuit family $\mathcal{C}$ if there is an additional key space $\mathcal{K}_c$, and three algorithms $F.\text{setup}$, $F.\text{constrain}$ and $F.\text{eval}$ as follows:

- $F.\text{setup}(1^\lambda)$ is a PPT algorithm that takes the security parameter λ as input and outputs a description of the key space $\mathcal{K}$, the constrained key space $\mathcal{K}_c$ and the PRF F .
- $F.\text{constrain}(k, C)$ is a PPT algorithm that takes as input a PRF key $k \in \mathcal{K}$ and a circuit $C \in \mathcal{C}$ and outputs a constrained key $k_C \in \mathcal{K}_c$.
- $F.\text{eval}(k_C, x)$ is a deterministic polynomial time algorithm that takes as input a constrained key $k_C \in \mathcal{K}_c$ and $x \in \mathcal{X}$ and outputs an element $y \in \mathcal{Y}$. Let k_C be the output of $F.\text{constrain}(k, C)$. For correctness, we require the following:

$$F.\text{eval}(k_C, x) = \begin{cases} F(k, x) & \text{if } C(x) = 1 \\ \perp & \text{otherwise} \end{cases}$$

3.1 Security of Constrained Pseudorandom Functions

Intuitively, we require that even after obtaining several constrained keys, no polynomial time adversary can distinguish a truly random string from the PRF evaluation at a point not accepted by the queried circuits. This intuition can be formalized by the following security game between a challenger and an adversary A .

Let $F : \mathcal{K} \times \mathcal{X} \rightarrow \mathcal{Y}$ be a constrained PRF with respect to a circuit family $\mathcal{C}$. The security game consists of three phases.

Setup Phase The challenger chooses a random key $k \leftarrow \mathcal{K}$ and a random bit $b \leftarrow \{0, 1\}$.

Query Phase In this phase, A is allowed to ask for the following queries:

- **Evaluation Query** A sends $x \in \mathcal{X}$, and receives $F(k, x)$.
- **Key Query** A sends a circuit $C \in \mathcal{C}$, and receives $F.\text{constrain}(k, C)$.
- **Challenge Query** A sends $x \in \mathcal{X}$ as a challenge query. If $b = 0$, the challenger outputs $F(k, x)$. Else, the challenger outputs a random element $y \leftarrow \mathcal{Y}$.

Guess A outputs a guess b' of b .

Let $E \subset \mathcal{X}$ be the set of evaluation queries, $L \subset \mathcal{C}$ be the set of constrained key queries and $Z \subset \mathcal{X}$ the set of challenge queries. A wins if $b = b'$ and $E \cap Z = \emptyset$ and for all $C \in L, z \in Z, C(z) = 0$. The advantage of A is defined to be $\text{Adv}_A^F(\lambda) = \Pr[A \text{ wins}]$.

Definition 3.1. The PRF F is a secure constrained PRF with respect to $\mathcal{C}$ if for all PPT adversaries A $\text{Adv}_A^F(\lambda)$ is negligible in λ .

In the above definition the challenge query oracle may be queried multiple times on different points, and either all the challenge responses are correct PRF evaluations or they are all random points. We remark that Boneh-Waters [BW13] argued that such a definition was equivalent (via a hybrid argument) to a definition where the adversary may only submit one challenge query. In the next section, the adversary may submit only one challenge.

3.2 Puncturable Pseudorandom Functions

In this work, we will focus on puncturable pseudorandom functions, a subset of constrained PRFs where the “circuits” output 1 on every input except one “punctured” point. A PRF $F : \mathcal{K} \times \mathcal{X} \rightarrow \mathcal{Y}$ is a puncturable pseudorandom function if there is an additional key space $\mathcal{K}_p$ and three polynomial time algorithms $F.\text{setup}$, $F.\text{eval}$ and $F.\text{puncture}$ as follows:

- $F.\text{setup}(1^\lambda)$ is a randomized algorithm that takes the security parameter λ as input and outputs a description of the key space $\mathcal{K}$, the punctured key space $\mathcal{K}_p$ and the PRF F .
- $F.\text{puncture}(k, x)$ is a randomized algorithm that takes as input a PRF key $k \in \mathcal{K}$ and $x \in \mathcal{X}$, and outputs a key $k_x \in \mathcal{K}_p$.
- $F.\text{eval}(k_x, x')$ is a deterministic algorithm that takes as input a punctured key $k_x \in \mathcal{K}_p$ and $x' \in \mathcal{X}$. Let $k \in \mathcal{K}, x \in \mathcal{X}$ and $k_x \leftarrow F.\text{puncture}(k, x)$. For correctness, we need the following property:

$$F.\text{eval}(k_x, x') = \begin{cases} F(k, x') & \text{if } x \neq x' \\ \perp & \text{otherwise} \end{cases}$$

The security game between the challenger and the adversary A consists of the following four phases.

Setup Phase The challenger chooses uniformly at random a PRF key $k \leftarrow \mathcal{K}$ and a bit $b \leftarrow \{0, 1\}$.

Evaluation Query Phase A queries for polynomially many evaluations. For each evaluation query x , the challenger sends $F(k, x)$ to A .

Challenge Phase A chooses a challenge $x^* \in \mathcal{X}$. The challenger computes $k_{x^*} \leftarrow F.\text{puncture}(k, x^*)$. If $b = 0$, the challenger outputs k_{x^*} and $F(k, x^*)$. Else, the challenger outputs k_{x^*} and $y \leftarrow \mathcal{Y}$ chosen uniformly at random.

Guess A outputs a guess b' of b .

Let $E \subset \mathcal{X}$ be the set of evaluation queries. A wins if $b = b'$ and $x^* \notin E$. The advantage of A is defined to be $\text{Adv}_A^F(\lambda) = \Pr[A \text{ wins}]$.

Definition 3.2. The PRF F is a secure puncturable PRF if for all probabilistic polynomial time adversaries A $\text{Adv}_A^F(\lambda)$ is negligible in λ .

Recall that since the adversary can compute any evaluation of F except the punctured point using the key given during the Challenge Phase, access to the Evaluation Query oracle is redundant and can be removed after the Challenge Phase.

3.2.1 t -Puncturable Pseudorandom Functions

The notion of puncturable PRFs can be naturally extended to that of t -puncturable PRFs, where it is possible to derive a key punctured at any set S of size at most t .

Let $t(\cdot)$ be a polynomial. A PRF $F_t : \mathcal{K} \times \mathcal{X} \rightarrow \mathcal{Y}$ is a t -puncturable pseudorandom function if there is an additional key space $\mathcal{K}_p$ and three polynomial time algorithms $F_t.\text{setup}$, $F_t.\text{eval}$ and $F_t.\text{puncture}$ defined as follows.

- $F_t.\text{setup}(1^\lambda)$ is a randomized algorithm that takes the security parameter λ as input and outputs a description of the key space $\mathcal{K}$, the punctured key space $\mathcal{K}_p$ and the PRF F_t .
- $F_t.\text{puncture}(k, S)$ is a randomized algorithm that takes as input a PRF key $k \in \mathcal{K}$ and $S \subset \mathcal{X}$, $|S| \leq t(\lambda)$, and outputs a t -punctured key $K_S \in \mathcal{K}_p$.
- $F_t.\text{eval}(k_S, x')$ is a deterministic algorithm that takes as input a t -punctured key $k_S \in \mathcal{K}_p$ and $x' \in \mathcal{X}$. Let $k \in \mathcal{K}$, $S \subset \mathcal{X}$ and $k_S \leftarrow F_t.\text{puncture}(k, S)$. For correctness, we need the following property:

$$F_t.\text{eval}(k_S, x') = \begin{cases} F_t(k, x') & \text{if } x' \notin S \\ \perp & \text{otherwise} \end{cases}$$

The security game between the challenger and adversary is similar to the security game for puncturable PRFs. However, in this case, the adversary is allowed to make multiple challenge queries (as in the security game for constrained PRFs). The game consists of the following three phases.

Setup Phase The challenger chooses a random key $k \leftarrow \mathcal{K}$ and a random bit $b \leftarrow \{0, 1\}$.

Query Phase In this phase, A is allowed to ask for the following queries:

- **Evaluation Query** A sends $x \in \mathcal{X}$, and receives $F_t(k, x)$.
- **Key Query** A sends a set $S \subset \mathcal{X}$, and receives $F_t.\text{puncture}(k, S)$.
- **Challenge Query** A sends $x \in \mathcal{X}$ as a challenge query. If $b = 0$, the challenger outputs $F_t(k, x)$. Else, the challenger outputs a random element $y \leftarrow \mathcal{Y}$.

Guess A outputs a guess b' of b .

Let $x_1, \dots, x_{q_1} \in \mathcal{X}$ be the evaluation queries, $S_1, \dots, S_{q_2} \subset \mathcal{X}$ be the t -punctured key queries and $x_1^*, \dots, x_s^*$ be the challenge queries. A wins if $\forall i \leq q_1, j \leq s, x_i \neq x_j^*, \forall i \leq q_2, j \leq s, x_j^* \in S_i$ and $b' = b$. The advantage of A is defined to be $\text{Adv}_A^{F_t}(\lambda) = \Pr[A \text{ wins}]$.

Definition 3.3. The PRF F_t is a secure t -puncturable PRF if for all PPT adversaries $\mathcal{A}$ $\text{Adv}_A^{F_t}(\lambda)$ is negligible in λ .

4 Construction

We now describe our puncturable PRF family. It consists of the PRF $F : \mathcal{K} \times \mathcal{X} \rightarrow \mathcal{Y}$ and the three algorithms $F.\text{setup}, F.\text{puncture}$ and $F.\text{eval}$. The input domain is $\mathcal{X} = \{0, 1\}^\ell$, where $\ell = \ell(\lambda)$. We define the key space $\mathcal{K}$ and range space $\mathcal{Y}$ as part of the setup algorithm described next.

F.setup(1^λ) $F.\text{setup}$, on input 1^λ , first runs $\mathcal{G}$ to compute $(N, p, q, \mathbb{G}, \mathbb{G}_p, \mathbb{G}_q, g_1, g_2) \leftarrow \mathcal{G}(1^\lambda)$. Let n, θ be polynomials such that there exists a θ -admissible hash function h mapping $\ell(\lambda)$ bits to $n(\lambda)$ bits. For simplicity of notation, we will drop the dependence of ℓ and n on λ .

The key space is $\mathcal{K} = \mathbb{G} \times (\mathbb{Z}_N^2)^n$ and the range is $\mathcal{Y} = \mathbb{G}$. The setup algorithm chooses $v \in \mathbb{G}$, $d_{i,b} \in \mathbb{Z}_N$, for $i = 1$ to n and $b \in \{0, 1\}$, and sets $k = (v, ((d_{1,0}, d_{1,1}), \dots, (d_{n,0}, d_{n,1})))$.

The PRF F for key k on input x is then computed as follows. Let $k = (v, ((d_{1,0}, d_{1,1}), \dots, (d_{n,0}, d_{n,1}))) \in \mathbb{G} \times (\mathbb{Z}_N^2)^n$ and $h(x) = (b_1, \dots, b_n)$, where $b_i \in \{0, 1\}$. Then,

$$F(k, x) = v^{\prod_{j=1}^n d_{j,b_j}}.$$

F.puncture(k, x') $F.\text{puncture}$ computes an obfuscation of the program $\text{PuncturedKey}_{k,x'}$ defined in Figure ??; that is, $K_{x'} \leftarrow i\mathcal{O}(\lambda, \text{PuncturedKey}_{k,x'})$. The program $\text{PuncturedKey}_{k,x'}$ is padded to be of appropriate size.⁵

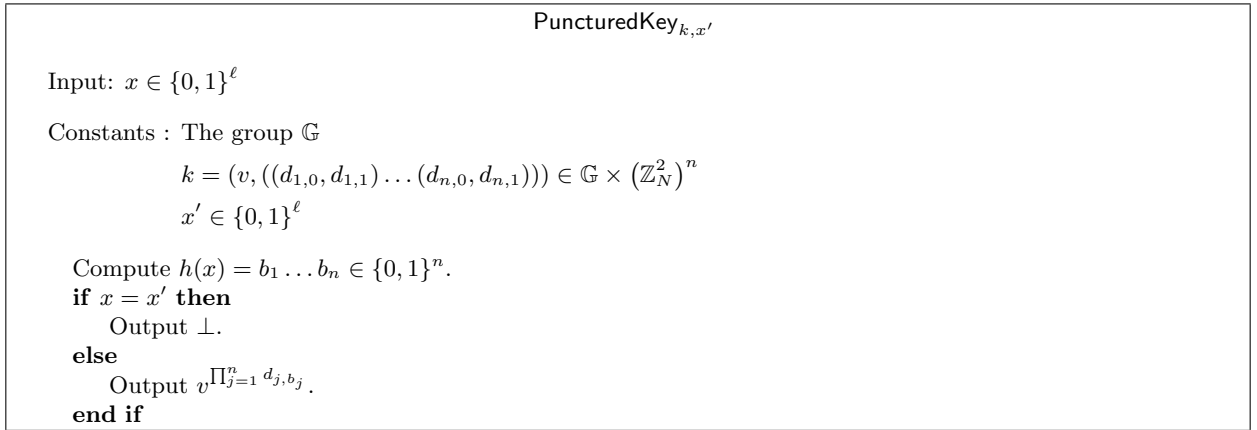


Figure 1: Program PuncturedKey

F.eval($K_{x'}, x$) The punctured key $K_{x'}$ is a program that takes an ℓ -bit input. We define

$$F.\text{eval}(K_{x'}, x) = K_{x'}(x).$$

⁵Looking ahead, in the proof of security, the program $\text{PuncturedKey}_{k,x'}$ will be replaced by $\text{PuncturedKey}'_{V,w,D,u,x'}$, $\text{PuncturedKeyAlt}_{u,k,k',x'}$ and $\text{PuncturedKeyAlt}'_{u,W,E,k,x'}$ in subsequent hybrids. Since this transformation relies on $i\mathcal{O}$ being secure, we need that all programs have same size. Hence, all programs are padded appropriately to ensure that they have the same size.

4.1 Proof of Security

We will now prove that our construction is a secure puncturable PRF as defined in Definition 3.2. Specifically, the claim we show is:

Theorem 4.1 (Main Theorem). Assuming $i\mathcal{O}$ is a secure indistinguishability obfuscator and the Subgroup Hiding Assumption holds for groups output by $\mathcal{G}$, the PRF F defined above, together with algorithms $F.\text{setup}$, $F.\text{puncture}$ and $F.\text{eval}$, is a secure punctured pseudorandom function as defined in Definition 3.2.

Proof. In order to prove this, we define the following sequence of games. Assume the adversary $\mathcal{A}$ makes $q = q(\lambda)$ evaluation queries (where $q(\cdot)$ is a polynomial) before sending the challenge input.

4.1.1 Sequence of Games

We underline the primary changes from one game to the next.

Game 0 This game is the original security game from Definition 3.2 between the challenger and $\mathcal{A}$ instantiated by the construction under analysis. Here the challenger first chooses a random PRF key, then $\mathcal{A}$ makes evaluation queries and finally sends the challenge input. The challenger responds by sending a key punctured at the challenge input, and either a PRF evaluation at the challenged point or a random value.

1. Let $(N, p, q, \mathbb{G}, \mathbb{G}_p, \mathbb{G}_q, g_1, g_2) \leftarrow \mathcal{G}(1^\lambda)$. Choose $v \in \mathbb{G}$, $d_{i,b} \in \mathbb{Z}_N$, for $i = 1$ to n and $b \in \{0, 1\}$, and set $k = (v, ((d_{1,0}, d_{1,1}), \dots, (d_{n,0}, d_{n,1})))$.
2. On any evaluation query $x_i \in \{0, 1\}^\ell$, compute $h(x_i) = b_1^i \dots b_n^i$ and output $v^{\prod_{j=1}^n d_{j,b_j^i}}$.
3. $\mathcal{A}$ sends challenge input x^* such that $x^* \neq x_i$ for all $i \leq q$. Compute $K_{x^*} \leftarrow i\mathcal{O}(\lambda, \text{PuncturedKey}_{k,x^*})$ and $h(x^*) = b_1^* \dots b_n^*$. Let $y_0 = v^{\prod_{j=1}^n d_{j,b_j^*}}$ and $y_1 \leftarrow \mathbb{G}$.
4. Flip coin $\beta \leftarrow \{0, 1\}$. Output (K_{x^*}, y_β) .
5. $\mathcal{A}$ outputs β' and wins if $\beta = \beta'$.

Game 1 This game is the same as the previous one, except that we simulate a partitioning game while the adversary operates and if an undesirable partition arises, we abort the game and decide whether or not the adversary “wins” by a coin flip. This partitioning game works as follows: the challenger samples $u \in \{0, 1, \perp\}^n$ using AdmSample and aborts if either there exists an evaluation query x such that $P_u(x) = 0$ or the challenge query x^* satisfies $P_u(x^*) = 1$.

1. Let $(N, p, q, \mathbb{G}, \mathbb{G}_p, \mathbb{G}_q, g_1, g_2) \leftarrow \mathcal{G}(1^\lambda)$. Choose $v \in \mathbb{G}$, $d_{i,b} \in \mathbb{Z}_N$, for $i = 1$ to n and $b \in \{0, 1\}$, and set $k = (v, ((d_{1,0}, d_{1,1}), \dots, (d_{n,0}, d_{n,1})))$.
Choose $u \leftarrow \text{AdmSample}(1^\lambda, q)$ and let $S_u = \{x : P_u(x) = 1\}$ (recall $P_u(x) = 0$ if $h(x)_j \neq u_j \forall 1 \leq j \leq n$).
2. On any evaluation query $x_i \in \{0, 1\}^\ell$, check if $P_u(x_i) = 1$.
If not, flip a coin $\gamma_i \leftarrow \{0, 1\}$ and abort. $\mathcal{A}$ wins if $\gamma_i = 1$.
Else compute $h(x_i) = b_1^i \dots b_n^i$ and output $v^{\prod_{j=1}^n d_{j,b_j^i}}$.
3. $\mathcal{A}$ sends challenge input x^* such that $x^* \neq x_i$ for all $i \leq q$. Check if $P_u(x^*) = 0$.
If not, flip a coin $\gamma^* \leftarrow \{0, 1\}$ and abort. $\mathcal{A}$ wins if $\gamma^* = 1$.
Else compute $K_{x^*} \leftarrow i\mathcal{O}(\lambda, \text{PuncturedKey}_{k,x^*})$ and $h(x^*) = b_1^* \dots b_n^*$. Let $y_0 = v^{\prod_{j=1}^n d_{j,b_j^*}}$ and $y_1 \leftarrow \mathbb{G}$.
4. Flip coin $\beta \leftarrow \{0, 1\}$. Output (K_{x^*}, y_β) .
5. $\mathcal{A}$ outputs β' and wins if $\beta = \beta'$.

Game 2 In this game, the challenger modifies the punctured key and outputs an obfuscation of PuncturedKeyAlt defined in Figure ?? . On inputs x such that $P_u(x) = 1$, the altered punctured key uses the same master key k as before. However, if $P_u(x) = 0$, the altered punctured key uses a different master key k' that is randomly chosen from the key space.

1. Let $(N, p, q, \mathbb{G}, \mathbb{G}_p, \mathbb{G}_q, g_1, g_2) \leftarrow \mathcal{G}(1^\lambda)$. Choose $v \in \mathbb{G}, d_{i,b} \in \mathbb{Z}_N$, for $i = 1$ to n and $b \in \{0, 1\}$, and set $k = (v, ((d_{1,0}, d_{1,1}), \dots, (d_{n,0}, d_{n,1})))$.
Choose $u \leftarrow \text{AdmSample}(1^\lambda, q)$.
2. On any evaluation query $x_i \in \{0, 1\}^\ell$, check if $P_u(x_i) = 1$.
If not, flip a coin $\gamma_i \leftarrow \{0, 1\}$ and abort. $\mathcal{A}$ wins if $\gamma_i = 1$.
Else compute $h(x_i) = b_1^i \dots b_n^i$ and output $v^{\prod_{j=1}^n d_{j,b_j^i}}$.
3. $\mathcal{A}$ sends challenge input x^* such that $x^* \neq x_i$ for all $i \leq q$. Check if $P_u(x^*) = 0$.
If not, flip a coin $\gamma^* \leftarrow \{0, 1\}$ and abort. $\mathcal{A}$ wins if $\gamma^* = 1$.
Else choose $w \in \mathbb{G}, e_{i,b} \in \mathbb{Z}_N$, for $i = 1$ to n and $b \in \{0, 1\}$, and set $k' = (w, ((e_{1,0}, e_{1,1}), \dots, (e_{n,0}, e_{n,1})))$.
Compute $K_{x^*} \leftarrow i\mathcal{O}(\lambda, \text{PuncturedKeyAlt}_{u,k,k',x^*})$ and $h(x^*) = b_1^* \dots b_n^*$. Let $y_0 = w^{\prod_{j=1}^n e_{j,b_j^*}}$ and $y_1 \leftarrow \mathbb{G}$.
4. Flip coin $\beta \leftarrow \{0, 1\}$. Output (K_{x^*}, y_β) .
5. $\mathcal{A}$ outputs β' and wins if $\beta = \beta'$.

PuncturedKeyAlt_{u,k,k',x'}

Input: $x \in \{0, 1\}^\ell$

Constants : The group $\mathbb{G}$

$$k = (v, ((d_{1,0}, d_{1,1}), \dots, (d_{n,0}, d_{n,1}))) \in \mathbb{G} \times (\mathbb{Z}_N^2)^n$$

$$k' = (w, ((e_{1,0}, e_{1,1}), \dots, (e_{n,0}, e_{n,1}))) \in \mathbb{G} \times (\mathbb{Z}_N^2)^n$$

$$x' \in \{0, 1\}^\ell, u \in \{0, 1, \perp\}^n$$

Compute $h(x) = b_1 \dots b_n$.

if $x = x'$ **then**

Output $\perp$.

else if $P_u(x) = 0$ **then**

output $w^{\prod_{j=1}^n e_{j,b_j}}$.

else

Output $v^{\prod_{j=1}^n d_{j,b_j}}$.

end if

Figure 2: Program PuncturedKeyAlt

Game 3 In this game, the challenger changes how the master key k' is chosen so that some elements contain an a -factor, for use on inputs x where $P_u(x) = 0$.

1. Let $(N, p, q, \mathbb{G}, \mathbb{G}_p, \mathbb{G}_q, g_1, g_2) \leftarrow \mathcal{G}(1^\lambda)$. Choose $v \in \mathbb{G}, d_{i,b} \in \mathbb{Z}_N$, for $i = 1$ to n and $b \in \{0, 1\}$, and set $k = (v, ((d_{1,0}, d_{1,1}), \dots, (d_{n,0}, d_{n,1})))$.
Choose $u \leftarrow \text{AdmSample}(1^\lambda, q)$.
2. On any evaluation query $x_i \in \{0, 1\}^\ell$, check if $P_u(x_i) = 1$.
If not, flip a coin $\gamma_i \leftarrow \{0, 1\}$ and abort. $\mathcal{A}$ wins if $\gamma_i = 1$.
Else compute $h(x_i) = b_1^i \dots b_n^i$ and output $v^{\prod_{j=1}^n d_{j,b_j^i}}$.
3. $\mathcal{A}$ sends challenge input x^* such that $x^* \neq x_i$ for all $i \leq q$. Check if $P_u(x^*) = 0$.
If not, flip a coin $\gamma^* \leftarrow \{0, 1\}$ and abort. $\mathcal{A}$ wins if $\gamma^* = 1$.
Else choose $w \leftarrow \mathbb{G}, a \leftarrow \mathbb{Z}_N^*$ and $e'_{i,b} \leftarrow \mathbb{Z}_N$. Let $e_{i,b} = e'_{i,b} \cdot a$ if $h(x^*)_i = b$, else $e_{i,b} = e'_{i,b}$.
Let $k' = (w, ((e_{1,0}, e_{1,1}), \dots, (e_{n,0}, e_{n,1})))$ and $K_{x^*} \leftarrow i\mathcal{O}(\lambda, \text{PuncturedKeyAlt}_{u,k,k',x^*})$.
Let $h(x^*) = b_1^* \dots b_n^*$ and $y_0 = w^{\prod_{j=1}^n e_{j,b_j^*}}$ and $y_1 \leftarrow \mathbb{G}$.
4. Flip coin $\beta \leftarrow \{0, 1\}$. Output (K_{x^*}, y_β) .
5. $\mathcal{A}$ outputs β' and wins if $\beta = \beta'$.

Game 4 This game is the same as the previous one, except that the altered punctured program contains the constants $\{w^{a^i}\}_{i=0}^n$ hardwired. These terms are used to compute the output of the punctured program. The punctured key is an obfuscation of $\text{PuncturedKeyAlt}'$ defined in Figure ??.

1. Let $(N, p, q, \mathbb{G}, \mathbb{G}_p, \mathbb{G}_q, g_1, g_2) \leftarrow \mathcal{G}(1^\lambda)$. Choose $v \in \mathbb{G}, d_{i,b} \in \mathbb{Z}_N$, for $i = 1$ to n and $b \in \{0, 1\}$, and set $k = (v, ((d_{1,0}, d_{1,1}), \dots, (d_{n,0}, d_{n,1})))$.
Choose $u \leftarrow \text{AdmSample}(1^\lambda, q)$.
2. On any evaluation query $x_i \in \{0, 1\}^\ell$, check if $P_u(x_i) = 1$.
If not, flip a coin $\gamma_i \leftarrow \{0, 1\}$ and abort. $\mathcal{A}$ wins if $\gamma_i = 1$.
Else compute $h(x_i) = b_1^i \dots b_n^i$ and output $v^{\prod_j d_{j,b_j^i}}$.
3. $\mathcal{A}$ sends challenge input x^* such that $x^* \neq x_i$ for all $i \leq q$. Check if $P_u(x^*) = 0$.
If not, flip a coin $\gamma^* \leftarrow \{0, 1\}$ and abort. $\mathcal{A}$ wins if $\gamma^* = 1$.
Else choose $w \leftarrow \mathbb{G}, a \leftarrow \mathbb{Z}_N^*$ and $e'_{i,b} \leftarrow \mathbb{Z}_N$.
Let $W = (w, w^a, \dots, w^{a^{n-1}}), E = ((e'_{1,0}, e'_{1,1}), \dots, (e'_{n,0}, e'_{n,1}))$ and $K''_{x^*} \leftarrow i\mathcal{O}(\lambda, \text{PuncturedKeyAlt}'_{u,W,E,k,x^*})$.
Let $h(x^*) = b_1^* \dots b_n^*, y_0 = (w^{a^n})^{\prod_j e'_{j,b_j^*}}$ and $y_1 \leftarrow \mathbb{G}$.
4. Flip coin $\beta \leftarrow \{0, 1\}$. Output (K''_{x^*}, y_β) .
5. $\mathcal{A}$ outputs β' and wins if $\beta = \beta'$.

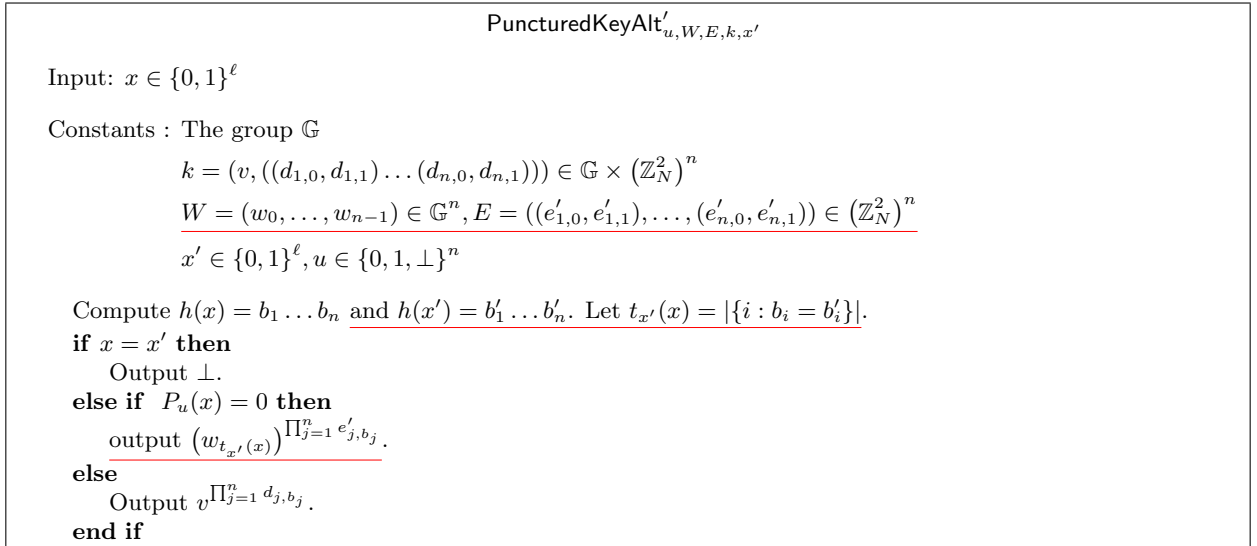


Figure 3: Program PuncturedKeyAlt'

Game 5 In this game, we replace the term w^{a^n} with a random element from $\mathbb{G}$. Hence, both y_0 and y_1 are random elements of $\mathbb{G}$, thereby ensuring that any adversary has zero advantage in this game.

1. Let $(N, p, q, \mathbb{G}, \mathbb{G}_p, \mathbb{G}_q, g_1, g_2) \leftarrow \mathcal{G}(1^\lambda)$. Choose $v \in \mathbb{G}, d_{i,b} \in \mathbb{Z}_N$, for $i = 1$ to n and $b \in \{0, 1\}$, and set $k = (v, ((d_{1,0}, d_{1,1}), \dots, (d_{n,0}, d_{n,1})))$.
Choose $u \leftarrow \text{AdmSample}(1^\lambda, q)$.
2. On any evaluation query $x_i \in \{0, 1\}^\ell$, check if $P_u(x_i) = 1$.
If not, flip a coin $\gamma_i \leftarrow \{0, 1\}$ and abort. $\mathcal{A}$ wins if $\gamma_i = 1$.
Else compute $h(x_i) = b_1^i \dots b_n^i$ and output $v^{\prod_{j=1}^n d_{j,b_j^i}}$.
3. $\mathcal{A}$ sends challenge input x^* such that $x^* \neq x_i$ for all $i \leq q$. Check if $P_u(x^*) = 0$.
If not, flip a coin $\gamma^* \leftarrow \{0, 1\}$ and abort. $\mathcal{A}$ wins if $\gamma^* = 1$.

Else choose $w \leftarrow \mathbb{G}$, $a \leftarrow \mathbb{Z}_N^*$, and $e'_{i,b} \leftarrow \mathbb{Z}_N$. Let $W = (w, w^a, \dots, w^{a^{n-1}})$, $E = ((e'_{1,0}, e'_{1,1}), \dots, (e'_{n,0}, e'_{n,1}))$ and $K_{x^*} \leftarrow i\mathcal{O}(\lambda, \text{PuncturedKeyAlt}'_{u,W,E,k,x^*})$.

Let $h(x^*) = b_1^* \dots b_n^*$. Choose $T \leftarrow \mathbb{G}$ and let $y_0 = (T)^{\prod_{j=1}^n e'_{j,b_j^*}}$ and $y_1 \leftarrow \mathbb{G}$.

4. Flip coin $\beta \leftarrow \{0, 1\}$. Output (K_{x^*}, y_β) .

5. $\mathcal{A}$ outputs β' and wins if $\beta = \beta'$.

4.1.2 Adversary's Advantage in these Games

Let $\text{Adv}_{\mathcal{A}}^i$ denote the advantage of adversary $\mathcal{A}$ in Game i . We will now show that if an adversary $\mathcal{A}$ has non-negligible advantage in Game i , then $\mathcal{A}$ has non-negligible advantage in Game $i + 1$. Finally, we show that $\mathcal{A}$ has advantage 0 in Game 5.

Claim 4.1. For any adversary $\mathcal{A}$, $\text{Adv}_{\mathcal{A}}^1 \geq \text{Adv}_{\mathcal{A}}^0 / \theta(q)$.

Proof. This claim follows from the θ -admissibility of the hash function h . Recall h is θ -admissible if for all $x_1, \dots, x_q, x^*$, $\Pr[\forall i, P_u(x_i) = 1 \wedge P_u(x^*) = 0] \geq 1/\theta(q)$, where the probability is only over the choice of $u \leftarrow \text{AdmSample}(1^\lambda, q)$. Therefore, if $\mathcal{A}$ wins with advantage ϵ in Game 0, then $\mathcal{A}$ wins with advantage at least $\epsilon/\theta(q)$ in Game 1. $\blacksquare$

Claim 4.2. Assuming $i\mathcal{O}$ is a secure indistinguishability obfuscator and the Subgroup Hiding Assumption holds, for any PPT adversary $\mathcal{A}$,

$$\text{Adv}_{\mathcal{A}}^1 - \text{Adv}_{\mathcal{A}}^2 \leq \text{negl}(\lambda).$$

The proof of this claim involves multiple intermediate experiments and can be found in Appendix A.

Claim 4.3. For any PPT adversary $\mathcal{A}$, $\text{Adv}_{\mathcal{A}}^3 = \text{Adv}_{\mathcal{A}}^2$.

Proof. Game 2 and Game 3 are identical, except for the manner in which the constants $e_{i,b}$ are chosen. In Game 2, $e_{i,b} \leftarrow \mathbb{Z}_N$, while in Game 3, the challenger first chooses $e'_{i,b} \leftarrow \mathbb{Z}_N$, $a \leftarrow \mathbb{Z}_N^*$, and sets $e_{i,b} = e'_{i,b} \cdot a$ if $h(x)_i = b$, else sets $e_{i,b} = e'_{i,b}$. Since $a \in \mathbb{Z}_N^*$ (and therefore is invertible), $e'_{i,b} \cdot a$ is also a uniformly random element in $\mathbb{Z}_N$ if $e'_{i,b}$ is. Hence the two experiments are identical. $\blacksquare$

Claim 4.4. If there exists a PPT adversary $\mathcal{A}$ such that $\text{Adv}_{\mathcal{A}}^3 - \text{Adv}_{\mathcal{A}}^4$ is non-negligible in λ , then there exists a PPT distinguisher $\mathcal{B}$ that breaks the security of $i\mathcal{O}$ with advantage non-negligible in λ .

Proof. Suppose there exists a PPT adversary $\mathcal{A}$ such that $\text{Adv}_{\mathcal{A}}^3 - \text{Adv}_{\mathcal{A}}^4 = \epsilon$. We will construct a PPT algorithm $\mathcal{B}$ that breaks the security of $i\mathcal{O}$ with advantage ϵ by interacting with $\mathcal{A}$. $\mathcal{B}$ first sets up the parameters, including u and k , and answers the evaluation queries of $\mathcal{A}$ exactly as in steps 1 and 2 of Game 3, which are identical to steps 1 and 2 of Game 4. When $\mathcal{A}$ sends $\mathcal{B}$ a challenge input x^* , $\mathcal{B}$ checks that $P_u(x^*) = 0$ and if not aborts (identical in both games).

Next $\mathcal{B}$ chooses further values to construct the circuits: $w \leftarrow \mathbb{G}$, $a \leftarrow \mathbb{Z}_N^*$ and $e'_{i,b} \leftarrow \mathbb{Z}_N$. Let $e_{i,b} = e'_{i,b} \cdot a$ if $h(x^*)_i = b$, else $e_{i,b} = e'_{i,b}$. Let $k' = (w, ((e_{1,0}, e_{1,1}), \dots, (e_{n,0}, e_{n,1})))$, $W = (w, w^a, \dots, w^{a^{n-1}})$ and $E = ((e'_{1,0}, e'_{1,1}), \dots, (e'_{n,0}, e'_{n,1}))$.

$\mathcal{B}$ constructs circuit $C_0 = \text{PuncturedKeyAlt}_{u,k,k',x^*}$ and circuit $C_1 = \text{PuncturedKeyAlt}'_{u,W,E,k,x^*}$, and sends C_0, C_1 to the $i\mathcal{O}$ challenger. $\mathcal{B}$ receives $K_{x^*} \leftarrow i\mathcal{O}(C_b)$ from the challenger. It computes $h(x^*) = b_1^* \dots b_n^*$, $y_0 = w^{\prod_{j=1}^n e'_{j,b_j^*}}$, $y \leftarrow \mathbb{G}$, $\beta \leftarrow \{0, 1\}$, sends (K_{x^*}, y_β) to $\mathcal{A}$ and receives β' in response. If $\beta = \beta'$, $\mathcal{B}$ outputs 0, else it outputs 1.

We will now prove that the circuits C_0 and C_1 have identical functionality. Consider any ℓ bit string x , and let $h(x) = b_1 \dots b_n$. Recall $t_{x^*}(x) = |\{i : b_i = b_i^*\}|$.

For any $x \in \{0, 1\}^\ell$ such that $x = x^*$, both circuits output $\perp$.

For any $x \in \{0, 1\}^\ell$ such that $x \neq x^*$ and $P_u(x) = 1$, both circuits output $v^{\prod_{j=1}^n d_{j,b_j}}$.

For any $x \in \{0, 1\}^\ell$ such that $x \neq x^*$ and $P_u(x) = 0$, we have

$$C_0(x) = \text{PuncturedKeyAlt}_{u,k,k',x^*}(x) = w^{\prod_{j=1}^n e_{j,b_j}} = w^{a^{t_{x^*}(x)} \prod_{j=1}^n e'_{j,b_j}} = \\ (w_{t_{x^*}(x)})^{\prod_{j=1}^n e'_{j,b_j}} = \text{PuncturedKeyAlt}'_{u,W,E,k,x^*}(x) = C_1(x).$$

Since C_0 and C_1 have identical functionality, $\Pr[\mathcal{B} \text{ wins}] = \Pr[\mathcal{A} \text{ wins in Game 3}] - \Pr[\mathcal{A} \text{ wins in Game 4}]$. If $\text{Adv}_{\mathcal{A}}^3 - \text{Adv}_{\mathcal{A}}^4 = \epsilon$, then $\mathcal{B}$ wins the $i\mathcal{O}$ security game with advantage ϵ . ■

Claim 4.5. If there exists a PPT adversary $\mathcal{A}$ such that $\text{Adv}_{\mathcal{A}}^4 - \text{Adv}_{\mathcal{A}}^5$ is non-negligible in λ , then there exists a PPT adversary $\mathcal{B}$ that breaks Assumption 2 with advantage non-negligible in λ .

Proof. Suppose there exists an adversary $\mathcal{A}$ such that $\text{Adv}_{\mathcal{A}}^4 - \text{Adv}_{\mathcal{A}}^5 = \epsilon$, then we can build an adversary that breaks Assumption 2 with advantage ϵ . The games are identical except that Game 5 replaces the term w^{a^n} with a random element of $\mathbb{G}$. On input an Assumption 2 instance $(N, \mathbb{G}, \mathbb{G}_p, \mathbb{G}_q, g_1, g_2, w, w^a, \dots, w^{a^{n-1}})$ together with challenge value T (which is either w^{a^n} or a random element in $\mathbb{G}$), use these parameters as in Game 5 with $\mathcal{A}$. If $\mathcal{A}$ guesses it was in Game 4, guess that $T = w^{a^n}$, else guess that T was random. ■

Observation 4.1. For any adversary $\mathcal{A}$, $\text{Adv}_{\mathcal{A}}^5 = 0$.

Proof. If the challenger aborts either during the evaluation or challenge phase, then $\mathcal{A}$ has 0 advantage, since $\mathcal{A}$ wins with probability 1/2. If the challenger does not abort during both these phases, then $\mathcal{A}$ receives (K_{x^*}, y_β) , and $\mathcal{A}$ must guess β . However, both y_0 and y_1 are uniformly random elements in $\mathbb{G}$, and therefore, $\text{Adv}_{\mathcal{A}}^5 = 0$. ■

4.1.3 Conclusion of the Main Proof

Given Claims 4.1-4.5 and Observation 4.1, we can conclude that if $i\mathcal{O}$ is a secure indistinguishability obfuscator and Assumption 1 holds (recall Appendix A shows that Assumption 1 implies Assumption 2), then any PPT adversary $\mathcal{A}$ has negligible advantage in the puncturable PRF security game (i.e., Game 0). ■

5 Construction of t -Puncturable PRFs from Puncturable PRFs

In this section, we present our construction of t -puncturable PRFs from puncturable PRFs and indistinguishability obfuscation. Let $F : \mathcal{K} \times \mathcal{X} \rightarrow \mathcal{Y}$ be a puncturable PRF, and $F.\text{setup}$, $F.\text{puncture}$, $F.\text{eval}$ the corresponding setup, puncturing and evaluation algorithms. We now describe our t -puncturable PRF F_t , and the corresponding algorithms $F_t.\text{setup}$, $F_t.\text{puncture}$ and $F_t.\text{eval}$.

5.1 Construction

$F_t.\text{setup}(1^\lambda)$ $F_t.\text{setup}$ is the same as $F.\text{setup}$.

$F_t.\text{puncture}(\mathbf{k}, \mathbf{S})$ $F_t.\text{puncture}(k, S)$ computes an obfuscation of the program $\text{PuncturedKey}_{k,S}^t$ defined in Figure ??; that is, $K_S \leftarrow i\mathcal{O}(\lambda, \text{PuncturedKey}_{k,S}^t)$. As before, the program $\text{PuncturedKey}_{k,S}^t$ is padded to be of appropriate size.

$F_t.\text{eval}(\mathbf{K}_S, \mathbf{x})$ The punctured key K_S is a program that takes an input in $\mathcal{X}$. We define

$$F_t.\text{eval}(K_S, x) = K_S(x).$$

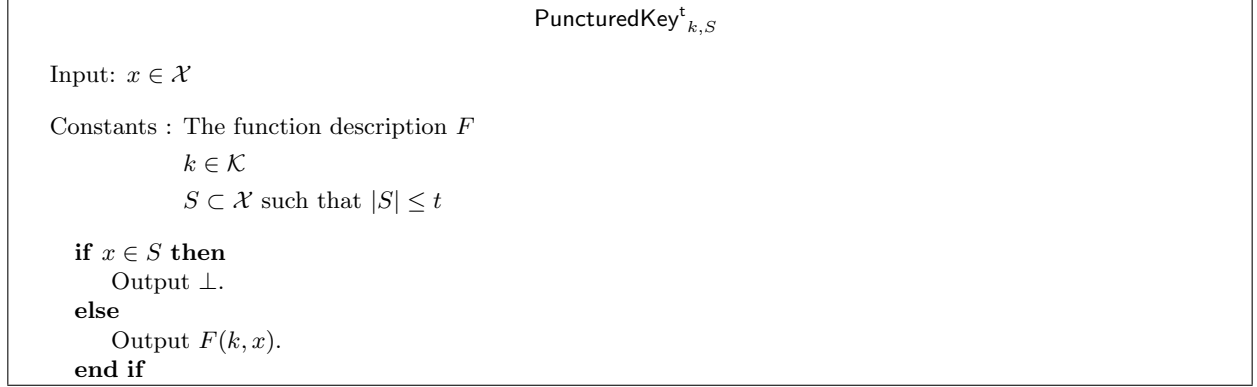


Figure 4: Program PuncturedKey^t

5.2 Proof of Security

We will now prove that the above construction is a secure t -puncturable PRF as defined in Definition 3.3.

Theorem 5.1. Assuming $i\mathcal{O}$ is a secure indistinguishability obfuscator and F , together with $F.\text{setup}$, $F.\text{puncture}$ and $F.\text{eval}$ is a secure puncturable PRF, the PRF F_t defined above, together with $F_t.\text{setup}$, $F_t.\text{puncture}$ and $F_t.\text{eval}$, is a secure t -puncturable PRF.

For simplicity, we will assume that the adversary makes q_1 evaluation queries, q_2 punctured key queries and 1 challenge query. As shown by [BW13], this can easily be extended to the general case of multiple challenge queries via a hybrid argument. We will first define the intermediate hybrid experiments.

Game 0 This game is the original security game between the challenger and adversary $\mathcal{A}$, where the challenger first chooses a PRF key, then $\mathcal{A}$ makes evaluation/ t -punctured key queries and finally sends the challenge input. The challenger responds with either the PRF evaluation at challenge input, or sends a random element of the range space.

1. Choose a key $k \leftarrow \mathcal{K}$.
2. $\mathcal{A}$ makes evaluation/ t -punctured key queries.
 - (a) If $\mathcal{A}$ sends an evaluation query x_i , then output $F(k, x_i)$.
 - (b) If $\mathcal{A}$ sends a t -punctured key query for set S_j , output $K_{S_j} \leftarrow i\mathcal{O}(\lambda, \text{PuncturedKey}^t_{k, S_j})$.
3. $\mathcal{A}$ sends challenge query x^* such that $x^* \neq x_i \forall i \leq q_1$ and $x^* \in S_j \forall j \leq q_2$. Choose $\beta \leftarrow \{0, 1\}$. If $\beta = 0$, output $y = F(k, x^*)$, else output $y \leftarrow \mathcal{Y}$.
4. $\mathcal{A}$ sends β' and wins if $\beta = \beta'$.

Game 1 This game is the same as the previous one, except that the challenger introduces an abort condition. When the first t -punctured key query S_1 is made, the challenger guesses the challenge query $\tilde{x} \leftarrow S_1$. The challenger aborts if any of the evaluation queries are $\tilde{x}$, if any of the future t -punctured key queries does not contain $\tilde{x}$ or if the challenge query $x^* \neq \tilde{x}$.

1. Choose a key $k \leftarrow \mathcal{K}$.
2. $\mathcal{A}$ makes evaluation/ t -punctured key queries.

Let S_1 be the first t -punctured key query. Choose $\tilde{x} \leftarrow S_1$ and output $K_{S_1} \leftarrow i\mathcal{O}(\lambda, \text{PuncturedKey}^t_{k, S_1})$.

For all evaluation queries x_i before S_1 , output $F(k, x_i)$.

For all queries after S_1 , do the following.

 - (a) If $\mathcal{A}$ sends an evaluation query x_i and $x_i = \tilde{x}$, abort. Choose $\gamma_i^1 \leftarrow \{0, 1\}$. $\mathcal{A}$ wins if $\gamma_i^1 = 1$.
 - Else if $x_i \neq \tilde{x}$, output $F(k, x_i)$.

- (b) If $\mathcal{A}$ sends a t -punctured key query for set S_j and $\tilde{x} \notin S_j$, abort. Choose $\gamma_i^2 \leftarrow \{0, 1\}$. $\mathcal{A}$ wins if $\gamma_i^2 = 1$.
Else if $\tilde{x} \in S_j$, output $K_{S_j} \leftarrow i\mathcal{O}(\lambda, \text{PuncturedKey}^t_{k, S_j})$.
3. $\mathcal{A}$ sends challenge query x^* such that $x^* \neq x_i \forall i \leq q_1$ and $x^* \in S_j \forall j \leq q_2$.
 If $\tilde{x} \neq x^*$, abort. Choose $\gamma^* \leftarrow \{0, 1\}$. $\mathcal{A}$ wins if $\gamma^* = 1$.
Else if $\tilde{x} = x^*$, choose $\beta \leftarrow \{0, 1\}$. If $\beta = 0$, output $y = F(k, x^*)$, else output $y \leftarrow \mathcal{Y}$.
4. $\mathcal{A}$ sends β' and wins if $\beta = \beta'$.

Next, we define q_2 games, Game 1_l , $1 \leq l \leq q_2$. Let Game $1_0 = \text{Game } 1$.

Game 1_l In this game, the first l punctured key queries use $K_{\tilde{x}}$, while the remaining use k .

1. Choose a key $k \leftarrow \mathcal{K}$.
2. $\mathcal{A}$ makes evaluation/ t -punctured key queries.
 Let S_1 be the first t -punctured key query. Choose $\tilde{x} \leftarrow S_1$ and compute $K_{\tilde{x}} \leftarrow F.\text{puncture}(k, \tilde{x})$.
 Output $K_{S_1} \leftarrow i\mathcal{O}(\lambda, \text{PuncturedKeyAlt}^t_{K_{\tilde{x}}, S_1})$, where PuncturedKeyAlt^t is defined in Figure ??.
 For all evaluation queries x_i before S_1 , output $F(k, x_i)$.
 For all queries after S_1 , do the following.
 - (a) If $\mathcal{A}$ sends an evaluation query x_i and $x_i = \tilde{x}$, abort. Choose $\gamma_i^1 \leftarrow \{0, 1\}$. $\mathcal{A}$ wins if $\gamma_i^1 = 1$.
 Else if $x_i \neq \tilde{x}$, output $F.\text{eval}(K_{\tilde{x}}, x_i) = F(k, x_i)$.
 - (b) If $\mathcal{A}$ sends a t -punctured key query for set S_j and $\tilde{x} \notin S_j$, abort. Choose $\gamma_i^2 \leftarrow \{0, 1\}$. $\mathcal{A}$ wins if $\gamma_i^2 = 1$.
 Else if $\tilde{x} \in S_j$ and $j \leq l$, output $K_{S_j} \leftarrow i\mathcal{O}(\lambda, \text{PuncturedKeyAlt}^t_{K_{\tilde{x}}, S_j})$.
 Else output $K_{S_j} \leftarrow i\mathcal{O}(\lambda, \text{PuncturedKey}^t_{k, S_j})$.
3. $\mathcal{A}$ sends challenge query x^* such that $x^* \neq x_i \forall i \leq q_1$ and $x^* \in S_j \forall j \leq q_2$.
 If $\tilde{x} \neq x^*$, abort. Choose $\gamma^* \leftarrow \{0, 1\}$. $\mathcal{A}$ wins if $\gamma^* = 1$.
 Else if $\tilde{x} = x^*$, choose $\beta \leftarrow \{0, 1\}$. If $\beta = 0$, output $y = F(k, x^*)$, else output $y \leftarrow \mathcal{Y}$.
4. $\mathcal{A}$ sends β' and wins if $\beta = \beta'$.

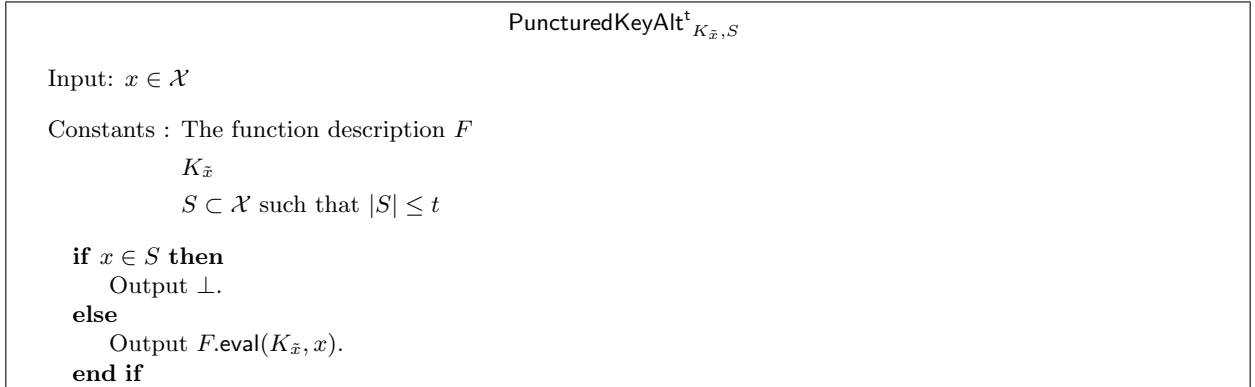


Figure 5: Program PuncturedKeyAlt^t

Game 2 In this game, the challenger outputs a random element as the response to the challenge query.

1. Choose a key $k \leftarrow \mathcal{K}$.
2. $\mathcal{A}$ makes evaluation/ t -punctured key queries.
 Let S_1 be the first t -punctured key query. Choose $\tilde{x} \leftarrow S_1$ and compute $K_{\tilde{x}} \leftarrow F.\text{puncture}(k, \tilde{x})$.
 Output $K_{S_1} \leftarrow i\mathcal{O}(\lambda, \text{PuncturedKeyAlt}^t_{K_{\tilde{x}}, S_1})$.

For all evaluation queries x_i before S_1 , output $F(k, x_i)$.

For all queries after S_1 , do the following.

- (a) If $\mathcal{A}$ sends an evaluation query x_i and $x_i = \tilde{x}$, abort. Choose $\gamma_i^1 \leftarrow \{0, 1\}$. $\mathcal{A}$ wins if $\gamma_i^1 = 1$.
Else if $x_i \neq \tilde{x}$, output $F.\text{eval}(K_{\tilde{x}}, x_i) = F(k, x_i)$.
- (b) If $\mathcal{A}$ sends a t -punctured key query for set S_j and $\tilde{x} \notin S_j$, abort. Choose $\gamma_i^2 \leftarrow \{0, 1\}$. $\mathcal{A}$ wins if $\gamma_i^2 = 1$.
Else if $\tilde{x} \in S_j$, output $K_{S_j} \leftarrow i\mathcal{O}(\lambda, \text{PuncturedKeyAlt}_{K_{\tilde{x}}, S_j}^t)$.
3. $\mathcal{A}$ sends challenge query x^* such that $x^* \neq x_i \forall i \leq q_1$ and $x^* \in S_j \forall j \leq q_2$.
If $\tilde{x} \neq x^*$, abort. Choose $\gamma^* \leftarrow \{0, 1\}$. $\mathcal{A}$ wins if $\gamma^* = 1$.
Else if $\tilde{x} = x^*$, choose $\beta \leftarrow \{0, 1\}$ and output $y \leftarrow \mathcal{Y}$.
4. $\mathcal{A}$ sends β' and wins if $\beta = \beta'$.

5.2.1 Adversary's Advantage in these Games

Let $\text{Adv}_{\mathcal{A}}^i$ denote the advantage of adversary $\mathcal{A}$ in Game i .

Observation 5.1. For any adversary $\mathcal{A}$, $\text{Adv}_{\mathcal{A}}^1 \geq \text{Adv}_{\mathcal{A}}^0/t$.

Proof. Since one of the elements of S_1 will be the challenge input, and $|S_1| \leq t$, the challenger's guess is correct with probability $1/|S_1| \geq 1/t$. Hence, $\text{Adv}_{\mathcal{A}}^1 \geq \text{Adv}_{\mathcal{A}}^0/t$. ■

We will now show that Game 1_l and Game 1_{l+1} are computationally indistinguishable, assuming $i\mathcal{O}$ is secure.

Claim 5.1. If there exists a PPT adversary $\mathcal{A}$ such that $\text{Adv}_{\mathcal{A}}^{1_l} - \text{Adv}_{\mathcal{A}}^{1_{l+1}}$ is non-negligible in λ , then there exists a PPT distinguisher $\mathcal{B}$ that breaks the security of $i\mathcal{O}$ with advantage non-negligible in λ .

Proof. Note that the only difference between Game 1_l and Game 1_{l+1} is in the response to the $(l+1)th$ t -punctured key query. In Game 1_l , $\text{PuncturedKey}_{k, S_{l+1}}^t$ is used to compute $K_{S_{l+1}}$, while in Game 1_{l+1} , $\text{PuncturedKeyAlt}_{K_{\tilde{x}}, S_{l+1}}^t$ is used. Suppose there exists a PPT adversary $\mathcal{A}$ such that $\text{Adv}_{\mathcal{A}}^{1_l} - \text{Adv}_{\mathcal{A}}^{1_{l+1}} = \epsilon$. We will construct a PPT algorithm $\mathcal{B}$ that interacts with $\mathcal{A}$ and breaks the security of $i\mathcal{O}$ with advantage ϵ .

$\mathcal{B}$ chooses $k \leftarrow \mathcal{K}$ and for all evaluation queries x_i before the first t -punctured key query, outputs $F(k, x_i)$. On receiving the first t -punctured key query S_1 , $\mathcal{B}$ chooses $\tilde{x} \leftarrow S_1$ and computes $K_{\tilde{x}} \leftarrow F.\text{puncture}(k, \tilde{x})$. The evaluation queries are computed as in Game 1_l and 1_{l+1} . The first l t -punctured key queries are constructed using k , while the last $q_2 - l - 1$ t -punctured keys are constructed using $K_{\tilde{x}}$ (as in Game 1_l and Game 1_{l+1}). For the $(l+1)th$ query, $\mathcal{B}$ does the following. $\mathcal{B}$ sets $C_0 = \text{PuncturedKey}_{k, S_{l+1}}^t$ and $C_1 = \text{PuncturedKeyAlt}_{K_{\tilde{x}}, S_{l+1}}^t$, and sends C_0, C_1 to the $i\mathcal{O}$ challenger, and receives $K_{S_{l+1}}$ in response, which it sends to $\mathcal{A}$.

Finally, after all queries, the challenger sends the challenge query x^* . $\mathcal{B}$ checks that $\tilde{x} = x^*$, sets $y_0 = F(k, x^*)$ and chooses $y_1 \leftarrow \mathcal{Y}, \beta \leftarrow \{0, 1\}$. It outputs y_β and receives β' in response. If $\beta = \beta'$, $\mathcal{B}$ outputs 0, else it outputs 1.

From the correctness property of puncturable PRFs, it follows that $F.\text{eval}(K_{\tilde{x}}, x) = F(k, x)$ for all $x \notin S_{l+1}$. Hence, the circuits C_0 and C_1 are functionally identical. This completes our proof. ■

Next, we show that Game 1_{q_2} and Game 2 are computationally indistinguishable.

Claim 5.2. If there exists a PPT adversary $\mathcal{A}$ such that $\text{Adv}_{\mathcal{A}}^{1_{q_2}} - \text{Adv}_{\mathcal{A}}^2$ is non-negligible in λ , then there exists a PPT distinguisher $\mathcal{B}$ that breaks the security of puncturable PRF F with advantage non-negligible in λ .

Proof. We will use $\mathcal{A}$ to construct a PPT algorithm $\mathcal{B}$ that breaks the security of puncturable PRF F with advantage $\text{Adv}_{\mathcal{A}}^{1_{q_2}} - \text{Adv}_{\mathcal{A}}^2$. Observe that in Game 1_{q_2} , the challenger requires the master key k only for the evaluation queries before the first t -punctured key query. After the first t -punctured key query S_1 , the challenger chooses $\tilde{x} \leftarrow S_1$, computes a punctured key $K_{\tilde{x}}$, and uses this to compute all future evaluation queries and t -punctured keys.

$\mathcal{B}$ begins interacting with $\mathcal{A}$. For each evaluation query x_i before the first t -punctured key query, $\mathcal{B}$ sends x_i to the puncturable PRF challenger, and receives y_i , which it forwards to $\mathcal{A}$. On receiving the first t -punctured key query S_1 , $\mathcal{B}$ chooses $\tilde{x} \leftarrow S_1$ and sends $\tilde{x}$ as challenge input to the puncturable PRF challenger. $\mathcal{B}$ receives $K_{\tilde{x}}$ and y . It uses $K_{\tilde{x}}$ for all remaining queries. On receiving challenge x^* from $\mathcal{A}$, $\mathcal{B}$ checks $x^* = \tilde{x}$ and sends y . $\mathcal{B}$ sends $\mathcal{A}$'s response to the PRF challenger.

Note that until the challenge query is made, both games are identical and $\mathcal{B}$ simulates them perfectly. If y is truly random, then $\mathcal{A}$ receives a response as per Game 2, else it receives a response as per Game 1_{q_2} . ■

Finally, we have the following simple observation.

Observation 5.2. For any adversary, $\text{Adv}_{\mathcal{A}}^3 = 0$.

From the above claims and observations, we can conclude that if $i\mathcal{O}$ is a secure indistinguishability obfuscator as per Definition 2.2, and F , together with $F.\text{setup}$, $F.\text{puncture}$, $F.\text{eval}$ is a secure puncturable PRF as per Definition 3.2, then any PPT adversary $\mathcal{A}$ has negligible advantage in Game 0.

Acknowledgements

We thank Allison Bishop Lewko for helpful comments and discussions.

References

- [ABB10] Shweta Agrawal, Dan Boneh, and Xavier Boyen. Efficient lattice (h)ibe in the standard model. In *Proceedings of the 29th Annual international conference on Theory and Applications of Cryptographic Techniques*, EUROCRYPT'10, pages 553–572, Berlin, Heidelberg, 2010. Springer-Verlag.
- [BB04] Dan Boneh and Xavier Boyen. Secure identity based encryption without random oracles. In *CRYPTO*, pages 443–459, 2004.
- [BCPR13] Nir Bitansky, Ran Canetti, Omer Paneth, and Alon Rosen. Indistinguishability obfuscation vs. auxiliary-input extractable functions: One must fall. *Cryptology ePrint Archive*, Report 2013/641, 2013. <http://eprint.iacr.org/>.
- [BGI13] Elette Boyle, Shafi Goldwasser, and Ioana Ivan. Functional signatures and pseudorandom functions. *IACR Cryptology ePrint Archive*, 2013:401, 2013.
- [BW13] Dan Boneh and Brent Waters. Constrained pseudorandom functions and their applications. In *ASIACRYPT*, pages 280–300, 2013.
- [BZ14] Dan Boneh and Mark Zhandry. Multiparty key exchange, efficient traitor tracing, and more from indistinguishability obfuscation. In *CRYPTO*, 2014.
- [CM14] Melissa Chase and Sarah Meiklejohn. Déjà q: Using dual systems to revisit q-type assumptions. In *EUROCRYPT*, pages 622–639, 2014.
- [FHPS13] Eduarda S. V. Freire, Dennis Hofheinz, Kenneth G. Paterson, and Christoph Striecks. Programmable hash functions in the multilinear setting. In *CRYPTO*, pages 513–530, 2013.

- [FKPR14] Georg Fuchsbauer, Momchil Konstantinov, Krzysztof Pietrzak, and Vanishree Rao. Adaptive security of constrained prfs. *IACR Cryptology ePrint Archive*, 2014:416, 2014.
- [GGH⁺13] Sanjam Garg, Craig Gentry, Shai Halevi, Mariana Raykova, Amit Sahai, and Brent Waters. Candidate indistinguishability obfuscation and functional encryption for all circuits. In *FOCS*, 2013.
- [GGM84] Oded Goldreich, Shafi Goldwasser, and Silvio Micali. How to construct random functions (extended abstract). In *FOCS*, pages 464–479, 1984.
- [GJM02] Philippe Golle, Stanislaw Jarecki, and Ilya Mironov. Cryptographic primitives enforcing communication and storage complexity. In *Financial Cryptography*, pages 120–135, 2002.
- [GLSW14] Craig Gentry, Allison Lewko, Amit Sahai, and Brent Waters. Indistinguishability obfuscation from the multilinear subgroup elimination assumption. Cryptology ePrint Archive, Report 2014/309, 2014. <http://eprint.iacr.org/>.
- [HK12] Dennis Hofheinz and Eike Kiltz. Programmable hash functions and their applications. *J. Cryptology*, 25(3):484–527, 2012.
- [Hof14] Dennis Hofheinz. Fully secure constrained pseudorandom functions using random oracles. *IACR Cryptology ePrint Archive*, 2014:372, 2014.
- [HSW13] Susan Hohenberger, Amit Sahai, and Brent Waters. Full domain hash from (leveled) multilinear maps and identity-based aggregate signatures. In *CRYPTO*, 2013.
- [HSW14] Susan Hohenberger, Amit Sahai, and Brent Waters. Replacing a random oracle: Full domain hash from indistinguishability obfuscation. In *EUROCRYPT*, pages 201–220, 2014.
- [HW10] Susan Hohenberger and Brent Waters. Constructing verifiable random functions with large input spaces. In *EUROCRYPT*, pages 656–672, 2010.
- [KPTZ13] Aggelos Kiayias, Stavros Papadopoulos, Nikos Triandopoulos, and Thomas Zacharias. Delegatable pseudorandom functions and applications. In *ACM Conference on Computer and Communications Security*, pages 669–684, 2013.
- [LW12] Allison B. Lewko and Brent Waters. New proof methods for attribute-based encryption: Achieving full security through selective techniques. In *CRYPTO*, pages 180–198, 2012.
- [NR04] Moni Naor and Omer Reingold. Number-theoretic constructions of efficient pseudo-random functions. *J. ACM*, 51(2):231–262, 2004.
- [SW14] Amit Sahai and Brent Waters. How to use indistinguishability obfuscation: deniable encryption, and more. In *STOC*, pages 475–484, 2014.
- [Wat05] Brent Waters. Efficient identity-based encryption without random oracles. In *EUROCRYPT*, pages 114–127, 2005.
- [Wat09] Brent Waters. Dual system encryption: Realizing fully secure ibe and hibe under simple assumptions. In *CRYPTO*, pages 619–636, 2009.

A Proof of Claim 4.2 ($\text{Adv}_{\mathcal{A}}^1 - \text{Adv}_{\mathcal{A}}^2$ is negligible)

Recall that Claim 4.2 states that assuming $i\mathcal{O}$ is a secure indistinguishability obfuscator and the Subgroup Hiding Assumption holds, for any PPT adversary $\mathcal{A}$,

$$\text{Adv}_{\mathcal{A}}^1 - \text{Adv}_{\mathcal{A}}^2 \leq \text{negl}(\lambda).$$

In order to prove this, we establish a sequence of intermediate experiments Game 1A to Game 1G and show that any two consecutive experiments are computationally indistinguishable. First, we recall Game 1.

Game 1 This is Game 1 from Section 4.1.1.

1. Let $(N, p, q, \mathbb{G}, \mathbb{G}_p, \mathbb{G}_q, g_1, g_2) \leftarrow \mathcal{G}(1^\lambda)$. Choose $u \leftarrow \text{AdmSample}(1^\lambda, q)$.
Choose $v \leftarrow \mathbb{G}$, $d_{i,b} \leftarrow \mathbb{Z}_N$. Set $k = (v, ((d_{1,0}, d_{1,1}), \dots, (d_{n,0}, d_{n,1})))$.
2. On any evaluation query $x_i \in \{0, 1\}^\ell$, check if $P_u(x_i) = 1$.
If not, flip a coin $\gamma_i \leftarrow \{0, 1\}$ and abort. $\mathcal{A}$ wins if $\gamma_i = 1$.
Else compute $h(x_i) = b_1^i \dots b_n^i$ and output $v^{\prod_j d_{j,b_j^i}}$.
3. $\mathcal{A}$ sends challenge input x^* such that $x^* \neq x_i$ for all $i \leq q$. Check if $P_u(x^*) = 0$.
If not, flip a coin $\gamma^* \leftarrow \{0, 1\}$ and abort. $\mathcal{A}$ wins if $\gamma^* = 1$.
Else compute $K_{x^*} \leftarrow i\mathcal{O}(\lambda, \text{PuncturedKey}_{k,x^*})$ and $h(x^*) = b_1^* \dots b_n^*$. Let $y_0 = v^{\prod_j d_{j,b_j^*}}$ and $y_1 \leftarrow \mathbb{G}$.
4. Flip coin $\beta \leftarrow \{0, 1\}$. Output (K_{x^*}, y_β) .
5. $\mathcal{A}$ outputs β' and wins if $\beta = \beta'$.

Game 1A We change the sampling procedure for the $d_{i,b}$ values so that some include a factor of a .

1. Let $(N, p, q, \mathbb{G}, \mathbb{G}_p, \mathbb{G}_q, g_1, g_2) \leftarrow \mathcal{G}(1^\lambda)$. Choose $u \leftarrow \text{AdmSample}(1^\lambda, q)$.
Choose $v \leftarrow \mathbb{G}$, $d'_{i,b} \leftarrow \mathbb{Z}_N$, $a \leftarrow \mathbb{Z}_N^*$ and set $d_{i,b} = d'_{i,b}$ if $u_i = b$, else $d_{i,b} = d'_{i,b} \cdot a$.⁶
Set $k = (v, ((d_{1,0}, d_{1,1}), \dots, (d_{n,0}, d_{n,1})))$.
2. On any evaluation query $x_i \in \{0, 1\}^\ell$, check if $P_u(x_i) = 1$.
If not, flip a coin $\gamma_i \leftarrow \{0, 1\}$ and abort. $\mathcal{A}$ wins if $\gamma_i = 1$.
Else compute $h(x_i) = b_1^i \dots b_n^i$ and output $v^{\prod_j d_{j,b_j^i}}$.
3. $\mathcal{A}$ sends challenge input x^* such that $x^* \neq x_i$ for all $i \leq q$. Check if $P_u(x^*) = 0$.
If not, flip a coin $\gamma^* \leftarrow \{0, 1\}$ and abort. $\mathcal{A}$ wins if $\gamma^* = 1$.
Else compute $K_{x^*} \leftarrow i\mathcal{O}(\lambda, \text{PuncturedKey}_{k,x^*})$ and $h(x^*) = b_1^* \dots b_n^*$. Let $y_0 = v^{\prod_j d_{j,b_j^*}}$ and $y_1 \leftarrow \mathbb{G}$.
4. Flip coin $\beta \leftarrow \{0, 1\}$. Output (K_{x^*}, y_β) .
5. $\mathcal{A}$ outputs β' and wins if $\beta = \beta'$.

Game 1B We substitute an alternative method of computing the punctured key program output using a differently formatted input.

1. Let $(N, p, q, \mathbb{G}, \mathbb{G}_p, \mathbb{G}_q, g_1, g_2) \leftarrow \mathcal{G}(1^\lambda)$. Choose $u \leftarrow \text{AdmSample}(1^\lambda, q)$. Let $r_u(x) = |\{j : u_j \neq h(x)_j\}|$.
Choose $v \leftarrow \mathbb{G}$, $d'_{i,b} \leftarrow \mathbb{Z}_N$, $a \leftarrow \mathbb{Z}_N^*$ and set $d_{i,b} = d'_{i,b}$ if $u_i = b$, else $d_{i,b} = d'_{i,b} \cdot a$.
Let $D = ((d'_{1,0}, d'_{1,1}), \dots, (d'_{n,0}, d'_{n,1}))$, $V = (v, v^a, \dots, v^{a^{n-1}})$ and $w = v^{a^n}$.
2. On any evaluation query $x_i \in \{0, 1\}^\ell$, check if $P_u(x_i) = 1$.⁷
If not, flip a coin $\gamma_i \leftarrow \{0, 1\}$ and abort. $\mathcal{A}$ wins if $\gamma_i = 1$.
Else compute $h(x_i) = b_1^i \dots b_n^i$. Output $v^{\prod_j d_{j,b_j^i}} = \left(v^{a^{r_u(x_i)}}\right)^{\prod_j d'_{j,b_j^i}}$.
3. $\mathcal{A}$ sends challenge input x^* such that $x^* \neq x_i$ for all $i \leq q$. Check if $P_u(x^*) = 0$.
If not, flip a coin $\gamma^* \leftarrow \{0, 1\}$ and abort. $\mathcal{A}$ wins if $\gamma^* = 1$.
Else compute $K_{x^*} \leftarrow i\mathcal{O}(\lambda, \text{PuncturedKey}'_{V,w,D,u,x^*})$ and $h(x^*) = b_1^* \dots b_n^*$. Let $y_0 = v^{\prod_j d_{j,b_j^*}} = w^{\prod_j d'_{j,b_j^*}}$ and $y_1 \leftarrow \mathbb{G}$.
4. Flip coin $\beta \leftarrow \{0, 1\}$. Output (K_{x^*}, y_β) .
5. $\mathcal{A}$ outputs β' and wins if $\beta = \beta'$.

⁶If $u_i = \perp$, then $d_{i,b} = d'_{i,b} \cdot a$ for $b = 0, 1$.

⁷Note that $r_u(x) < n$ if $P_u(x) = 1$, else $r_u(x) = n$.

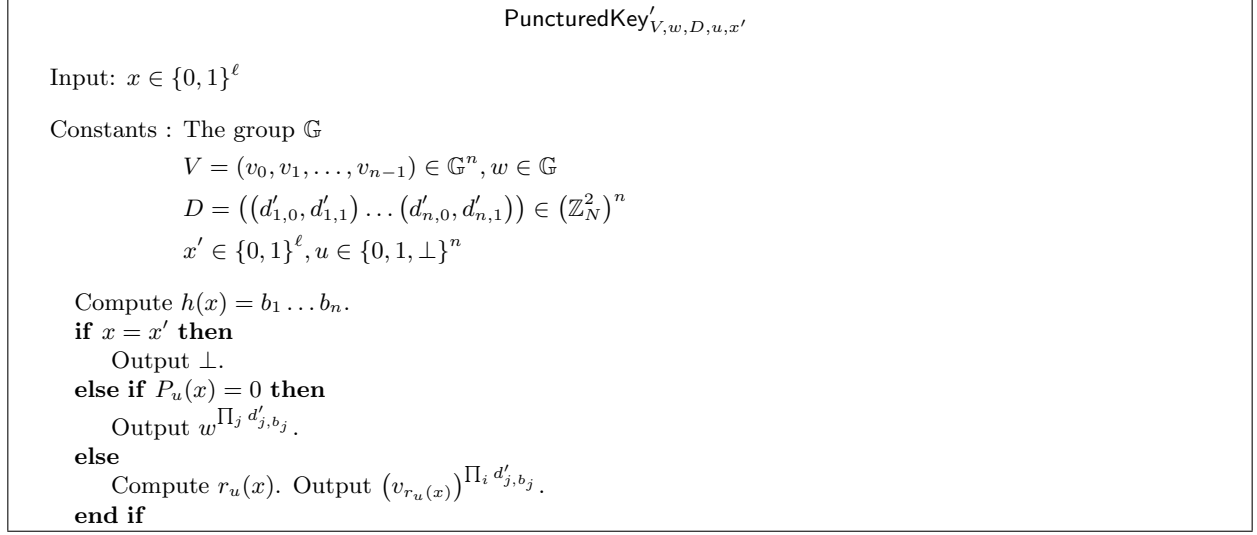


Figure 6: Program $\text{PuncturedKey}'$

Game 1C In this game, the term v^{a^n} is replaced by a random element of $\mathbb{G}$.

1. Let $(N, p, q, \mathbb{G}, \mathbb{G}_p, \mathbb{G}_q, g_1, g_2) \leftarrow \mathcal{G}(1^\lambda)$. Choose $u \leftarrow \text{AdmSample}(1^\lambda, q)$. Let $r_u(x) = |\{j : u_j \neq h(x)_j\}|$. Choose $v \leftarrow \mathbb{G}$, $d'_{i,b} \leftarrow \mathbb{Z}_N$, $a \leftarrow \mathbb{Z}_N^*$ and set $d_{i,b} = d'_{i,b}$ if $u_i = b$, else $d_{i,b} = d'_{i,b} \cdot a$. Let $D = ((d'_{1,0}, d'_{1,1}) \dots (d'_{n,0}, d'_{n,1}))$, $V = (v, v^a, \dots, v^{a^{n-1}})$ and $w \leftarrow \mathbb{G}$.
2. On any evaluation query $x_i \in \{0, 1\}^\ell$, check if $P_u(x_i) = 1$. If not, flip a coin $\gamma_i \leftarrow \{0, 1\}$ and abort. $\mathcal{A}$ wins if $\gamma_i = 1$. Else compute $h(x_i) = b_1^i \dots b_n^i$. Output $v^{\prod_j d_{j,b_j^i}} = \left(v^{a^{r_u(x_i)}}\right)^{\prod_j d'_{i,b_j^i}}$.
3. $\mathcal{A}$ sends challenge input x^* such that $x^* \neq x_i$ for all $i \leq q$. Check if $P_u(x^*) = 0$. If not, flip a coin $\gamma^* \leftarrow \{0, 1\}$ and abort. $\mathcal{A}$ wins if $\gamma^* = 1$. Else compute $K_{x^*} \leftarrow i\mathcal{O}(\lambda, \text{PuncturedKey}'_{V,w,D,u,x^*})$ and $h(x^*) = b_1^* \dots b_n^*$. Let $y_0 = w^{\prod_j d'_{j,b_j^*}}$ and $y_1 \leftarrow \mathbb{G}$.
4. Flip coin $\beta \leftarrow \{0, 1\}$. Output (K_{x^*}, y_β) .
5. $\mathcal{A}$ outputs β' and wins if $\beta = \beta'$.

Game 1D This game is same as the previous one, except that v and w are chosen from subgroups $\mathbb{G}_p$ and $\mathbb{G}_q$ respectively, instead of $\mathbb{G}$.

1. Let $(N, p, q, \mathbb{G}, \mathbb{G}_p, \mathbb{G}_q, g_1, g_2) \leftarrow \mathcal{G}(1^\lambda)$. Choose $u \leftarrow \text{AdmSample}(1^\lambda, q)$. Let $r_u(x) = |\{j : u_j \neq h(x)_j\}|$. Choose $v \leftarrow \mathbb{G}_p$, $w \leftarrow \mathbb{G}_q$, $d'_{i,b} \leftarrow \mathbb{Z}_N$, $a \leftarrow \mathbb{Z}_N^*$ and set $d_{i,b} = d'_{i,b}$ if $u_i = b$, else $d_{i,b} = d'_{i,b} \cdot a$. Let $D = ((d'_{1,0}, d'_{1,1}) \dots (d'_{n,0}, d'_{n,1}))$, $V = (v, v^a, \dots, v^{a^{n-1}})$.
2. On any evaluation query $x_i \in \{0, 1\}^\ell$, check if $P_u(x_i) = 1$. If not, flip a coin $\gamma_i \leftarrow \{0, 1\}$ and abort. $\mathcal{A}$ wins if $\gamma_i = 1$. Else compute $h(x_i) = b_1^i \dots b_n^i$. Output $v^{\prod_j d_{j,b_j^i}} = \left(v^{a^{r_u(x_i)}}\right)^{\prod_j d'_{i,b_j^i}}$.
3. $\mathcal{A}$ sends challenge input x^* such that $x^* \neq x_i$ for all $i \leq q$. Check if $P_u(x^*) = 0$. If not, flip a coin $\gamma^* \leftarrow \{0, 1\}$ and abort. $\mathcal{A}$ wins if $\gamma^* = 1$. Else compute $K_{x^*} \leftarrow i\mathcal{O}(\lambda, \text{PuncturedKey}'_{V,w,D,u,x^*})$ and $h(x^*) = b_1^* \dots b_n^*$. Let $y_0 = w^{\prod_j d'_{j,b_j^*}}$ and $y_1 \leftarrow \mathbb{G}$.
4. Flip coin $\beta \leftarrow \{0, 1\}$. Output (K_{x^*}, y_β) .
5. $\mathcal{A}$ outputs β' and wins if $\beta = \beta'$.

Game 1E

1. Let $(N, p, q, \mathbb{G}, \mathbb{G}_p, \mathbb{G}_q, g_1, g_2) \leftarrow \mathcal{G}(1^\lambda)$. Choose $u \leftarrow \text{AdmSample}(1^\lambda, q)$. Choose $v \leftarrow \mathbb{G}_p$, $w \leftarrow \mathbb{G}_q$, $d_{i,b} \leftarrow \mathbb{Z}_N$. Let $d_{i,b,p} = d_{i,b} \bmod p$ and $d_{i,b,q} = d_{i,b} \bmod q$.
Set $k = (v, ((d_{1,0,p}, d_{1,1,p}), \dots, (d_{n,0,p}, d_{n,1,p})))$ and $k' = (w, ((d_{1,0,q}, d_{1,1,q}), \dots, (d_{n,0,q}, d_{n,1,q})))$.
2. On any evaluation query $x_i \in \{0, 1\}^\ell$, check if $P_u(x_i) = 1$.
 If not, flip a coin $\gamma_i \leftarrow \{0, 1\}$ and abort. $\mathcal{A}$ wins if $\gamma_i = 1$.
 Else compute $h(x_i) = b_1^i \dots b_n^i$. Output $v^{\prod_j d_{j,b_j^i}} = v^{\prod_j d_{j,b_j^i,p}}$.
3. $\mathcal{A}$ sends challenge input x^* such that $x^* \neq x_i$ for all $i \leq q$. Check if $P_u(x^*) = 0$.
 If not, flip a coin $\gamma^* \leftarrow \{0, 1\}$ and abort. $\mathcal{A}$ wins if $\gamma^* = 1$.
 Else compute $K_{x^*} \leftarrow i\mathcal{O}(\lambda, \text{PuncturedKeyAlt}_{u,k,k',x^*})$ and $h(x^*) = b_1^* \dots b_n^*$. Let $y_0 = w^{\prod_j d_{j,b_j^*}} = w^{\prod_j d_{j,b_j^*,q}}$ and $y_1 \leftarrow \mathbb{G}$.
4. Flip coin $\beta \leftarrow \{0, 1\}$. Output (K_{x^*}, y_β) .
5. $\mathcal{A}$ outputs β' and wins if $\beta = \beta'$.

Game 1F

1. Let $(N, p, q, \mathbb{G}, \mathbb{G}_p, \mathbb{G}_q, g_1, g_2) \leftarrow \mathcal{G}(1^\lambda)$. Choose $u \leftarrow \text{AdmSample}(1^\lambda, q)$. Choose $v \leftarrow \mathbb{G}_p$, $w \leftarrow \mathbb{G}_q$, $d_{i,b,p} \leftarrow \mathbb{Z}_p$, $e_{i,b,q} \leftarrow \mathbb{Z}_q$.
Set $k = (v, ((d_{1,0,p}, d_{1,1,p}), \dots, (d_{n,0,p}, d_{n,1,p})))$ and $k' = (w, ((e_{1,0,q}, e_{1,1,q}), \dots, (e_{n,0,q}, d_{n,1,q})))$.
2. On any evaluation query $x_i \in \{0, 1\}^\ell$, check if $P_u(x_i) = 1$.
 If not, flip a coin $\gamma_i \leftarrow \{0, 1\}$ and abort. $\mathcal{A}$ wins if $\gamma_i = 1$.
 Else compute $h(x_i) = b_1^i \dots b_n^i$. Output $v^{\prod_j d_{j,b_j^i,p}}$.
3. $\mathcal{A}$ sends challenge input x^* such that $x^* \neq x_i$ for all $i \leq q$. Check if $P_u(x^*) = 0$.
 If not, flip a coin $\gamma^* \leftarrow \{0, 1\}$ and abort. $\mathcal{A}$ wins if $\gamma^* = 1$.
 Else compute $K_{x^*} \leftarrow i\mathcal{O}(\lambda, \text{PuncturedKeyAlt}_{u,k,k',x^*})$ and $h(x^*) = b_1^* \dots b_n^*$. Let $y_0 = w^{\prod_j e_{j,b_j^*,q}}$ and $y_1 \leftarrow \mathbb{G}$.
4. Flip coin $\beta \leftarrow \{0, 1\}$. Output (K_{x^*}, y_β) .
5. $\mathcal{A}$ outputs β' and wins if $\beta = \beta'$.

Game 1G This game is same as the previous one, except that the $d_{i,b}$ and $e_{i,b}$ values are uniformly chosen from $\mathbb{Z}_N$ instead of $\mathbb{Z}_p$ and $\mathbb{Z}_q$, respectively.

1. Let $(N, p, q, \mathbb{G}, \mathbb{G}_p, \mathbb{G}_q, g_1, g_2) \leftarrow \mathcal{G}(1^\lambda)$. Choose $u \leftarrow \text{AdmSample}(1^\lambda, q)$. Choose $v \leftarrow \mathbb{G}_p$, $w \leftarrow \mathbb{G}_q$, $d_{i,b} \leftarrow \mathbb{Z}_N$, $e_{i,b} \leftarrow \mathbb{Z}_N$.
 Set $k = (v, ((d_{1,0}, d_{1,1}), \dots, (d_{n,0}, d_{n,1})))$ and $k' = (w, ((e_{1,0}, e_{1,1}), \dots, (e_{n,0}, d_{n,1})))$.
2. On any evaluation query $x_i \in \{0, 1\}^\ell$, check if $P_u(x_i) = 1$.
 If not, flip a coin $\gamma_i \leftarrow \{0, 1\}$ and abort. $\mathcal{A}$ wins if $\gamma_i = 1$.
 Else compute $h(x_i) = b_1^i \dots b_n^i$. Output $v^{\prod_j d_{j,b_j^i}}$.
3. $\mathcal{A}$ sends challenge input x^* such that $x^* \neq x_i$ for all $i \leq q$. Check if $P_u(x^*) = 0$.
 If not, flip a coin $\gamma^* \leftarrow \{0, 1\}$ and abort. $\mathcal{A}$ wins if $\gamma^* = 1$.
 Else compute $K_{x^*} \leftarrow i\mathcal{O}(\lambda, \text{PuncturedKeyAlt}_{u,k,k',x^*})$ and $h(x^*) = b_1^* \dots b_n^*$. Let $y_0 = w^{\prod_j e_{j,b_j^*}}$ and $y_1 \leftarrow \mathbb{G}$.
4. Flip coin $\beta \leftarrow \{0, 1\}$. Output (K_{x^*}, y_β) .
5. $\mathcal{A}$ outputs β' and wins if $\beta = \beta'$.

Game 2 This is Game 2 from Section 4.1.1. This game is same as the previous one, except that v and w are chosen from $\mathbb{G}$ instead of the subgroups $\mathbb{G}_p$ and $\mathbb{G}_q$ respectively.

1. Let $(N, p, q, \mathbb{G}, \mathbb{G}_p, \mathbb{G}_q, g_1, g_2) \leftarrow \mathcal{G}(1^\lambda)$. Choose $u \leftarrow \text{AdmSample}(1^\lambda, q)$. Choose $v \leftarrow \mathbb{G}$, $w \leftarrow \mathbb{G}$, $d_{i,b} \leftarrow \mathbb{Z}_N$, $e_{i,b} \leftarrow \mathbb{Z}_N$.
Set $k = (v, ((d_{1,0}, d_{1,1}), \dots, (d_{n,0}, d_{n,1})))$ and $k' = (w, ((e_{1,0}, e_{1,1}), \dots, (e_{n,0}, e_{n,1})))$.
2. On any evaluation query $x_i \in \{0, 1\}^\ell$, check if $P_u(x_i) = 1$.
If not, flip a coin $\gamma_i \leftarrow \{0, 1\}$ and abort. $\mathcal{A}$ wins if $\gamma_i = 1$.
Else compute $h(x_i) = b_1^{i_1} \dots b_n^{i_n}$. Output $= v^{\prod_j d_{j,b_j^i}}$.
3. $\mathcal{A}$ sends challenge input x^* such that $x^* \neq x_i$ for all $i \leq q$. Check if $P_u(x^*) = 0$.
If not, flip a coin $\gamma^* \leftarrow \{0, 1\}$ and abort. $\mathcal{A}$ wins if $\gamma^* = 1$.
Else compute $K_{x^*} \leftarrow i\mathcal{O}(\lambda, \text{PuncturedKeyAlt}_{u,k,k',x^*})$ and $h(x^*) = b_1^* \dots b_n^*$. Let $y_0 = w^{\prod_j e_{j,b_j^*}}$ and $y_1 \leftarrow \mathbb{G}$.
4. Flip coin $\beta \leftarrow \{0, 1\}$. Output (K_{x^*}, y_β) .
5. $\mathcal{A}$ outputs β' and wins if $\beta = \beta'$.

We will now prove that the difference in advantage of any PPT adversary in two consecutive game is negligible in λ . Let $\text{Adv}_{\mathcal{A}}^{1\alpha}$ denote the advantage of adversary $\mathcal{A}$ in Game 1 α .

Observation A.1. For any adversary $\mathcal{A}$, $\text{Adv}_{\mathcal{A}}^{1A} = \text{Adv}_{\mathcal{A}}^{1A}$.

Proof. The only difference between Game 1 and Game 1A is the manner in which the constants $d_{i,b}$ are chosen. In Game 1, the challenger chooses $d_{i,b} \leftarrow \mathbb{Z}_N$. In Game 1A, the challenger first chooses $d'_{i,b} \leftarrow \mathbb{Z}_N$, $a \leftarrow \mathbb{Z}_N^*$ and then sets $d_{i,b}$ as either $d'_{i,b}$ or $d'_{i,b} \cdot a$, depending on u_i . However, note that in either case, $d_{i,b}$ is a uniformly random element in $\mathbb{Z}_N$ since $a \in \mathbb{Z}_N^*$. ■

Claim A.1. If there exists a PPT adversary $\mathcal{A}$ such that $\text{Adv}_{\mathcal{A}}^{1A} - \text{Adv}_{\mathcal{A}}^{1B}$ is non-negligible in λ , then there exists a PPT adversary $\mathcal{B}$ that breaks the security of $i\mathcal{O}$ with advantage non-negligible in λ .

Proof. Suppose there exists a PPT adversary $\mathcal{A}$ such that $\text{Adv}_{\mathcal{A}}^{1A} - \text{Adv}_{\mathcal{A}}^{1B} = \epsilon$. We will construct a PPT algorithm $\mathcal{B}$ that breaks the security of $i\mathcal{O}$ with advantage ϵ by interacting with $\mathcal{A}$. $\mathcal{B}$ establishes some parameters that will be needed to interact with $\mathcal{A}$ and which will be used as part of the circuits designed by $\mathcal{B}$: first it chooses $v \leftarrow \mathbb{G}$, $a \leftarrow \mathbb{Z}_N^*$, $d'_{i,b} \leftarrow \mathbb{Z}_N$, $u \leftarrow \text{AdmSample}(1^\lambda, q)$ and sets $d_{i,b} = d'_{i,b}$ if $u_i = b$, else $d_{i,b} = d'_{i,b} \cdot a$. On receiving evaluation query x_i , $\mathcal{B}$ first checks that $P_u(x_i) = 1$. If so, it computes $h(x) = b_1^{i_1} \dots b_n^{i_n}$ and sends $v^{\prod_j d_{j,b_j^i}}$ to $\mathcal{A}$.

On receiving challenge query x^* from $\mathcal{A}$, $\mathcal{B}$ checks $P_u(x^*) = 0$. Now, $\mathcal{B}$ is ready to construct the circuits. It sets $k = (v, ((d_{1,0}, d_{1,1}), \dots, (d_{n,0}, d_{n,1})))$, $V = (v, v^a, \dots, v^{a^{n-1}})$, $w = v^{a^n}$ and $D = ((d'_{1,0}, d'_{1,1}), \dots, (d'_{n,0}, d'_{n,1}))$. It uses k to construct circuit $C_0 = \text{PuncturedKey}_{k,x^*}$, uses V, w, D to construct $C_1 = \text{PuncturedKey}'_{V,w,D,u,x^*}$ and sends C_0, C_1 to the $i\mathcal{O}$ challenger. $\mathcal{B}$ receives $K_{x^*} \leftarrow i\mathcal{O}(C_b)$ from the challenger. It computes $h(x^*) = b_1^* \dots b_n^*$, $y_0 = v^{\prod_j d_{j,b_j^*}}$, $y \leftarrow \mathbb{G}$, $\beta \leftarrow \{0, 1\}$, sends (K_{x^*}, y_β) to $\mathcal{A}$ and receives β' in response. If $\beta = \beta'$, $\mathcal{B}$ outputs 0, else it outputs 1.

We will now prove that the circuits C_0 and C_1 have identical functionality. Consider any ℓ bit string x , and let $h(x) = b_1 \dots b_n$. Recall $r_u(x) = |\{j : u_j \neq b_j\}|$.

$$\prod_j d_{j,b_j} = \left(\prod_{u_j=b_j} d'_{j,b_j} \right) \cdot \left(\prod_{u_j \neq b_j} (d'_{j,b_j} \cdot a) \right) = \left(\prod_{u_j=b_j} d'_{j,b_j} \right) \cdot a^{r_u(x)} \cdot \left(\prod_{u_j \neq b_j} d'_{j,b_j} \right)$$

Hence, $v^{\prod_j d_{j,b_j}} = \left(v^{a^{r_u(x)}} \right)^{\prod_j d'_{j,b_j}}$. Also, recall $r_u(x) = n$ iff $P_u(x) = 0$. Therefore, to compute $v^{\prod_j d_{j,b_j}}$ for some x such that $P_u(x) = 1$, we only need the constant $\{d'_{i,b}\}$ and $\{v, v^a, v^{a^2}, \dots, v^{a^{n-1}}\}$. Similarly, if $P_u(x) = 0$, we only need the constants $\{d'_{i,b}\}$ and v^{a^n} to compute $v^{\prod_j d_{j,b_j}}$. Using this observation, we can prove that the programs $\text{PuncturedKey}_{k,x^*}$ and $\text{PuncturedKey}'_{V,w,D,u,x^*}$ have identical functionality.

For any $x \in \{0, 1\}^\ell$ such that $P_u(x) = 1$,

$$\text{PuncturedKey}'_{V,w,D,u,x^*}(x) = \left(v^{a^{r_u(x)}}\right)^{\prod_j d'_{j,b_j}} = v^{\prod_j d_{j,b_j}} = \text{PuncturedKey}_{k,x^*}(x)$$

For any $x \in \{0, 1\}^\ell$ such that $P_u(x) = 0$,

$$\text{PuncturedKey}'_{V,w,D,u,x^*}(x) = w^{\prod_j d'_{j,b_j}} = \left(v^{a^n}\right)^{\prod_j d'_{j,b_j}} = v^{\prod_j d_{j,b_j}} = \text{PuncturedKey}_{k,x^*}(x)$$

Since C_0 and C_1 have identical functionality, $\Pr[\mathcal{B} \text{ wins}] = \Pr[\mathcal{A} \text{ wins in Game 1A}] - \Pr[\mathcal{A} \text{ wins in Game 1B}]$. If $\text{Adv}_{\mathcal{A}}^{1A} - \text{Adv}_{\mathcal{A}}^{1B} = \epsilon$, then $\mathcal{B}$ wins the $i\mathcal{O}$ security game with advantage ϵ . $\blacksquare$

Claim A.2. If there exists a PPT adversary $\mathcal{A}$ such that $\text{Adv}_{\mathcal{A}}^{1B} - \text{Adv}_{\mathcal{A}}^{1C}$ is non-negligible in λ , then there exists a PPT adversary $\mathcal{B}$ that breaks Assumption 2 with advantage non-negligible in λ .

Proof. Suppose there exists a PPT adversary $\mathcal{A}$ such that $\text{Adv}_{\mathcal{A}}^{1B} - \text{Adv}_{\mathcal{A}}^{1C} = \epsilon$. We will construct a PPT algorithm $\mathcal{B}$ that uses $\mathcal{A}$ and breaks Assumption 2 with advantage at least ϵ . $\mathcal{B}$ receives the group description $\mathbb{G}$, $(v, v^a, \dots, v^{a^{n-1}}) \in \mathbb{G}^n$ and $T \in \mathbb{G}$, where $T = v^{a^n}$ or $T \leftarrow \mathbb{G}$. $\mathcal{B}$ chooses $d'_{i,b} \leftarrow \mathbb{Z}_N$, $u \leftarrow \text{AdmSample}(1^\lambda, q)$. $\mathcal{B}$ sets $V = (v, \dots, v^{a^{n-1}})$, $w = T$, $D = ((d'_{1,0}, d'_{1,1}), \dots, (d'_{n,0}, d'_{n,1}))$.

On evaluation query x_i , $\mathcal{B}$ first checks that $P_u(x_i) = 1$. Next, it computes $h(x) = b_1^i \dots b_n^i$, and finally outputs $\left(v^{a^{r_u(x_i)}}\right)^{\prod_j d'_{j,b_j^i}}$. Note that $r_u(x) < n$ if $P_u(x) = 0$. Therefore, $\mathcal{B}$ can simulate the evaluation query phase perfectly.

On challenge query x^* , $\mathcal{B}$ first checks that $P_u(x^*) = 0$. Next, it computes $h(x) = b_1^* \dots b_n^*$, $K_{x^*} \leftarrow i\mathcal{O}(\lambda, \text{PuncturedKey}'_{V,w,D,u,x^*})$. It sets $y_0 = w^{\prod_j d'_{j,b_j^*}}$ and $y_1 \leftarrow \mathbb{G}$.

Finally, $\mathcal{B}$ chooses $\beta \leftarrow \{0, 1\}$, sends (K_{x^*}, y_β) from $\mathcal{A}$, and receives β' . If $\beta = \beta'$, $\mathcal{B}$ outputs 0 (i.e. it guesses $T = v^{a^n}$), else it outputs 1.

Clearly, if $\text{Adv}_{\mathcal{A}}^{1B} - \text{Adv}_{\mathcal{A}}^{1C} = \epsilon$, then $\mathcal{B}$ also wins with advantage ϵ . $\blacksquare$

Claim A.3. If there exists a PPT adversary $\mathcal{A}$ such that $\text{Adv}_{\mathcal{A}}^{1C} - \text{Adv}_{\mathcal{A}}^{1D}$ is non-negligible in λ , then there exists a PPT adversary $\mathcal{B}$ that breaks Assumption 1 with advantage non-negligible in λ .

Proof. We will prove this claim using an intermediate experiment Hyb . Hyb is the same as Game 1C, except that $v \leftarrow \mathbb{G}_p$. We will prove that Game 1C and Hyb are computationally indistinguishable, and similarly, Hyb and Game 1D are computationally indistinguishable.

Suppose there exists a PPT adversary $\mathcal{A}$ such that $\text{Adv}_{\mathcal{A}}^{1C} - \text{Adv}_{\mathcal{A}}^{\text{Hyb}} = \epsilon$. Using $\mathcal{A}$, we will construct a PPT algorithm $\mathcal{B}$ that breaks Assumption 1. $\mathcal{B}$ receives $\mathbb{G}$, $\mathbb{G}_p$, $\mathbb{G}_q$, $g_1 \leftarrow \mathbb{G}_p$, $g_2 \leftarrow \mathbb{G}_q$ and T , where $T \leftarrow \mathbb{G}_p$ or $T \leftarrow \mathbb{G}$. $\mathcal{B}$ sets $v = T$, chooses $a \leftarrow \mathbb{Z}_N^*$, $d'_{i,b} \leftarrow \mathbb{Z}_N$, $u \leftarrow \text{AdmSample}$, $w \leftarrow \mathbb{G}$ and computes $V = (v, v^a, \dots, v^{a^{n-1}})$ and $d_{i,b}$ as in Game 1C.

On receiving evaluation query x_i from $\mathcal{A}$, $\mathcal{B}$ computes $h(x_i) = b_1^i \dots b_n^i$, checks if $P_u(x_i) = 1$ and responds with $\left(v^{a^{r_u(x_i)}}\right)^{\prod_j d'_{j,b_j^i}}$.

On receiving challenge query x^* , $\mathcal{B}$ checks $P_u(x^*) = 0$, and computes $K_{x^*} \leftarrow i\mathcal{O}(\lambda, \text{PuncturedKey}'_{V,w,D,u,x^*})$, $y_0 = (w)^{\prod_j d'_{j,b_j^*}}$, $y_1 \leftarrow \mathbb{G}$, $\beta \leftarrow \{0, 1\}$ and outputs (K_{x^*}, y_β) . If $\mathcal{A}$ outputs β , $\mathcal{B}$ guesses $v \in \mathbb{G}$, else guesses $v \in \mathbb{G}_p$. Notice that if $\text{Adv}_{\mathcal{A}}^{\text{Hyb}} - \text{Adv}_{\mathcal{A}}^{1C} = \epsilon$, then $\mathcal{B}$ breaks Assumption 1 with advantage ϵ .

In order to prove our claim, we now need to show that the experiments Hyb and Game 1D are computationally indistinguishable. Note that the only difference between these two experiments is the manner in which w is chosen. In Hyb , $w \leftarrow \mathbb{G}$, while in Game 1D, $w \leftarrow \mathbb{G}_q$. We can show that if Assumption 1 holds, then Hyb and Game 1D are computationally indistinguishable. This argument is exactly similar to the step from Game 1C to Hyb , the only difference being that the assumption used will be that $w \leftarrow \mathbb{G}$ is computationally indistinguishable from $w \leftarrow \mathbb{G}_q$. $\blacksquare$

Claim A.4. If there exists a PPT adversary $\mathcal{A}$ such that $\text{Adv}_{\mathcal{A}}^{1D} - \text{Adv}_{\mathcal{A}}^{1E}$ is non-negligible in λ , then there exists a PPT adversary $\mathcal{B}$ that breaks the security of $i\mathcal{O}$ with advantage non-negligible in λ .

Proof. We will use two intermediate experiments Hyb_1 and Hyb_2 , and use the security of $i\mathcal{O}$ to prove that (a) Game 1D and Hyb_1 are computationally indistinguishable, (b) Hyb_1 and Hyb_2 are identical experiments, and (c) Hyb_2 and Game 1E are computationally indistinguishable.

First we define the experiments Hyb_1 and Hyb_2 . In Hyb_1 , the challenger first samples $u, v, w, d'_{i,b}, a$ as in Game 1D, and sets $d_{i,b} = d'_{i,b}$ if $u_i = b$, else $d_{i,b} = a$ (same as before). The evaluation query responses are also same as in Game 1D. For the challenge query x^* , the challenger first checks that $P_u(x^*) = 0$. Next, it sets $w' = w^{1/a^n}$, computes $y_0 = w'^{\prod_j d_{j,b_j^*}} = w^{\prod_j d_{j,b_j^*}}$, $y_1 \leftarrow \mathbb{G}$ as in Game 1D. However, the punctured key K_{x^*} is computed using PuncturedKeyAlt . The challenger sets $k = (v, ((d_{1,0}, d_{1,1}), \dots, (d_{n,0}, d_{n,1})))$, $k' = (w', ((d_{1,0}, d_{1,1}), \dots, (d_{n,0}, d_{n,1})))$ and computes $K_{x^*} \leftarrow i\mathcal{O}(\lambda, \text{PuncturedKeyAlt}_{k,k',u,x^*})$.

Hyb_2 is the same as Hyb_1 , except that it chooses $d_{i,b} \leftarrow \mathbb{Z}_N$ and uses w instead of w' in the key k' and for computing y_0 .

Proof of (a): Suppose there exists a PPT adversary $\mathcal{A}$ such that $\text{Adv}_{\mathcal{A}}^{1D} - \text{Adv}_{\mathcal{A}}^{\text{Hyb}_1} = \epsilon$. As in the proof of Claim A.1, we will first construct a PPT algorithm $\mathcal{B}$ that uses $\mathcal{A}$ to break the security of $i\mathcal{O}$. $\mathcal{B}$ samples $u, v, w, d'_{i,b}$ and computes $d_{i,b}$ as in Game 1D/ Hyb . On receiving evaluation query x_i , it checks $P_u(x_i) = 0$ and outputs $v^{\prod_j d_{j,b_j^i}}$. On receiving challenge query x^* , $\mathcal{B}$ sets $V = (v, v^a, \dots, v^{a^{n-1}})$, $D = ((d_{1,0}, d_{1,1}), \dots, (d'_{n,0}, d'_{n,1}))$, k, k' as in Hyb , and constructs circuits $C_0 = \text{PuncturedKey}'_{V,w,D,u,x^*}$ and $C_1 = \text{PuncturedKeyAlt}_{k,k',u,x^*}$. It sends C_0, C_1 to the $i\mathcal{O}$ challenger, and receives $K_{x^*} \leftarrow i\mathcal{O}(\lambda, C_b)$. $\mathcal{B}$ computes y_0 , chooses y_1, β , sends (K_{x^*}, y_β) to $\mathcal{A}$ and receives β' in response. If $\beta = \beta'$, $\mathcal{B}$ outputs 0, else outputs 1.

In order to complete this proof, we show that circuits $\text{PuncturedKey}'_{V,w,D,u,x^*}$ and $\text{PuncturedKeyAlt}_{k,k',u,x^*}$ have identical functionality.

For any $x \in \{0, 1\}^\ell$ such that $P_u(x) = 1$, $h(x) = b_1 \dots b_n$,

$$\text{PuncturedKeyAlt}_{k,k',u,x^*}(x) = v^{\prod_j d_{j,b_j}} = \left(v^{a^{r_u(x)}}\right)^{\prod_j d'_{j,b_j}} = \text{PuncturedKey}'_{V,w,D,u,x^*}(x).$$

For any $x \in \{0, 1\}^\ell$ such that $P_u(x) = 0$, $x \neq x^*$, $h(x) = b_1 \dots b_n$,

$$\text{PuncturedKey}_{k,k',u,x^*}(x) = w'^{\prod_j d_{j,b_j}} = \left(w'^{a^n}\right)^{\prod_j d'_{j,b_j}} = w^{\prod_j d'_{j,b_j}} = \text{PuncturedKey}'_{V,w,D,u,x^*}(x).$$

For $x = x^*$, both programs outputs $\perp$. Therefore, the two programs are functionally equivalent. Hence, $\mathcal{B}$ breaks the security of $i\mathcal{O}$ with advantage ϵ .

Proof of (b): Note that in both Hyb_1 and Hyb_2 , $d_{i,b}$ is a uniformly random element in $\mathbb{Z}_N$ (since $a \in \mathbb{Z}_N^*$). Also, if $w \leftarrow \mathbb{G}_q$, then w^{1/a^n} is a uniformly random element in $\mathbb{G}_q$. From these two observations, it follows that the two experiments Hyb_1 and Hyb_2 are the same.

Proof of (c): Game 1E is similar to Hyb_2 , except that the challenger chooses k, k' differently. In Game 1E, the challenger chooses $v \leftarrow \mathbb{G}_p, w \leftarrow \mathbb{G}_q, d_{i,b} \leftarrow \mathbb{Z}_N$. It sets $d_{i,b,p} = d_{i,b} \bmod p$, $d_{i,b,q} = d_{i,b} \bmod q$, $k = (v, ((d_{1,0,p}, d_{1,1,p}), \dots, (d_{n,0,p}, d_{n,1,p})))$ and $k' = (w, ((d_{1,0,q}, d_{1,1,q}), \dots, (d_{n,0,q}, d_{n,1,q})))$. Suppose there exists a PPT adversary $\mathcal{A}$ such that $\text{Adv}_{\mathcal{A}}^{\text{Hyb}_2} - \text{Adv}_{\mathcal{A}}^{1E} = \epsilon$. Consider the following PPT algorithm $\mathcal{B}$ that uses $\mathcal{A}$ to break the security of $i\mathcal{O}$. $\mathcal{B}$ chooses $v, w, \{d_{i,b}\}$ and sets $k = (v, ((d_{1,0}, d_{1,1}), \dots, (d_{n,0}, d_{n,1})))$, $k' = (w, ((d_{1,0}, d_{1,1}), \dots, (d_{n,0}, d_{n,1})))$, $\tilde{k} = (v, ((d_{1,0,p}, d_{1,1,p}), \dots, (d_{n,0,p}, d_{n,1,p})))$ and $\tilde{k}' = (w, ((d_{1,0,q}, d_{1,1,q}), \dots, (d_{n,0,q}, d_{n,1,q})))$. $\mathcal{B}$ uses the constants v and $\{d_{i,b}\}_{i,b}$ to respond to $\mathcal{A}$'s evaluation queries. On receiving challenge query x^* , $\mathcal{B}$ constructs circuits $C_0 = \text{PuncturedKeyAlt}_{k,k',u,x^*}$ and $C_1 = \text{PuncturedKeyAlt}_{\tilde{k},\tilde{k}',u,x^*}$ and sends C_0, C_1 to the $i\mathcal{O}$ challenger. It receives K_{x^*} in response. $\mathcal{B}$ computes y_0, y_1 , chooses $\beta \leftarrow \{0, 1\}$ and sends (K_{x^*}, y_β) to $\mathcal{A}$, and receives β' in response. If $\beta = \beta'$, then $\mathcal{B}$ outputs 0, else it outputs 1.

Since $v \in \mathbb{G}_p$, $v^{\prod_j d_{j,b_j}} = v^{\prod_j d_{j,b_j,p}}$ for any sequence $b_1 \dots b_n$. Similarly, since $w \in \mathbb{G}_q$, $w^{\prod_j d_{j,b_j}} = w^{\prod_j d_{j,b_j,q}}$. Therefore the circuits C_0, C_1 have identical functionality. Hence, $\mathcal{B}$ breaks the security of $i\mathcal{O}$ with advantage ϵ . ■

Claim A.5. For any adversary $\mathcal{A}$, $\text{Adv}_{\mathcal{A}}^{1F} = \text{Adv}_{\mathcal{A}}^{1E}$.

Proof. We will show that Game 1E and Game 1F are identical, and therefore, any adversary has the same advantage in both games. Note that the only difference between Game 1E and Game 1F is the manner in which the keys k, k' are chosen. In Game 1E, the challenger chooses $v \in \mathbb{G}_p$, $w \in \mathbb{G}_q$, $d_{i,b} \in \mathbb{Z}_N$ and sets $k = (v, ((d_{1,0,p}, d_{1,1,p}), \dots, (d_{n,0,p}, d_{n,1,p})))$ and $k' = (w, ((d_{1,0,q}, d_{1,1,q}), \dots, (d_{n,0,q}, d_{n,1,q})))$. In Game 1F, the challenger chooses $d_{i,b,p} \leftarrow \mathbb{Z}_p$, $e_{i,b,q} \leftarrow \mathbb{Z}_q$ and sets $k = (v, ((d_{1,0,p}, d_{1,1,p}), \dots, (d_{n,0,p}, d_{n,1,p})))$ and $k' = (w, ((e_{1,0,q}, e_{1,1,q}), \dots, (e_{n,0,q}, e_{n,1,q})))$. Using Chinese Remainder Theorem, it follows that $\{(x \bmod p, x \bmod q) : x \leftarrow \mathbb{Z}_N\} \equiv \{(x, y) : x \leftarrow \mathbb{Z}_p, y \leftarrow \mathbb{Z}_q\}$, and hence, Game 1E and Game 1F are identical. ■

Claim A.6. If there exists a PPT adversary $\mathcal{A}$ such that $\text{Adv}_{\mathcal{A}}^{1F} - \text{Adv}_{\mathcal{A}}^{1G}$ is non-negligible in λ , then there exists a PPT adversary $\mathcal{B}$ that breaks the security of $i\mathcal{O}$ with advantage non-negligible in λ .

Proof. The proof of this claim is similar to the proof of Claim A.4, part (c). ■

Claim A.7. If there exists a PPT adversary $\mathcal{A}$ such that $\text{Adv}_{\mathcal{A}}^{1G} - \text{Adv}_{\mathcal{A}}^2$ is non-negligible in λ , then there exists a PPT adversary $\mathcal{B}$ that breaks Assumption 1 with advantage non-negligible in λ .

Proof. The proof of this claim is similar to the one for Claim A.3. ■

B Reducing Assumption 2 to Subgroup Hiding Assumption in Composite Order DDH-Hard Groups

Chase and Meiklejohn [CM14] showed that in bilinear groups of composite order, many q -type reductions are implied by subgroup hiding. The paper also states that a similar result holds in composite order DDH-hard groups. For completeness, we include a proof of this implication.

Theorem B.1. Let $(N, p, q, \mathbb{G}, \mathbb{G}_p, \mathbb{G}_q, g_1, g_2) \leftarrow \mathcal{G}(1^\lambda)$. Then, Assumption 1 implies Assumption 2.

As in [CM14], the proof of this theorem uses the *dual system* technique introduced by Waters [Wat09]. First, we define $4n + 1$ hybrid experiments **Game 1**, **Game 1₁**, **Game 1₂**, **Game 1₃**, ..., **Game $n - 1$** , **Game $n - 1_1$** , **Game $n - 1_2$** , **Game $n - 1_3$** , **Game n** and then show that (a) no PPT adversary can distinguish between consecutive hybrid experiments (b) any adversary has negligible advantage in **Game n** .

B.1 Sequence of Games

We underline the changes from one game to the next.

Game 0 : Choose $v \leftarrow \mathbb{G}$, $a \leftarrow \mathbb{Z}_N$, $\beta \in \{0, 1\}$ and set $T_0 = v^{a^n}$, $T_1 \leftarrow \mathbb{G}$.
Output $(N, \mathbb{G}, \mathbb{G}_p, \mathbb{G}_q, g_1, g_2, v, v^a, \dots, v^{a^{n-1}}, T_\beta)$.

Game i : Choose $v_1 \leftarrow \mathbb{G}_p$, $a_1, \dots, a_i, r_1, \dots, r_i \leftarrow \mathbb{Z}_N$, $\beta \in \{0, 1\}$ and set $T_0 = v_1^{\sum_{j=1}^i r_j a_j^n}$, $T_1 \leftarrow \mathbb{G}$.
Output $(N, \mathbb{G}, \mathbb{G}_p, \mathbb{G}_q, g_1, g_2, \underline{v_1^{\sum_{j=1}^i r_j}}, \underline{v_1^{\sum_{j=1}^i r_j a_j}}, \dots, \underline{v_1^{\sum_{j=1}^i r_j a_j^{n-1}}}, T_\beta)$.

Game i_1 : Choose $v_1 \leftarrow \mathbb{G}_p$, $v_2 \leftarrow \mathbb{G}_q$, $a_1, \dots, a_i, r_1, \dots, r_i \leftarrow \mathbb{Z}_N$, $\beta \in \{0, 1\}$ and set $T_0 = v_1^{\sum_{j=1}^i r_j a_j^n} v_2^{a_i^n}$, $T_1 \leftarrow \mathbb{G}$.

Output $(N, \mathbb{G}, \mathbb{G}_p, \mathbb{G}_q, g_1, g_2, v_1^{\sum_{j=1}^i r_j} v_2, v_1^{\sum_{j=1}^i r_j a_j} v_2^{a_i}, \dots, v_1^{\sum_{j=1}^i r_j a_j^{n-1}} v_2^{a_i^{n-1}}, T_\beta)$

Game i_2 : Choose $v_1 \leftarrow \mathbb{G}_p$, $v_2 \leftarrow \mathbb{G}_q$, $a_1, \dots, a_i, a_{i+1}, r_1, \dots, r_i \leftarrow \mathbb{Z}_N$, $\beta \in \{0, 1\}$ and set $T_0 = v_1^{\sum_{j=1}^i r_j a_j^n} v_2^{a_{i+1}^n}$, $T_1 \leftarrow \mathbb{G}$.

Output $(N, \mathbb{G}, \mathbb{G}_p, \mathbb{G}_q, g_1, g_2, v_1^{\sum_{j=1}^i r_j} v_2, v_1^{\sum_{j=1}^i r_j a_j} v_2^{a_{i+1}}, \dots, v_1^{\sum_{j=1}^i r_j a_j^{n-1}} v_2^{a_{i+1}^{n-1}}, T_\beta)$.

Game i_3 : Choose $v_1 \leftarrow \mathbb{G}_p$, $v_2 \leftarrow \mathbb{G}_q$, $a_1, \dots, a_i, a_{i+1}, r_1, \dots, r_i, r_{i+1} \leftarrow \mathbb{Z}_N$, $\beta \in \{0, 1\}$ and set $T_0 = v_1^{\sum_{j=1}^{i+1} r_j a_j^n} v_2^{a_{i+1}^n}$, $T_1 \leftarrow \mathbb{G}$.

Output $(N, \mathbb{G}, \mathbb{G}_p, \mathbb{G}_q, g_1, g_2, v_1^{\sum_{j=1}^{i+1} r_j} v_2, v_1^{\sum_{j=1}^{i+1} r_j a_j} v_2^{a_{i+1}}, \dots, v_1^{\sum_{j=1}^{i+1} r_j a_j^{n-1}} v_2^{a_{i+1}^{n-1}}, T_\beta)$.

B.2 Adversary's Advantage in these Games

Let $\text{Adv}_{\mathcal{A}}^{1\alpha}$ denote the advantage of adversary $\mathcal{A}$ in Game α . Note that in Game 0, $a \leftarrow \mathbb{Z}_N$, while in Assumption 2, $a \leftarrow \mathbb{Z}_N^*$. This results in security loss that is negligible in λ .

Claim B.1. If there exists a PPT adversary $\mathcal{A}$ such that $\text{Adv}_{\mathcal{A}}^{10} - \text{Adv}_{\mathcal{A}}^{11} = \epsilon$, then there exists a PPT algorithm $\mathcal{B}$ that can distinguish between a random element of $\mathbb{G}$ and a random element of $\mathbb{G}_p$ with advantage ϵ .

Proof. The only difference between the two games is that in Game 0, the challenger chooses $v \leftarrow \mathbb{G}$, while in Game 1, the challenger chooses $v_1 \leftarrow \mathbb{G}_p$. $\mathcal{B}$ receives $N, \mathbb{G}, \mathbb{G}_p, \mathbb{G}_q, g_1, g_2$ and w from the challenger, where $w \leftarrow \mathbb{G}$ or $w \leftarrow \mathbb{G}_p$. $\mathcal{B}$ sets $v = w$, chooses $a \in \mathbb{Z}_N$, $\beta \in \{0, 1\}$ and sets $T_0 = v^{a^n}$, $T_1 \leftarrow \mathbb{G}$. It sends $(N, \mathbb{G}, \mathbb{G}_p, \mathbb{G}_q, g_1, g_2, v, v^a, \dots, v^{a^{n-1}}, T_\beta)$ to $\mathcal{A}$ and receives β' in response. If $\beta' = \beta$, $\mathcal{B}$ sends 1 to the challenger (indicating that $w \in \mathbb{G}$), else it sends 0. Clearly, if $w \in \mathbb{G}$, then this corresponds to Game 0, else it corresponds to Game 1. If $\mathcal{A}$ wins Game 0 with probability p_0 , and wins Game 1 with probability p_1 , then the advantage of $\mathcal{B}$ in breaking Assumption 1 is $p_0 - p_1 = \epsilon$. $\blacksquare$

Claim B.2. If there exists a PPT adversary $\mathcal{A}$ such that $\text{Adv}_{\mathcal{A}}^{1i} - \text{Adv}_{\mathcal{A}}^{1i_1} = \epsilon$, then there exists a PPT algorithm $\mathcal{B}$ that can distinguish between a random element of $\mathbb{G}$ and a random element of $\mathbb{G}_p$ with advantage ϵ .

Proof. The PPT algorithm $\mathcal{B}$ is defined as follows. First, $\mathcal{B}$ receives $N, \mathbb{G}, \mathbb{G}_p, \mathbb{G}_q, g_1, g_2$ and w from the challenger, where $w \leftarrow \mathbb{G}$ or $w \leftarrow \mathbb{G}_p$. $\mathcal{B}$ chooses $v' \leftarrow \mathbb{G}_p$, $a_1, \dots, a_{i-1}, r_1, \dots, r_{i-1} \leftarrow \mathbb{Z}_N$, $\beta \leftarrow \{0, 1\}$ and sets $T_0 = v'^{\sum_{j=1}^{i-1} r_j a_j^n} w^{a_i^n}$ and $T_1 \leftarrow \mathbb{G}$. It sends $(N, \mathbb{G}, \mathbb{G}_p, \mathbb{G}_q, g_1, g_2, v'^{\sum_{j=1}^{i-1} r_j} w, v'^{\sum_{j=1}^{i-1} r_j a_j} w^{a_i}, \dots, v'^{\sum_{j=1}^{i-1} r_j a_j^{n-1}} w^{a_i^{n-1}}, T_\beta)$ to $\mathcal{A}$ and receives β' in response. If $\beta' = \beta$, it sends 0 to the challenger, else it sends 1.

If $w \in \mathbb{G}$, then $w = v'^r v_2$ for some $r \in \mathbb{Z}_N, v_2 \in \mathbb{G}_q$. If $w \in \mathbb{G}_p$, then $w = v'^r$ for some $r \in \mathbb{Z}_N$. Therefore, $\mathcal{B}$ implicitly sets $r_i = r$, and hence, if $w \in \mathbb{G}_p$, $\mathcal{A}$ receives the challenge as per Game i , otherwise it receives the challenge as per Game i_1 . $\blacksquare$

Claim B.3. For any adversary $\mathcal{A}$, $\text{Adv}_{\mathcal{A}}^{1i_1} = \text{Adv}_{\mathcal{A}}^{1i_2}$.

Proof. This step is information-theoretic. First, let us recall the Chinese Remainder Theorem. For distinct primes p, q and $N = pq$, we have

$$\{(x \bmod p, x \bmod q) : x \leftarrow \mathbb{Z}_N\} \equiv \{(x \bmod p, y \bmod q) : x \leftarrow \mathbb{Z}_N, y \leftarrow \mathbb{Z}_N\}.$$

Hence, it follows that distributions D_1 and D_2 are identical, where

$$D_1 = \left\{ \sum_{j=1}^i r_j a_j \mod p, \dots, \sum_{j=1}^i r_j a_j^n \mod p, a_i \mod q : r_1, \dots, r_i, a_1, \dots, a_i \leftarrow \mathbb{Z}_N \right\}$$

$$D_2 = \left\{ \sum_{j=1}^i r_j a_j \mod p, \dots, \sum_{j=1}^i r_j a_j^n \mod p, a_{i+1} \mod q : r_1, \dots, r_i, a_1, \dots, a_i, a_{i+1} \leftarrow \mathbb{Z}_N \right\}.$$

Note that since $v \in \mathbb{G}_p$, $v^x = v^{x \mod p}$. Similarly, $w^y = w^{y \mod q}$. Hence it follows that any adversary has identical advantage in **Game** i_1 and **Game** i_2 . ■

Claim B.4. If there exists a PPT adversary $\mathcal{A}$ such that $\text{Adv}_{\mathcal{A}}^{1i_2} - \text{Adv}_{\mathcal{A}}^{1i_3} = \epsilon$, then there exists a PPT algorithm $\mathcal{B}$ that can distinguish between a random element of $\mathbb{G}$ and a random element of $\mathbb{G}_q$ with advantage ϵ .

Proof. The PPT algorithm $\mathcal{B}$ is defined as follows. First, $\mathcal{B}$ receives $N, \mathbb{G}, \mathbb{G}_p, \mathbb{G}_q, g_1, g_2$ and w from the challenger, where $w \leftarrow \mathbb{G}$ or $w \leftarrow \mathbb{G}_q$. $\mathcal{B}$ chooses $v' \leftarrow \mathbb{G}_p$, $a_1, \dots, a_i, r_1, \dots, r_i \leftarrow \mathbb{Z}_N$, $\beta \leftarrow \{0, 1\}$ and sets $T_0 = v'^{\sum_{j=1}^i r_j a_j^n} w^{a_{i+1}}$ and $T_1 \leftarrow \mathbb{G}$. It sends $(N, \mathbb{G}, \mathbb{G}_p, \mathbb{G}_q, g_1, g_2, v'^{\sum_{j=1}^i r_j} w, v'^{\sum_{j=1}^i r_j a_j} w^{a_{i+1}}, \dots, v'^{\sum_{j=1}^i r_j a_j^{n-1}} w^{a_{i+1}^{n-1}}, T_\beta)$ to $\mathcal{A}$ and receives β' in response. If $\beta' = \beta$, it sends 0 to the challenger (i.e. $\mathcal{B}$ guesses that $w \leftarrow \mathbb{G}_q$), else it sends 1.

As in the proof of B.2, if $w \in \mathbb{G}$, then $w = v'^r v_2$, else $w = v_2$ for some r, v_2 . $\mathcal{B}$ therefore implicitly sets $r_{i+1} = r$. If $w \in \mathbb{G}_q$, then $\mathcal{A}$ gets a **Game** i_2 challenge, else a **Game** i_3 challenge. ■

Claim B.5. If there exists a PPT adversary $\mathcal{A}$ such that $\text{Adv}_{\mathcal{A}}^{1i_3} - \text{Adv}_{\mathcal{A}}^{1i+1} = \epsilon$, then there exists a PPT algorithm $\mathcal{B}$ that can distinguish between a random element of $\mathbb{G}$ and a random element of $\mathbb{G}_q$ with advantage ϵ .

Proof. The proof for this step is similar to the proof of Claim B.4. ■

Claim B.6. For any adversary $\mathcal{A}$, $\text{Adv}_{\mathcal{A}}^{1n} \leq \text{negl}(\lambda)$.

Proof. Let us consider the following distributions (all operations are modulo p):

$$D_1 = \left\{ \left(\sum_{j=1}^{n+1} r_j, \sum_{j=1}^{n+1} r_j a_j, \dots, \sum_{j=1}^{n+1} r_j a_j^{n-1}, \sum_{j=1}^{n+1} r_j a_j^n \right) : r_1, \dots, r_{n+1}, a_1, \dots, a_{n+1} \leftarrow \mathbb{Z}_p \right\}$$

$$D_2 = \left\{ \left(\sum_{j=1}^{n+1} r_j, \sum_{j=1}^{n+1} r_j a_j, \dots, \sum_{j=1}^n r_j a_j^{n-1}, r \right) : r_1, \dots, r_{n+1}, a_1, \dots, a_{n+1}, r \leftarrow \mathbb{Z}_p \right\}$$

In order to show that $\text{Adv}_{\mathcal{A}}^{1n+1} \leq \text{negl}(\lambda)$, it suffices to show that D_1 and D_2 are statistically indistinguishable. In fact, one can prove the following stronger statement.

Observation B.1. $D_1 \approx_s \{(c_1, \dots, c_{n+1}) : c_1, \dots, c_{n+1} \leftarrow \mathbb{Z}_p\}$

Note that D_1 can also be expressed as

$$\{(r_1 \dots r_{n+1}) \cdot V_{a_1 \dots a_{n+1}} : r_i, a_j \in \mathbb{Z}_p\}$$

where $V_{a_1 \dots a_{n+1}}$ is a Vandermonde matrix of dimension $(n+1) \times (n+1)$.

$$V_{a_1 \dots a_{n+1}} = \begin{bmatrix} 1 & a_1 & \cdots & a_1^n \\ 1 & a_2 & \cdots & a_2^n \\ \vdots & \vdots & \ddots & \vdots \\ 1 & a_{n+1} & \cdots & a_{n+1}^n \end{bmatrix}$$

If all a_i s are distinct and non-zero, then $V_{a_1 \dots a_{n+1}}$ is invertible. Hence, if the entries a_i are chosen uniformly at random from $\mathbb{Z}_p$, then with overwhelming probability, $V_{a_1 \dots a_n}$ is invertible. Thus, there is a bijection between $\{(r_1, \dots, r_{n+1}) \cdot V : r_i \in \mathbb{Z}_p\}$ and $\mathbb{Z}_p^{n+1}$. This proves our observation. ■

CloudSourcing Cryptography: Automating the Generation of Outsourced Cryptographic Algorithms*

J Ayo Akinyele

Matthew Pagano

Matthew Green

Avi Rubin

ABSTRACT

We present CloudSource, an automated tool for developing “outsourcing-ready” cryptographic schemes. CloudSource is designed to programmatically analyze existing pairing-based encryption schemes, and to derive new algorithms that allow users to outsource portions of the cryptographic work to an untrusted server.

The CloudSource tool addresses a growing need, as we increasingly deploy cryptography in environments where users employ limited mobile devices and yet have ready access to cloud-based computing resources. The techniques we propose form part of a growing line of work aimed towards the automated analysis and engineering of cryptographic protocols, and will help to reduce the need for manual optimization of these constructions.

1. INTRODUCTION

Over the past several years, a number of new cryptographic technologies have entered the literature. These technologies include improvements in existing schemes as well as entirely new paradigms such as Identity-Based Encryption [11,42], Functional Encryption [40] and Fully Homomorphic Encryption [26]. As a result, we now have entirely new capabilities that were not possible using earlier techniques.

While these technologies offer us new tools to solve long-standing security problems, they have serious limitations that have prevented their use in practice. One notable limitation is the fact that constructions often need to be tailored to a specific application before they can be used. The application may have specific requirements, *e.g.*, related to the efficiency and bandwidth overhead of the scheme. A surprising fraction of the cryptographic literature is devoted to detailing precisely these sorts of optimizations, from basic scheme optimizations (*e.g.*, re-writing cryptographic oper-

ations to use different operations) [17, 38, 39] to advanced techniques such as secure outsourcing of cryptographic operations [28, 33, 51] and batch processing [2, 13, 24, 50].

Unfortunately, it is not possible for human beings to customize all constructions to every possible application. Moreover, safely applying these techniques is typically a non-trivial process; even domain experts have been known to make important mistakes that can completely undermine the security of a construction, *e.g.*, [15, 43]. For most implementers it is simply not practical to obtain the resources to make these changes, and as a consequence, many schemes in the literature remain undeployed.

In this paper we continue a line of research that investigates whether *automated* techniques can be developed for the purpose of re-designing and implementing cryptographic schemes [2, 3, 5, 37]. The general approach in this work is to take an existing, provably-secure cryptographic scheme as input, and to apply a series of techniques in order to arrive at a new, secure cryptographic scheme. The challenge in this work is to identify useful transformations that preserve security properties, and to develop tools that can safely implement these techniques.

Outsourcing cryptography. In this work we focus on the specific problem of outsourcing cryptographic operations to untrusted parties. This is an area that has received a great deal of attention due to the increasing availability of cloud computing resources. The benefits of outsourcing are many: first, it can greatly reduce the computational burden of cryptographic operations, something that is particularly relevant to mobile devices (where complex decryption operations on *e.g.*, attribute-based encryption ciphertexts can take many seconds [28]). Second, it can dramatically reduce bandwidth overhead for a decryptor by reducing the size of complex ciphertexts into a shorter “partially decrypted” ciphertext. Moreover, implementers may be able to reduce the size of a Trusted Codebase (or the complexity of a hardware device like a smartcard) by outsourcing complex operations to untrusted hardware [28].

While general techniques exist for the purpose of securely outsourcing *arbitrary* computations, *e.g.*, [26], these schemes are not yet practical enough for the applications we envision – *i.e.*, offloading meaningful cryptographic work to a third party. However, a separate line of work has considered more practical outsourcing schemes for *specific* cryptographic constructions [16, 28, 33, 34, 36, 51]. However, each of these proposals addresses only one individual scheme. This is unfortunate, since developing outsourcing-ready encryption schemes from scratch is a challenging and error-prone

*This work was partially supported by DARPA and the Air Force Research Laboratory (AFRL) under contract FA8750-11-2-0211. This document is a pre-print for DARPA and not for public distribution.

task. Fortunately, the previous work of [28, 34] indicates that it is possible to *modify* certain classes of pairing-based encryption scheme into outsourcing-ready schemes without damaging the underlying construction. While these techniques are applied on a per-scheme basis, we believe that they are also amenable to automated techniques.

Our contribution. In this work we develop a new technique for automatically generating *outsourcing-ready* encryption schemes for a large class of pairing-based cryptographic constructions. Our technique, which we call CloudSource, allows us to begin with the description of a secure encryption scheme in the domain-specific Scheme Description Language (SDL) [2], then to apply a series of programmatic transformations in order to arrive at a new scheme with outsourcing capability. Once processed, we can readily evaluate the efficiency of the scheme and even translate it into a working C++ or Python implementation. To the best of our knowledge, this is the first such generalization of outsourcing techniques. We tested CloudSource on 9 different pairing-based encryption algorithms from the academic literature to demonstrate the ability of our tool to produce an outsourced scheme and executable code suitable for mobile devices with limited processing power.

The most important aspect of our work is that we are able to retain the security guarantees of the original construction, even in the face of a possible adversarial outsourcing service. To ensure this, we apply a series of blinding rules according to a formula that preserves certain security properties of the underlying scheme.

Overview of our techniques. We now provide an overview of our approach. Without loss of generality we will use the example of public-key encryption, but note that our approach can be used with Identity-Based encryption [10, 11, 14, 31, 44], Attribute-Based Encryption [32, 40, 45] Broadcast Encryption [12, 23], and Functional Encryption [30, 49], along with many other scheme types.

Let $\Pi = (\text{KeyGen}, \text{Enc}, \text{Dec})$ be a semantically-secure secure encryption scheme. Our goal is to construct three new algorithms: **KeygenOut**, **Transform** and **Decout**. The new **KeygenOut** algorithm takes the same inputs as **KeyGen**, but produces a *pair* of decryption keys: the **Transform Key** (TK), which can be delivered to an untrusted outsourcing party, and **Secret Key** (SK) which is held secret. Intuitively, it must be possible to deliver TK to an untrusted party who uses it to execute the **Transform** algorithm on a ciphertext. When this process has completed, we use the **Decout** algorithm with SK in order to reveal the plaintext.

The challenge is to develop an approach that guarantees three things. First, the resulting combination of **KeygenOut**, **Transform**, **Decout** is *correct*. Second, that the resulting scheme retains its security, even when the (possibly adversarial) outsourcing party learns TK (but not SK). Finally, we must ensure this without requiring difficult manual security analysis of the resulting scheme.

We draw the intuition for our approach from [28], which explored a known observation that in some pairing-based schemes it is possible to modify the original scheme by adding *blinding factors* to a standard decryption key such that this blinding can be removed following decryption. These modified decryption keys serve admirably in the role of **Transform Key**, while the blinding factors themselves can be retained as SK.

While the key blinding technique provides significant performance benefits to pairing-heavy algorithms, applying the technique generally is non-trivial. For one, it can be tedious. There are multiple factors that determine how to choose blinding values for a given scheme and searching this space by hand is quite daunting. Second, it can be error-prone. Mistakes are easily made when applying blinding exponents in a way that becomes impossible to unblind in the final decryption. Aside from correctness, there is a separate issue of the security of the resulting **Transform** key. In some cases, choosing too few blinding factors for a scheme may result in an insecure **Transform** key that might offer little to no protection in practice, while choosing *too many* can lead to an inefficient outsourcing scheme. Generalizing the process requires us to carefully balance efficiency of key blinding while preserving the security of the blinded keys.

1.1 Our Approach

Figure 1 provides a brief overview of the techniques used in CloudSource. At a high level, CloudSource is designed to analyze a scheme, extract the key generation and decryption equations, and derive working code for the modified **Keygen**, **Transform** and final decryption (**KeygenOut**, **Transform**, **Decout**) algorithms. This involves two main components:

1. The CloudSource tool, which takes as input an SDL file describing an encryption scheme. It applies a series of programmatic transformations and derives the outsourced encryption scheme. The output of this is a second SDL file containing the modified **KeygenOut**, **Transform** and **Decout** algorithms.
2. A Code Generator, which takes the output of the CloudSource tool and generates working source code to implement the outsourced encryption scheme. The user can choose either Python or C++ as the output language.

The CloudSource tool outputs a human-readable file written in LaTeX describing the new algorithms, and containing a security proof of the construction that these maintain the semantic security of the original scheme. The following sections describe the architecture in detail.

2. PRELIMINARIES

Before we give details of our approach, we first provide some background on bilinear maps and discuss the tools and techniques utilized in developing the CloudSource tool. We also provide definitions for the type of cryptographic schemes we address in this work.

A bilinear map (or pairing) is an efficiently computable mapping $e : \mathbb{G}_1 \times \mathbb{G}_2 \rightarrow \mathbb{G}_T$ over three multiplicative cyclic groups $\mathbb{G}_1, \mathbb{G}_2$ and $\mathbb{G}_T$ of prime order, p . Pairings have two central properties. The first is bilinearity: let $\langle g_1 \rangle = \mathbb{G}_1, \langle g_2 \rangle = \mathbb{G}_2$ and $a, b \in \mathbb{Z}_p$, it holds that $e(g_1^a, g_2^b) = e(g_1, g_2)^{ab}$. The second property is non-degeneracy, which guarantees that $e(g_1, g_2) \neq 1$. Pairing-friendly groups come in two forms: *symmetric groups*, where $\mathbb{G}_1 = \mathbb{G}_2$ (or can be effectively due to an efficiently-computable isomorphism), and *asymmetric groups* where $\mathbb{G}_1 \neq \mathbb{G}_2$.

As a starting point for our automation, we rely on a high-level description of the encryption scheme in a domain-specific SDL. SDL was first introduced as part of the AutoBatch framework [2] to abstractly represent various types of

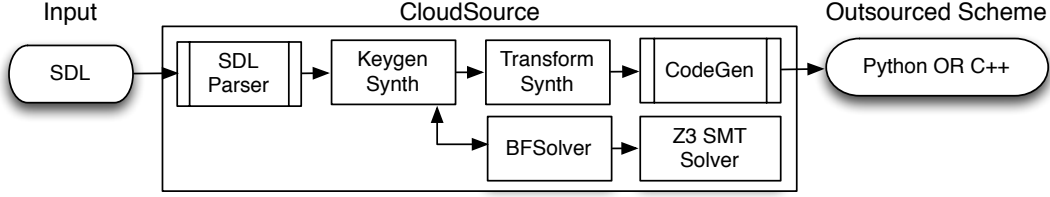


Figure 1: The flow of CloudSource. CloudSource takes as input a high-level description of an encryption algorithm in SDL. It processes the SDL for certain metadata. In particular, KeygenSynth analyzes the inputs and observes how the secret key is constructed in the Keygen algorithm. This information is passed to the BFSolver which utilizes an SMT solver to determine how the user’s decryption key should be protected. BFSolver returns a mapping of secret key variables to blinding factors needed to construct a Transform key securely. KeygenSynth uses the mapping to construct the outsourced keygen algorithm in SDL. Similarly, the TransformSynth components use the mapping to construct the computation of the cloud in Transform and the client in Decout. CloudSource produces a new SDL file of the outsourced algorithm and utilizes the code generator to produce executable code in Python or C++.

pairing-based signature schemes. We extend it to support encryption.

Boolean Satisfiability (or SAT) is the problem of determining whether boolean assignments exist for variables in a logical formula such that the formula is evaluated to *true*. Satisfiability Modulo Theories (SMT) problem is a decision problem for first-order logical formulas. SMT is a generalization of SAT to support richer logic such as with arithmetic, bit-vectors, quantifiers, arrays, and several other useful first-order theories [22]. High-performing state-of-the-art SMT Solvers (*e.g.*, Yices, Z3, CVC, MathSAT) have been developed over the years to address these problems. In practice, these tools form the building block for a variety of verification tools to perform bounded model checking, symbolic execution, static program analysis, and various constraint-satisfaction problems in computer science. Similarly, we utilize SMT solvers as a core building block of the CloudSource framework.

3. THE CLOUDSOURCE TOOLCHAIN

Overview of the approach.

At a high-level, CloudSource works by altering the structure of a scheme decryption key, adding blinding factors to the key such that an untrusted party can *partially* decrypt a ciphertext using the blinded key. This procedure forms the backbone of our new Transform algorithm, and the blinded key itself is the Transform key. We require that the structure of the key remain sufficiently intact that the original plaintext can be recovered from the output of Transform, given knowledge of the blinding factors.

We will now give an example of how the process works. For consistency with previous work, we will use the example of the Waters Ciphertext-Policy ABE scheme, which was manually outsourced by Green, Hohenberger and Waters [28]. For a given set of attributes S , the Waters scheme produces a master secret g^α and uses this to generate decryption keys of the following form. Note that $t \in_R \mathbb{Z}_q$ and g, g^a, S and the function F are public:

$$K = g^\alpha \cdot g^{at}, L = g^t, \{K_x = F(x)^t\}_{x \in S}$$

The intuition behind the blinding process stems from the observation that of all the elements referenced above, only

the value g^α is actually a secret. In principle, any party can, without access to the master secret value, select a random α' and generate a *random* pseudo-key $(K', L', \{K'_x\})$ with this value. Since the resulting key can be generated by any party it clearly offers no benefit to the attacker.

This observation alone would be of limited use, except for the observation that one can *blind* a valid key $(K, L, \{K_x\})$ such that it has the same distribution as the above pseudo-key. This is accomplished by selecting a random $z \in \mathbb{Z}_q$ and computing:

$$(K', L', \{K'_x\}) = (K^z, L^z, \{K_x^z\})$$

If one expresses $\alpha' = (\alpha \cdot z)$ and $t' = (t \cdot z)$ then clearly the distribution of the new key is identical to the pseudo-key described above. However, the important aspect of this scheme is that the blinding factor z can actually be removed following the decryption procedure, giving the identical result as if the original decryption key had been used on a ciphertext.

Of course, the above scheme (from the work of Green *et al.*) proves to be a relatively simple example. This is because the master secret contains only a single secret element, whereas many modern encryption schemes can contain larger secrets. Moreover, when multiple secrets are used together within a given term, it can be difficult to repeat the process above. Of course, one solution is to simply apply a separate blinding factor to each element of the key – however, this can prove tricky to invert, unless each element is treated separately as part of the resulting partially-decrypted ciphertext.

The challenge in our approach is therefore threefold: (1) to identify which elements of a valid decryption key contain secret values that can be simulated (*i.e.*, replaced with non-secret values via blinding), (2) to determine whether this blinding can survive the decryption process and be inverted, and (3) to minimize the number of blinding factors and values that must be returned by the Transform algorithm. We accomplish these tasks using the following components.

1. **SDL Parser.** This component parses the SDL into an intermediate representation, then uses this representation to identify the key generation and decryption algorithms.

2. **KeyGen Synthesizer.** Once the scheme has been parsed, this module analyzes the scheme to identify the Key Generation and Decryption algorithms, decomposes the secret keys, then traces the components of these keys to determine which are secret, which are public, and which are generated randomly.
3. **BFSolver.** Once the key has been successfully decomposed, this portion of the toolchain employs an SMT solver to determine how to blind the individual components of the key to ensure that (a) all secret key elements are correctly blinded, and (b) the minimal number of blinding factors is used (in order to ensure the most efficient algorithm).
4. **Transform Synthesizer.** Given an assignment of blinding factors, this module generates the new **KeygenOut**, **Transform** and **Decout** algorithms to produce the final outsourcing equations in an SDL representation.
5. **Code Generator.** Users can output the resulting scheme as an SDL file, or they can optionally convert the scheme into a working implementation in Python or C++ using the Code Generator component.

We now expand on each of the above components.

SDL Parser.

The input to our toolchain is a cryptosystem written in SDL, which was initially developed by Akinyele et al. [2] and we extend the SDL parser for CloudSource. The parser component processes a given SDL file, converting each line (a string) into a binary-tree node with all of the relevant information. In addition, the parser collects the following metadata:

1. **Types of all variables.** SDL is a typed language but we relieve the SDL designer by automatically inferring some types based on the computation. This information is used for type checking and for automatically generating C++ code (a statically-typed language).
2. **Variable dependency list.** This is a list of all variables upon which a given variable depends for its value. For example, if we have the two statements $x = a + b$ and $y = x + c$, the variable dependency list for y would be x , c , a , and b .
3. **Variable influence list.** This is a list of variables whose values are influenced by a given variable. If we consider the example above, the variable influence list of a is x and y . Both the variable dependency list and variable influence list are useful in helping us map relationships between variables.

Additionally, this first phase of the process collects a list of all variables referenced in a given function and distinguishes between public and secret variables in the **KeyGen** function. In addition, the SDL parser performs type checking as it reads in each line of SDL to ensure that all mathematical rules are followed. For example, the type checker ensures that the input to pairings has the right group type (based on whether the setting is symmetric or asymmetric); arguments to addition, subtraction, multiplication, and division are of the same group type; etc.

KeyGen Synthesizer.

The **KeygenSynth** component is responsible for producing SDL code to blind each secret-key element with the proper blinding factor. **KeygenSynth** first extracts all master secrets and secret-key elements from the user's SDL file by examining the output of the **Setup** and **KeyGen** routines. Next, for each secret-key element, **KeygenSynth** extracts all exponents used in the calculation of the element, as well as the relationship between exponents. To ensure that all exponents in the element's calculation are extracted (even across multiple layers of assignments), **KeygenSynth** performs a recursive traversal on the base-element representation of the element, extracting all exponents found during the traversal.

In addition to extracting all exponents, **KeygenSynth** must record the relationship between exponents. This relationship determines how blinding factors are applied to the exponents of a given secret-key element. For example, suppose a secret-key element x is calculated as follows:

$$x = a^b \cdot c^d$$

As in several previous works, our current blinding methodology is to exponentiate the secret-key element to a random element in $\mathbb{Z}_p$. Blinding x by y gives us the following expression:

$$x = a^{by} \cdot c^{dy}$$

Note that it is required that both exponents in the calculation of x (i.e., b and d) be blinded with the blinding factor y . In our input to the SMT solver, we label this as addition between the exponents. Addition requires that both exponents in the calculation be blinded by the blinding factor.

In contrast, suppose x is calculated as follows:

$$x = e^{fg}$$

When we blind x by exponentiating it to y , we can rearrange the blinding factor in either of the following two ways:

$$x = e^{fy} \cdot g \quad \text{or} \quad x = e^f \cdot gy$$

In other words, the blinding factor y can blind either the exponent f or the exponent g . In our input to the SMT solver, we label this as multiplication between the exponents. Multiplication allows either one of the exponents to be blinded by the blinding factor (but not both). We note that while these example expressions are simple, many expressions of secret-key elements in the schemes we have considered are very complex. This calls for an automated approach that is fast and less likely to produce errors, in addition to providing programmatic access to the exponents and relationships produced.

Next, **KeygenSynth** classifies each exponent extracted as either a master-secret element, or a random element generated during the **Keygen** routine. **KeygenSynth** then prepares all of this information for input to **BFSolver** and waits for a response. The response received will be the blinding factor to be associated with each secret-key element. As discussed in Section 3.1, **BFSolver** attempts to reduce the number of blinding factors as much as possible without reducing the security of the outsourced scheme.

Furthermore, our toolchain offers a limited form of term rewriting for easily understanding how a given variable was calculated, regardless of the number of expressions used in the final calculation. It is often necessary to determine all of the expressions used to calculate a variable, going all the way back to the base elements. For example, in determining how best to blind the elements of the secret key, we must first extract all exponents used in the calculations of

the secret-key elements, as well as the relationships between these exponents. A naive approach is to simply find the assignment node of that variable and extract the exponents from the expression. This can lead to an incomplete result in the case where there are multiple expressions that determine a variable's value.

Suppose that we need to extract the exponents and exponent relationships of a variable a , and that the assignment node of a is the following:

$$a = b^x$$

x is the only exponent in this expression, but there may be other exponents that we need to consider. Suppose the variable b is calculated as follows:

$$b = c^y$$

Consequently, we also need to extract the exponent y in identifying the exponents that contribute to the value of a . Furthermore, we need to understand the relationship between the exponents x and y (specifically, the fact that y is exponentiated to x).

To address situations such as this, our toolchain offers a term rewriting engine that rewrites expressions using their base values only. In our current instantiation, we define the base values to be random elements and hashed elements. Our term rewriting engine unfolds all expressions until the base elements are reached. This is performed during the initial parsing of the SDL file. As each assignment node is read in, each variable on the right side of the assignment node is replaced with its base-element form. In the example above, the expression of a would be rewritten as follows: $a := c^{y^x}$ where c , y , and x are all randomly generated or hashed elements. Note that while this example is (purposefully) simple, expressions for elements in modern cryptosystems can become highly complex and dependent on several other elements. Performing this term rewriting manually can become tedious, time-consuming, and error-prone. Furthermore, programmatic access to these base expressions is often required, as in the case of exponent and exponent-relationship extraction. Our term rewriting engine provides base expressions in binary trees that are convenient to navigate programmatically.

BFSolver.

At a high-level, **BFSolver** takes as input a configuration file that consists of the master secret key (MSK) exponents and random exponents selected in **Keygen**. In addition, the file includes the relationship between variables in each secret-key element of the decryption key. As mentioned before, these are extracted by the term writing engine as part of the **KeygenSynth** routine. The **BFSolver** processes these inputs and determines the possible space of blinding factors that can be assigned to the given key according to our blinding rules (see section 3.1 for more details). For instance, one rule states that all master secret variables must have unique blinding factors. We encodes these rules for the given decryption key as a series of constraints over the variables to the SMT solver. These constraints guide the solver in searching for a satisfiable solution that also meets our desired level of security. Once we apply these rules and derive the appropriate constraints, we execute the solver to check for a solution. If a solution is found, **BFSolver** immediately returns a mapping of blinding factors to secret-key elements to **KeygenSynth**.

1 KeygenOut Proof

Let α, a_1 be the MSK variable(s) and $r_1, r_2, z_2, z_1, id, tag_k$ be randomness selected in the **Keygen** algorithm. The **Keygen** algorithm runs to obtain the secret key, $SK = \{D_1, D_2, D_3, D_4, D_5, D_6, D_7, K, tag_k\}$ and is computed as follows:

SK:

$$D_1 = g^{\alpha \cdot a_1} \cdot v^{(r_1+r_2)}, D_2 = g^{-\alpha} \cdot v_1^{(r_1+r_2)} \cdot g^{z_1}, D_3 = g^{b-z_1}, D_4 = v_2^{(r_1+r_2)} \cdot g^{z_2}, D_5 = g^{b-z_2}, \\ D_6 = g^{b \cdot r_2}, D_7 = g^{r_1}, K = (u^{id} \cdot w^{tag_k} \cdot h^{r_1})^{r_1}, tag_k = tag_k$$

The **KeygenOut** algorithm selects blinding factors, $uf_0, bf_0, uf_1 \in \mathbb{Z}_p^*$. Let the new TK be computed as follows:

TK:

$$D'_1 = D_1^{uf_0}, D'_2 = D_2^{bf_0}, D'_3 = D_3^{bf_0}, D'_4 = D_4^{bf_0}, D'_5 = D_5^{bf_0}, D'_6 = D_6^{bf_0}, \\ D'_7 = D_7^{bf_0}, K' = K^{uf_1}, tag'_k = tag_k^{bf_0}$$

Note that a simulator given only public parameters (PK) can formulate a simulated TK_{sim} by randomly selecting $R_1, R_2, R_3, R_4, R_5, R_6, R_7, RK, Rtag_k$ and computing:

TK_{sim} :

$$\bar{D}_1 = R_1^{uf_0}, \bar{D}_2 = R_2^{bf_0}, \bar{D}_3 = R_3^{bf_0}, \bar{D}_4 = R_4^{bf_0}, \bar{D}_5 = R_5^{bf_0}, \bar{D}_6 = R_6^{bf_0}, \\ \bar{D}_7 = R_7^{bf_0}, \bar{K} = RK^{uf_1}, \bar{tag}_k = Rtag_k^{bf_0}$$

Observe that this simulated TK_{sim} has an identical distribution to TK. Let $D'_i = (D_i)^{uf_0}$, $\alpha' = \alpha \cdot bf_0$, $z'_1 = z_1 \cdot bf_0$, $r'_2 = r_2 \cdot bf_0$, $r'_1 = r_1 \cdot bf_0$, $z'_2 = z_2 \cdot bf_0$, $K' = (K)^{uf_1}$, $tag'_k = tag_k \cdot bf_0$ be the new MSK variables and random values selected in **KeygenOut**, and TK is computed as follows:

TK:

$$D'_1 = (g^{\alpha \cdot a_1} \cdot v^{(r_1+r_2)})^{uf_0}, D'_2 = g^{-\alpha'} \cdot v_1^{(r'_1+r'_2)} \cdot g^{z'_1}, D'_3 = g^{b-z'_1}, D'_4 = v_2^{(r'_1+r'_2)} \cdot g^{z'_2}, D'_5 = g^{b-z'_2}, \\ D'_6 = g^{b \cdot r'_2}, D'_7 = g^{r'_1}, K' = (u^{id} \cdot w^{tag_k} \cdot h^{r_1})^{uf_1}, tag'_k = tag_k^{bf_0}$$

Figure 2: A fragment of the proof of security which argues that the **KeygenOut algorithm preserves the security of the secret key. The proof shows that the constructed transform key is identically distributed to a randomly generated pseudo-key.**

Transform Synthesizer.

TransformSynth is responsible for producing a **Transform** and a **Decout** routine in the new outsourced scheme. To ensure correctness and efficiency, **TransformSynth** determines how to divide the computation of the original decryption routine into which components can be performed by the powerful, untrusted **Transform** routine, and which components can be performed by the lightweight, trusted **Decout** routine. There are three stages to **TransformSynth**: pre-processing, the main loop, and post-processing.

TransformSynth is given as input the secret-key element to blinding factor mapping produced by **BFSolver**. During the pre-processing phase, **TransformSynth** sets up the input parameters for the **Transform** and **Decout** routines. The **Transform** routine receives as input the same parameters that the original decryption routine received, which usually consists of the ciphertext and secret key (in the case of the **Transform** routine, the secret key is blinded). During the **Transform** routine, certain values that **Decout** needs are cached and stored in two data structures that are passed to **Decout** as input. One data structure is for cached values outside for loops, and the other is for cached values inside for loops, but the latter is not always sent (indeed, performance is improved when it is not sent). These data structures represent the partially decrypted ciphertext that is fully decrypted in **Decout**. **Decout** also receives as input the blinding factors produced by **KeygenSynth**. Note that **Decout** does not receive the full ciphertext or blinded se-

cret key as input parameters, as these are unnecessary for Decout.

In the main loop, `TransformSynth` loops over the lines of code in the original decryption routine. For each line, `TransformSynth` determines how the computation should be divided between the `Transform` routine and the `Decout` routine. First, the SDL node representing the current line of code is simplified using algorithms developed in previous research by Akinyele et al. [2]. This includes splitting all pairings as much as possible, unrolling dot products if the range values are known, converting all division into multiplication by inverses, and moving exponents inside the pairings. This improves efficiency and our ability to efficiently unblind pairings whose input is blinded. As we discuss later, we wish to organize the pairings into groups of pairings that are blinded by the same blinding factor. Doing so reduces processing time and the size of the data structures passed from `Transform` to `Decout`.

Any line of code that has at least one pairing goes through the following process. We recall from earlier sections on `KeygenSynth` that `BFSolver` returns a mapping between all secret-key elements and the blinding factors associated with each. `KeygenSynth` then blinds each element of the secret key by the appropriate blinding factor by exponentiating the former to the inverse of the latter. The blinded elements of the secret key form the blinded secret key that is passed to the `Transform` routine.

Suppose a line of code from the original decryption routine has at least one pairing in it. The first step is to use the automated techniques developed in `AutoBatch` [2] to combine pairings wherever possible. By combining pairings, we reduce the number of pairings that the `Transform` routine has to perform. This can lead to substantial cost savings due to the resource-intensive nature of pairings. Next, we group together all pairings that have input elements that are blinded by the same blinding factors. These pairings can be safely computed and then multiplied together in `Transform`, and then successfully unblinded in `Decout` by exponentiating the entire pairing product to the common blinding factor. While this does not reduce the processing time required in `Transform`, it does reduce the processing time required in `Decout` (which is more important), as well as reduce the size of the partially decrypted ciphertext that must be passed from `Transform` to `Decout` for `Decout` to fully decrypt (as fewer entries need to be stored there). Rather than naively computing all pairings and storing each individual result to the partially decrypted ciphertext structure, we determine in `TransformSynth` which pairings can be safely multiplied together in the `Transform` routine to save network bandwidth and processing time in `Decout`. It is for this reason that we endeavor to group together as many pairings based on common blinding factors as possible.

If the line of code does not have any pairings, but the line of computation can be performed by `Transform` (i.e., `Transform` has access to the variables used), we have the `Transform` routine perform the calculation and store the result in the partially decrypted ciphertext structure. `Decout` can later retrieve the result from this structure and use it in future calculations. We note that an optimization is to determine which variables `Decout` needs at any line of code to complete its subsequent calculations, and only store computed values from `Transform` if `Decout` actually requires them at some later point. This reduces both the partially de-

cryptured ciphertext size and processing time in `Decout`.

In the post-processing step, we finalize the output of `Transform` and the input of `Decout`. If there are any variables that `Decout` needs for its calculations that are not stored in the partially decrypted ciphertext structure, these variables are passed from `Transform` to `Decout` as separate parameters.

In certain cases, we may apply optimizations to improve the running time of `Decout` and decrease the partially decrypted ciphertext size. For example, certain for loops may not need to be written to `Decout`. This is the case if 1) the for loop is used to calculate a dot product; 2) the pairings in the for loop all share the same blinding factor; and 3) all pairings in the for loop have at least one blinded variable in them. If this is the case, we only need to take the result of the for loop calculation from `Transform` and unblind it with the single blinding factor in `Decout`. This is the equivalent of computing the blinded dot product in `Transform`, then simply performing one exponentiation in `Decout` to unblind the result. Similarly, if the for loop is used to calculate a dot product, and none of the pairings in the for loop has blinded variables, we do not need to write the for loop to `Decout`. In this case, we can simply save the result of the dot-product for loop to our cached data structures that are passed from `Transform` to `Decout` (i.e., the partially decrypted ciphertext).

If the criteria discussed above do not hold for a given for loop, we need to store the results of each run through the for loop to our data structures in `Transform` so they can be unblinded properly in `Decout`. In this case, we must pass the data structure that stores cached values from the for loops from `Transform` to `Decout` (which increases the size of the partially decrypted ciphertext); otherwise, we do not need to do so.

Code Generator.

The last step of the CloudSource toolchain is executable code generation. Up until this point, the internal components of our toolchain have been manipulating the input pairing-based encryption scheme in SDL. SDL is the intermediate representation (IR) upon which our tools are designed to operate. This enables a higher level of interoperability among disparate frontend input types, as is the case in many compiler designs. Rather than having to rewrite our tools for each frontend type of input we wish to support, we would only need to write a small translation engine that converts that type of input to SDL. Our tools would then be able to operate on the input scheme from there. In addition, by using an abstract, cryptography-centric high-level language such as SDL for our IR, our fundamental understanding of the cryptographic scheme is simpler and more accurate, rather than having to navigate through tedious and unnecessary details of a lower-level, non-cryptographic language. This cryptographic compiler approach often shows up in the literature [6, 7].

However, SDL itself is not executable. We clearly need the ability to execute the SDL that we generate (e.g., accuracy checks, benchmarking, proof-of-concept demonstrations, etc.) As a result, we extend the automated code-generation modules developed by Akinyele et al. [2] to translate SDL into Python and C++ executable code on the backend. We make use of these modules in CloudSource to produce Python and C++ code of the outsourced schemes we create, as well as the original non-outsourced schemes.

While we currently support Python and C++ only, we feel that our methods are sufficiently general to be able to support additional programming languages in the future without extreme burden or without having to redesign any part of our system.

3.1 Details of BFSolver

Given a secret key (SK), the goal of BFSolver is to determine which blinding factors to choose for each element in the key. There are a few approaches to blinding the secret key. One approach would be to choose a separate blinding factor for each element in the secret key. This approach is always secure but could result in inefficient decryption, especially when there are lists of group elements in the SK (e.g., in ABE). Alternatively, we could choose a separate blinding factor for each master secret key (MSK) value. While this approach is also secure, it can be hard to accomplish in practice and depends on how the decryption key is constructed. One major issue is that in some instances blinding a SK with only this approach might make it impossible to unblind in decryption. However, there are rules that govern when we can securely share blinding factors among SK elements. Furthermore, as shown in previous work [28], the more blinding factors that are shared among secret-key elements, the more efficient decryption becomes, thereby reducing ciphertext size.

Our approach is to first convert the aforementioned approaches to blinding into a series of general rules that can be systematically applied to a given secret key. At a high-level, they consist of the following:

1. all MSK values must be assigned unique blinding exponents.
2. all elements of SK must be blinded.
3. random values can share blinding factors with MSK values provided that they always share that value with the MSK value in SK.

The above rules form the bulk of what is involved when selecting blinding factors for a secret key. Once the blinding factor mapping satisfies these rules, a separate challenge is determining a lower bound for the mappings that does not violate the rules. Our central goal is to find such a lower bound for blinding factors to secret-key elements such that we preserve the security of the Transform key, TK. BFSolver utilizes the high-performance Z3 SMT Solver [21] as a core component to automate the selection of blinding factors for a given secret key and simultaneously obtain a lower bound.

3.2 Assigning Blinding Factors

At a high-level, BFSolver takes as input a configuration file that consists of the master secret key (MSK) exponents and random exponents selected in keygen. In addition, the file includes details of how each secret-key (SK) element is computed in SK. This is extracted by the term writing engine in KeygenSynth. BFSolver processes these inputs, instantiates the SMT solver and applies the above rules to the SK elements and expresses them as constraints over the variables in each SK element. We execute the solver with these inputs and check for a satisfiable solution.

While our implementation is tied to Z3, our architecture can support any compatible SMT Solver. In fact, our solution relies on a few basic features supported by many solvers

in practice. In particular, we require the ability to define algebraic datatypes, define possible values for those datatypes, specify constraints around variables of that type and allow the solver to search for a satisfiable solution under specified constraints. We also require the ability to isolate unsatisfiable constraints inside the solver via an unsatisfiable core, which is a common feature of many solvers. We will explain how we utilize these features in the sections to follow.

BFSolver is implemented in three phases. As we describe BFSolver, we will show how it applies to the Waters11 [48] CP-ABE keygen algorithm. In addition, we show how we automatically reproduce the solution described in previous work [28]. To recap, the Waters11 [48] SK is computed as follows:

$$K = g^\alpha \cdot g^{at}, L = g^t, \{K_x = F(x)^t\}_{x \in S}$$

where S is the user's attributes; α is the MSK; g , g^a are public parameters; F is a collision resistant hash function that maps $\{0, 1\}^* \rightarrow \mathbb{G}$; and t is a random exponent selected in keygen. We proceed to the first phase.

3.2.1 Phase 1: Setup Solver Input

The first phase is processing the secret key provided by KeygenSynth and setting up the search domain for the SMT Solver. This involves defining the algebraic datatype for blinding factors, declaring all exponents in SK element as blinding factor types, and declaring the *nil* value to indicate no blinding factor assigned. Next, we determine the upper bound on blinding factors for SK. Doing this appropriately for each SK is imperative for narrowing the solver's search space. By default, we compute the upper bound as the SK size. This indicates the least efficient solution of a unique blinding factor for each element in SK. In the Waters11 [48] example, we have three possible blinding factors in the worst case. All of this information is provided as input to the solver. The goal is to find the lower bound on blinding factors, as that leads to the most savings in terms of decryption time and ciphertext size.

3.2.2 Phase 2: Specify Constraints

The next phase is to first encode our general rules for blinding as constraints into the solver. The first rule specifies that each MSK variable must be assigned a unique blinding factor. We derive a constraint represented as a logical formula using $\wedge$ and $\vee$ connectors. For instance, given MSK variables a, b, c , we express this constraint as $a \neq b \wedge b \neq c \wedge a \neq nil \wedge b \neq nil \wedge c \neq nil$.¹ For the Waters11 scheme, this translates to just $a \neq nil$ because we have one MSK variable.

As mentioned before, the term rewriting engine in KeygenSynth outputs the relationships between exponents (or expressions) for each SK element. For example, for $g^a \cdot g^b$, the term rewriting engine outputs $a + b$ as the expression in the exponent. The second and third rules are applied by encoding the expressions associated with each SK element as constraints.

As described in section 3, we describe all expressions as either multiplication or addition². Our rules for these operations are fundamentally simple, but when combined can

¹In our implementation, we utilize the *Distinct* macro in Z3 to specify that each MSK variable should be given a unique blinding factor.

²Note that division and subtraction can always be rewritten in terms of multiplication and addition, respectively.

Process	Waters	BSW	LW	DFA-FE	DSE	HIBE	CKRS	SW	BGW
KeygenSynth	12.17s	12.78s	10.71s	36.76s	40.46s	12.12s	30.64s	13.32s	6.33s
TransformSynth	8.27s	8.69s	9.28s	10.43s	17.76s	7.99s	7.76s	10.77s	6.67s
Codegen-Python	2.89s	2.92s	0.65s	0.97s	0.91s	0.62s	0.83s	0.73s	0.48s
Codegen-C++	2.93s	3.07s	0.67s	1.03s	0.95s	0.64s	0.85s	0.74s	0.51s

Figure 3: Time in seconds required by the KeygenSynth, TransformSynth, Python Code Generator, and C++ Code Generator routines to process a variety of encryption schemes (averaged over 10 test runs). The running time for KeygenSynth includes the running time for BFSolver. In all cases, the standard deviation in the results were within $\pm 1\%$ of the average. We note that in each case, running times were directly proportional to the number of elements in the secret key. In addition, the Python Code Generator was slightly faster than the C++ Code Generator due to the additional type checking that is required for statically-typed languages such as C++.

produce complex constraints leading to many cases. The rules are as follows:

$$a + b \rightarrow a = b$$

An addition operator represents the constraint that whatever blinding factor the solver assigns to a must also be assigned to b .

$$a \cdot b \rightarrow (a \neq nil \wedge b = nil) \vee (a = nil \wedge b \neq nil)$$

A multiplication operator represents the constraint that the solver can assign either variable a or b a blinding factor (but not both). Finally, when a single exponent appears in a SK expression (e.g., $SK_0 \leftarrow a$), it indicates that it should be blinded due to the second rule (i.e., all SK elements must be blinded). The constraint for this type of expression is simply, $a \neq nil$.

For our Waters11 example, the term rewriter extracts three expressions: $K \leftarrow \alpha + t$, $L \leftarrow t$, $K_x \leftarrow t$. The corresponding constraints in the solver are $\alpha = t \wedge t \neq nil$ in addition to the previous constraint that $\alpha \neq nil$ for the MSK value. Upon constructing and specifying the constraints in the solver for SK, the next step is to execute the solver to check for a satisfiable solution.

3.2.3 Phase 3: Run Solver

The previous sections have shown how we describe the problem of choosing blinding factors to the solver as a series of constraints over a large space of possible variable mappings. Ultimately, the real challenge is finding the minimum number of blinding factors that meets our desired level of security. Once the constraints have been specified, the next phase is to check the solver for such a solution.

If a solution exists that satisfies all of the constraints, the solver returns a variable mapping of SK variables to blinding factors. In the Waters11 example, for instance, the solver determined that our constraints described earlier were satisfiable with respect to the blinding factor space of three. The variable mappings are as follows: $\alpha \leftarrow bf_0$, and $t \leftarrow bf_0$. This essentially means that α and t can share one single blinding factor.

If the solver does not find a satisfiable solution for the given constraints, we utilize the ability of Z3 to track and isolate the unsatisfiable constraints in the solver. In general, identifying unsatisfiable constraints are a fundamental feature of SMT solvers and we leverage this in BFSolver to determine the lower bound.

As described previously, constraints are linked to SK elements. When a solution cannot be obtained with the currently specified constraints, we determine which SK expression is associated with the unsatisfiable constraint using Z3's

Scheme	Num. BF's	Num. SK elements
WATERS11	1	$a + 2$
BGW05	1	n
DFA12	1	$3 \cdot t + 2 \cdot s + 2$
HIBE04	2	$\ell + 1$
BSW07	2	$2 \cdot a + 1$
DSE09	3	9
CKRS09	5	5
LW10	a	a
SW05	$a + 1$	$2 \cdot a$

Figure 4: The number of blinding factors and secret-key elements for each of the nine schemes we tested. a is the number of attributes in the secret key, n is the number of users in the system, t is the number of transitions, s is the number of accept states, and ℓ is the length of the hashed identities.

tracking mechanism. We remove the constraint associated with this expression in the solver and assign the corresponding SK element a unique blinding factor that is not shared thereafter. We then exclude the unsatisfiable constraint from the set of constraints and rerun the solver. If a solution is obtained, we return the mappings for the remaining SK elements in addition to the SK elements we have previously assigned unique blinding factors. Otherwise, we repeat the process of isolating the unsatisfiable constraint, assign the SK element another unique blinding factor, exclude it from the set of remaining constraints, and rerun the solver. This continues until we are without constraints, and at this point, we have reached the worst case solution of a unique blinding factor for each SK element. For instance, this was indeed the case for the CKRS09 scheme [14] (Blind and Anonymous IBE scheme). Figure 4 shows the number of secret-key elements and the number of blinding factors required for the nine schemes we tested.

4. GROUP SHARING OPTIMIZATIONS

Our techniques build on the exponent blinding approach employed in previous works. This approach has some limitations from the point of view of efficiency: in some cases it may be necessary to separately blind different components of the secret key, which increases the number of pairings required to compute the outsourced encryption scheme. It also leads to a similar increase in bandwidth.

In these circumstances it would be ideal if we could collapse several distinct blinding factors into a single blinding factor. Our observation is that in certain types of group

where the XDH or SXDH assumptions holds (*i.e.*, the DDH problem is hard in $\mathbb{G}_1$ or both $\mathbb{G}_1, \mathbb{G}_2$), it may be possible to reduce the number of blinding factors. We base this observation on the fact that for specific elements of the secret key (where the element appears only in one group, and has a random distribution), we can re-use the same blinding factor, based on the fact that an adversarial server should be unable to distinguish (g, h, g^a, h^a) from (g, h, g^a, g^b) for random $g, h \in \mathbb{G}_1, a, b \in \mathbb{Z}_q$. This optimization requires us to identify certain features of the key. We explore this optimization more fully and offer a proof of the statement in the full version of this work.

5. PERFORMANCE EVALUATION

In order to prove the soundness and efficiency of our outsourcing transformations, we conducted a series of benchmarks against all of the nine schemes that we cover in this paper. As previously discussed, CloudSource outputs executable code in both Python and C++ for the outsourced and non-outsourced versions of the scheme. In all cases, the automatically-generated Python and C++ code was implemented within Charm [1]. All of our tests were conducted on the C++ code due to its performance benefits over Python code of the same functionality.

System and Software Configuration. We conducted our benchmarks on two identical systems to ensure consistency. These systems were 2 x 2.66GHz 6-core Intel Xeon Macintosh Pro with 10GB of RAM and running Mac OS X Version 10.8.3. All of our tests were conducted in a single-threaded environment. We used Z3 v4.3 in our implementation of BFSolver. We utilized Charm v0.43 in C++ for our benchmarks against the MIRACL (v5.5.4) and RELIC (v0.3.4) libraries. We implement our schemes against Barreto-Naehrig (BN) curves due to their efficiency and measure our schemes on both Intel and limited ARM processors. In particular, we executed our schemes on a Motorola Droid 1 with a ARM Cortex A8 600 MHz processor, 256 SDRAM, 512MB flash memory, and runs Android v2.3.³ Our approach and results are discussed in the following sections.

5.1 Measurement Approach

For each test case, we perform three detailed experiments. First, we measure the cost of key generation and decryption in the original scheme compared to the outsourced key generation and decryption. Second, we compare the performance of a subset of the schemes on a limited mobile device to measure the effectiveness of outsourcing. Third, we compare the size of the full ciphertext produced by the original encryption to the size of the partially decrypted ciphertext returned by Transform. In both experiments, we observe that some schemes obtain a significant reduction in ciphertext size and decryption time while others benefit in one and not the other. We show the results of these experiments in Figures 5, 6 and 7.

Efficiency and Bandwidth of Schemes.⁴ For the ABE and Fuzzy-IBE schemes, we measure running time for all algorithms and ciphertext sizes at 100 attributes in the policy tree. Similarly, for the functional encryption (FE) schemes,

³We ran benchmarks on a rooted Droid device.

⁴Unless otherwise noted, our results were averaged over 100 iterations in all cases.

we construct a Deterministic Finite Automata (DFA) for accepting a string that contains hexadecimal characters. We also measure the running time and ciphertext size for a 1000-byte string, w , and a random message, $m \in \mathbb{G}_T$. We refer to the DFA functional encryption paper for more details of the scheme [49]. In all the IBE schemes, we measure running time and ciphertext size at 100-bytes for the public key string. Finally, for Broadcast Encryption (BE) [12], we measure the running times and ciphertext sizes at 100 users in a broadcast. In terms of bandwidth, in the cases where there were multiple binding factors needed to secure the secret key, there was a corresponding loss in the outsourced ciphertext size.

Efficiency of CloudSource. In Figure 3, we show the running time for the CloudSource toolchain. Our results indicate our ability to generate outsourced versions of cryptosystems in a reasonable amount of time. Furthermore, our utilization of the Z3 SMT solver enables us to avoid an inefficient approach for determining the number of blinding factors necessary to secure a transformation key.

Outsourcing for Mobile. Our results on the Droid 1 indicate that outsourcing is a useful tool for optimizing encryption schemes. The round trip time for Transform with a powerful server and Decout on a slow client still outperform decryption on a mobile device. These results are promising and show that having a tool like CloudSource that can automatically produce an outsource-ready cryptographic scheme is useful for everyday applications.

Scheme	Scheme Type	Model	Full CT Size (KB)	Out CT Size (KB)
Waters11 [48]	CP-ABE	RO	30.73	1.72
BSW [8]	CP-ABE	RO	30.80	1.72
LW [32]	MA-ABE	RO	96.05	62.48
DSE09 [47]	IBE	RO	1.49	3.42
HIBE04 [10]	HIBE	SM	1.14	1.72
CKRS09 [14]	Blind-IBE	SM	1.05	3.37
SW05 [40]	Fuzzy-IBE	RO	20.25	53.27
BGW05 [12]	BE	SM	0.75	1.70
DFA-FE12 [49]	FE	SM	20.61	8.53

Figure 6: A summary of our outsourcing results. We show the differences between full ciphertext and outsourced ciphertext sizes for all of our test cases.

6. RELATED WORK

Our research involves the intersection of secure and efficient outsourcing in pairing-based encryption schemes and automated cryptographic optimizations. We therefore survey both of these topics in this section.

With respect to the former, Green et al. [28] established a new technique for outsourcing decryption of attribute-based encryption (ABE). The authors showed how this technique could be manually developed for both CP-ABE [48] and KP-ABE [27]. In our work, we show how this manual approach can be automated in a generalized fashion.

There have been further research efforts in outsourcing ABE. Zhou et al. [51] presented a framework whereby resource-constrained mobile devices can securely outsource ABE encryption and decryption to cloud service providers (CSPs). Li et al. [33] designed a secure outsourced ABE system in which both key generation and decryption are outsourced to an untrusted provider.

BN256 on Intel

Schemes	Keygen	KeygenOut	Decrypt	Transform	Decout
<i>Attribute-Based Encryption (ABE)</i>					
BSW07 [8] Ciphertext-Policy ABE	0.25 s	0.48 s	3.30 s	3.06 s	0.01 s
Waters11 [48] Ciphertext-Policy ABE	0.09 s	0.16 s	3.36 s	2.86 s	0.01 s
LW10 [32] Decentralized ABE	0.14 s	0.21 s	4.30 s	3.85 s	0.67 s
<i>ID-Based Encryption</i>					
DSE09 [47] Dual-System	0.02 s	0.03 s	0.13 s	0.12 s	0.04 s
HIBE04 [10] Hierarchical-ID	0.02 s	0.03 s	0.09 s	0.08 s	0.01 s
CKRS09 [14] Blind and Anonymous-ID	0.02 s	0.03 s	0.07 s	0.06 s	0.02 s
SW05 [40] Fuzzy-ID	30.54ms	20.65 s	3.47 s	3.06 s	1.18 s
<i>Broadcast and Functional Encryption</i>					
BGW05 [12] Broadcast Encryption	0.16 s	0.31 s	0.03 s	0.05 s	0.007 s
DFA-FE12 [49] Regular Language Functional Encryption	2.58 s	5.10 s	21.45 s	20.67 s	2.95 s

Figure 5: Average running times over 100 iterations. We compare Keygen vs. KeygenOut, Decrypt vs. Transform and Decout for each of the nine pairing-based cryptosystems we cover in this paper. These benchmarks were executed against the MIRACL library on a server platform. In all test runs the standard deviation of the timing results were within $\pm 1\%$ of the average.

BN256 on Intel and ARM

Schemes	ARM Decrypt	Intel Transform	ARM Decout	Combined	Speedup Factor
<i>Attribute-Based Encryption (ABE)</i>					
BSW07 [8] Ciphertext-Policy ABE	73.6s	9.90s	0.20s	10.10s	7.3x
Waters11 [48] Ciphertext-Policy ABE	74.8s	5.93s	0.19s	6.12s	12.2x
LW10 [32] Decentralized ABE	85.8s	5.7s	10.5s	16.2s	5.3x
<i>ID-Based Encryption</i>					
HIBE04 [10] Hierarchical-ID	1.9s	0.10s	0.19s	0.29s	6.6x
CKRS09 [14] Blind and Anonymous-ID	1.6s	0.08s	0.45s	0.53s	3.0x
SW05 [40] Fuzzy-ID	7.1s	0.88s	1.90s	2.78s	2.6x
<i>Broadcast and Functional Encryption</i>					
BGW05 [12] Broadcast Encryption	0.94s	0.07s	0.11s	0.18s	5.2x

Figure 7: We compare the running times of Decrypt on a mobile phone (ARM processor) with that of Transform on our MacPro server (Intel processor) plus Decout on the mobile phone. Note that only the SW scheme was measured with 10 attributes over 10 iterations.

Outsourcing of encryption schemes has taken other forms in the literature. Hohenberger et al. [29] formally modeled the security of a resource-constrained client outsourcing computation to an untrusted helper with quantifications for efficiency and checkability.

Gennaro et al. [25] focused on verifiable computation using fully homomorphic encryption, in which a trusted but resource-constrained client can outsource the evaluation of a function with dynamic inputs to a set of workers. Chung et al. [19] extend the research by Gennaro et al. by reducing either the computational burden or the communication costs of the client in the offline processing phase of the protocol by Gennaro et al. Pirretti et al. [?] designed secure attribute-based systems using new constructions of ABE. Pirretti et al. also show how decryption in the scheme by Sahai and Waters [40] can be optimized to reduce the number of pairings, at the cost of increasing the number of exponentiations.

In this paper, we focus on pairing-based cryptosystems. Chevallier-Mames et al. [18] introduced a method whereby a resource-constrained client can securely outsource a pairing to a more computationally powerful entity.

There has been a large body of research on automating various cryptographic optimizations. MacKenzie et al. [35] designed a compiler that automatically generates protocols and source code for two-party function-sharing computation. Almeida et al. [3] created a zero-knowledge, certifying compiler that automatically translates an abstract proof goal written in the authors' Protocol Specification Language

(PSL) to a C implementation. Barbosa et al. [7] explored the use of CAO, a cryptographic domain-specific language and compiler, to improve the ability of cryptographic software engineers to write code for elliptic curve cryptography (ECC).

In addition, our research is close in nature to server-aided computation. Liu et al. [34] introduced Identity-Based Server-Aided Decryption (IBSAD). IBSAD is an identity-based encryption (IBE) scheme in which decryption is performed on a trusted client with the help of an external, untrusted server.

7. CONCLUSION

This paper investigated the problem of automating the process of designing outsourcing-ready cryptographic schemes. We believe that our results demonstrate that the problem is tractable and useful, and that these techniques could be useful amongst the growing collection of automated scheme design and analysis tools.

Our work leaves several open problems: first, our proposal uses a secure transformation that we have manually analyzed. However, to improve confidence in our results, we would like to output full reduction proofs that can be machine-verified using a proof checking tool, *e.g.*, CryptoVerif [9]. Secondly, we believe that there may be additional optimizations that we have not yet discovered, perhaps by re-writing schemes from one setting (symmetric pairings) to another.

Acknowledgments

Avi Rubin, J. Ayo Akinyele and Matthew Pagano are supported in part by NSF CNS-1010928 and HHS 90TR0003/01.

Matthew Green is supported in part by the Defense Advanced Research Projects Agency (DARPA) and the Air Force Research Laboratory (AFRL) under contract FA8750-11-2-0211, the Office of Naval Research contract N00014-11-1-0470, NSF grant CNS-1010928 and HHS 90TR0003/01.

Applying to all authors, the contents and views expressed are solely those of the authors and do not reflect the official policy, position or views of the Department of Defense, the HHS or the U.S. Government.

8. REFERENCES

- [1] AKINYELE, J. A., GARMAN, C., MIERS, I., PAGANO, M. W., RUSHANAN, M., GREEN, M., AND RUBIN, A. D. Charm: A framework for rapidly prototyping cryptosystems. *To appear, Journal of Cryptographic Engineering* (2013).
- [2] AKINYELE, J. A., GREEN, M., HOHENBERGER, S., AND PAGANO, M. W. Machine-generated algorithms, proofs and software for the batch verification of digital signature schemes. In *Proceedings of the 2012 ACM conference on Computer and communications security* (New York, NY, USA, 2012), CCS '12, ACM, pp. 474–487.
- [3] ALMEIDA, J. B., BANGERTER, E., BARBOSA, M., KRENN, S., SADEGHI, A.-R., AND SCHNEIDER, T. A certifying compiler for zero-knowledge proofs of knowledge based on Σ -protocols. In *Proceedings of the 15th European conference on Research in computer security* (Berlin, Heidelberg, 2010), ESORICS, Springer-Verlag, pp. 151–167.
- [4] ALMEIDA, J. B., BANGERTER, E., BARBOSA, M., KRENN, S., SADEGHI, A.-R., AND SCHNEIDER, T. A certifying compiler for zero-knowledge proofs of knowledge based on σ -protocols. In *Proceedings of the 15th European conference on Research in computer security* (Berlin, Heidelberg, 2010), ESORICS'10, Springer-Verlag, pp. 151–167.
- [5] ALMEIDA, J. B., BARBOSA, M., BANGERTER, E., BARTHE, G., KRENN, S., AND BÉGUELIN, S. Z. Full proof cryptography: verifiable compilation of efficient zero-knowledge protocols. In *CCS '12* (2012), pp. 488–500.
- [6] BANGERTER, E., KRENN, S., SEIFRIZ, M., AND ULTES-NITSCHKE, U. cplc - a cryptographic programming language and compiler. In *Information Security South Africa (ISSA), 2011* (aug. 2011), pp. 1–8.
- [7] BARBOSA, M., MOSS, A., AND PAGE, D. Compiler assisted elliptic curve cryptography. In *Proceedings of the 2007 OTM confederated international conference on On the move to meaningful internet systems: CoopIS, DOA, ODBASE, GADA, and IS - Volume Part II* (Berlin, Heidelberg, 2007), OTM'07, Springer-Verlag, pp. 1785–1802.
- [8] BETHENCOURT, J., SAHAI, A., AND WATERS, B. Ciphertext-policy Attribute-Based Encryption. In *Proceedings of the 2007 IEEE Symposium on Security and Privacy* (2007), IEEE Computer Society, pp. 321–334.
- [9] BLANCHET, B. CryptoVerif: A computationally sound mechanized prover for cryptographic protocols. In *Dagstuhl seminar "Formal Protocol Verification Applied"* (Oct. 2007).
- [10] BONEH, D., AND BOYEN, X. Short signatures without random oracles. In *EUROCRYPT '04* (2004), C. Cachin and J. Camenisch, Eds., vol. 3027 of LNCS, Springer, pp. 382–400.
- [11] BONEH, D., AND FRANKLIN, M. K. Identity-based encryption from the Weil pairing. In *CRYPTO* (2001), pp. 213–229.
- [12] BONEH, D., GENTRY, C., AND WATERS, B. Collusion resistant broadcast encryption with short ciphertexts and private keys. Springer-Verlag, pp. 258–275.
- [13] CAMENISCH, J., HOHENBERGER, S., AND PEDERSEN, M. Ø. Batch verification of short signatures. In *EUROCRYPT '07* (2007), vol. 4515 of LNCS, Springer, pp. 246–263. Full version at <http://eprint.iacr.org/2007/172>.
- [14] CAMENISCH, J., KOHLWEISS, M., RIAL, A., AND SHEEDY, C. Blind and anonymous identity-based encryption and authorised private searches on public key encrypted data. In *PKC* (Berlin, Heidelberg, 2009), Irvine, Springer-Verlag, pp. 196–214.
- [15] CAO, T., LIN, D., AND XUE, R. Security analysis of some batch verifying signatures from pairings. *International Journal of Network Security* 3, 2 (2006), 138–143.
- [16] CAO, T., AND MAO, X. Collusion attack on a server-aided unbalanced rsa key generation protocol. In *Communication Technology, 2006. ICCCT '06. International Conference on* (nov. 2006), pp. 1–3.
- [17] CHATTERJEE, S., AND SARKAR, P. Trading time for space: Towards an efficient IBE scheme with short(er) public parameters in the standard model. In *ICISC* (2005), vol. 3935 of LNCS, pp. 424–440.
- [18] CHEVALLIER-MAMES, B., CORON, J.-S., MCCULLAGH, N., NACCACHE, D., AND SCOTT, M. Secure delegation of elliptic-curve pairing. In *Smart Card Research and Advanced Application*, D. Gollmann, J.-L. Lanet, and J. Iguchi-Cartigny, Eds., vol. 6035 of *Lecture Notes in Computer Science*. Springer Berlin Heidelberg, 2010, pp. 24–35.
- [19] CHUNG, K.-M., KALAI, Y., AND VADHAN, S. Improved delegation of computation using fully homomorphic encryption. In *Proceedings of the 30th annual conference on Advances in cryptology* (Berlin, Heidelberg, 2010), CRYPTO'10, Springer-Verlag, pp. 483–501.
- [20] CRAMER, R., AND SHOUP, V. Design and analysis of practical public-key encryption schemes secure against adaptive chosen ciphertext attack. *SIAM J. Comput.* 33, 1 (Jan. 2004), 167–226.
- [21] DE MOURA, L., AND BJØRNER, N. Z3: an efficient smt solver. In *Proceedings of the Theory and practice of software, 14th international conference on Tools and algorithms for the construction and analysis of systems* (Berlin, Heidelberg, 2008), TACAS'08/ETAPS'08, Springer-Verlag, pp. 337–340.
- [22] DE MOURA, L., AND BJØRNER, N. Satisfiability modulo theories: introduction and applications. *Commun. ACM* 54, 9 (Sept. 2011), 69–77.
- [23] DODIS, Y., AND FAZIO, N. Public key broadcast encryption for stateless receivers. In *Digital Rights Management* (2003), J. Feigenbaum, Ed., vol. 2696 of *Lecture Notes in Computer Science*, Springer Berlin Heidelberg, pp. 61–80.
- [24] FERRARA, A. L., GREEN, M., HOHENBERGER, S., AND PEDERSEN, M. Ø. Practical short signature batch verification. In *CT-RSA* (2009), vol. 5473 of LNCS, pp. 309–324.
- [25] GENNARO, R., GENTRY, C., AND PARNO, B. Non-interactive verifiable computing: outsourcing computation to untrusted workers. In *Proceedings of the 30th annual conference on Advances in cryptology* (Berlin, Heidelberg, 2010), CRYPTO'10, Springer-Verlag, pp. 465–482.
- [26] GENTRY, C. Fully homomorphic encryption using ideal lattices. In *Proceedings of the 41st annual ACM symposium on Theory of computing* (New York, NY,

- USA, 2009), STOC, ACM, pp. 169–178.
- [27] GOYAL, V., PANDEY, O., SAHAI, A., AND WATERS, B. Attribute-based encryption for fine-grained access control of encrypted data. In *Proceedings of the 13th ACM conference on Computer and communications security* (New York, NY, USA, 2006), CCS '06, ACM, pp. 89–98.
 - [28] GREEN, M., HOHENBERGER, S., AND WATERS, B. Outsourcing the decryption of abe ciphertexts. In *Proceedings of the 20th USENIX conference on Security* (Berkeley, CA, USA, 2011), SEC'11, USENIX Association, pp. 34–34.
 - [29] HOHENBERGER, S., AND LYSYANSKAYA, A. How to securely outsource cryptographic computations. In *Proceedings of the Second international conference on Theory of Cryptography* (Berlin, Heidelberg, 2005), TCC'05, Springer-Verlag, pp. 264–282.
 - [30] IOVINO, V., AND PERSIANO, G. Hidden-vector encryption with groups of prime order. In *Proceedings of the 2nd international conference on Pairing-Based Cryptography* (Berlin, Heidelberg, 2008), Pairing '08, Springer-Verlag, pp. 75–88.
 - [31] LEWKO, A., SAHAI, A., AND WATERS, B. Revocation systems with very small private keys. In *Proceedings of the IEEE Symposium on Security and Privacy* (Washington, DC, USA, 2010), SP, IEEE Computer Society, pp. 273–285.
 - [32] LEWKO, A., AND WATERS, B. Decentralizing attribute-based encryption. In *EUROCRYPT* (2011), K. G. Patterson, Ed., vol. 6632 of LNCS, Springer, pp. 568–588. <http://eprint.iacr.org/>.
 - [33] LI, J., LI, J., CHEN, X., JIA, C., AND WONG, D. S. Secure outsourced attribute-based encryption. Cryptology ePrint Archive, Report 2012/635, 2012. <http://eprint.iacr.org/>.
 - [34] LIU, J. K., CHU, C. K., AND ZHOU, J. Identity-based server-aided decryption. In *Proceedings of the 16th Australasian conference on Information security and privacy* (Berlin, Heidelberg, 2011), ACISP'11, Springer-Verlag, pp. 337–352.
 - [35] MACKENZIE, P., OPREA, A., AND REITER, M. K. Automatic generation of two-party computations. In *Proceedings of the 10th ACM conference on Computer and communications security* (New York, NY, USA, 2003), CCS '03, ACM, pp. 210–219.
 - [36] MATSUMOTO, T., KATO, K., AND IMAI, H. Speeding up secret computations with insecure auxiliary devices. In *Proceedings on Advances in cryptography* (New York, NY, USA, 1990), CRYPTO '88, Springer-Verlag New York, Inc., pp. 497–506.
 - [37] MEIKLEJOHN, S., ERWAY, C. C., KÜPÇÜ, A., HINKLE, T., AND LYSYANSKAYA, A. ZKPD: a language-based system for efficient zero-knowledge proofs and electronic cash. In *Proceedings of the 19th USENIX conference on Security* (Berkeley, CA, USA, 2010), USENIX Security, USENIX Association, pp. 13–13.
 - [38] NACCACHE, D. Secure and practical identity-based encryption. Cryptology ePrint Archive, Report 2005/369, 2005. <http://eprint.iacr.org/>.
 - [39] RAMANNA, S. C., CHATTERJEE, S., AND SARKAR, P. Variants of waters' dual system primitives using asymmetric pairings - (extended abstract). In *PKC '12* (2012), pp. 298–315.
 - [40] SAHAI, A., AND WATERS, B. Fuzzy identity-based encryption. In *EUROCRYPT* (2005), pp. 457–473.
 - [41] SCHNORR, C. Efficient signature generation by smart cards. *Journal of Cryptology* 4 (1991), 161–174.
 - [42] SHAMIR, A. Identity-based cryptosystems and signature schemes. In *CRYPTO* (1984), vol. 196 of LNCS, pp. 47–53.
 - [43] STANEK, M. Attacking LCCC batch verification of RSA signatures, 2006. Cryptology ePrint Archive: Report 2006/111.
 - [44] WATERS, B. Efficient identity-based encryption without random oracles. In *EUROCRYPT '05* (2005), vol. 3494 of LNCS, Springer, pp. 320–329.
 - [45] WATERS, B. Ciphertext-policy attribute-based encryption: An expressive, efficient, and provably secure realization. Cryptology ePrint Archive, Report 2008/290, 2008. <http://eprint.iacr.org/>.
 - [46] WATERS, B. Dual System Encryption: Realizing Fully Secure IBE and HIBE under Simple Assumptions. In *CRYPTO* (2009), pp. 619–636.
 - [47] WATERS, B. Dual system encryption: Realizing fully secure ibe and hibe under simple assumptions. In *Proceedings of the 29th Annual International Cryptology Conference on Advances in Cryptology* (Berlin, Heidelberg, 2009), CRYPTO '09, Springer-Verlag, pp. 619–636.
 - [48] WATERS, B. Ciphertext-policy attribute-based encryption: an expressive, efficient, and provably secure realization. In *Proceedings of the 14th international conference on Practice and theory in public key cryptography conference on Public key cryptography* (Berlin, Heidelberg, 2011), PKC'11, Springer-Verlag, pp. 53–70.
 - [49] WATERS, B. Functional encryption for regular languages. In *Advances in Cryptology — CRYPTO 2012* (2012), R. Safavi-Naini and R. Canetti, Eds., vol. 7417 of *Lecture Notes in Computer Science*, Springer Berlin Heidelberg, pp. 218–235.
 - [50] ZAVERUCHA, G. M., AND STINSON, D. R. Group testing and batch verification. In *Proceedings of the 4th international conference on Information theoretic security* (Berlin, Heidelberg, 2010), ICITS'09, Springer-Verlag, pp. 140–157.
 - [51] ZHOU, Z., AND HUANG, D. Efficient and secure data storage operations for mobile cloud computing. In *Network and Service Management (CNSM), 2012 8th International Conference on* (oct. 2012), pp. 37–45.

Computing on Authenticated Data

Jae Hyun Ahn Johns Hopkins University arjuna@cs.jhu.edu	Dan Boneh [*] Stanford University dabo@cs.stanford.edu	Jan Camenisch [†] IBM Research – Zurich jca@zurich.ibm.com
Susan Hohenberger [‡] Johns Hopkins University susan@cs.jhu.edu	abhi shelat [§] University of Virginia abhi@cs.virginia.edu	Brent Waters [¶] University of Texas at Austin bwaters@cs.utexas.edu

July 3, 2012

Abstract

In tandem with recent progress on computing on encrypted data via fully homomorphic encryption, we present a framework for computing on *authenticated* data via the notion of slightly homomorphic signatures, or P -homomorphic signatures. With such signatures, it is possible for a third party to *derive* a signature on the object m' from a signature of m as long as $P(m, m') = 1$ for some predicate P which captures the “authenticatable relationship” between m' and m . Moreover, a derived signature on m' reveals *no extra information* about the parent m .

Our definition is carefully formulated to provide one unified framework for a variety of distinct concepts in this area, including arithmetic, homomorphic, quotable, redactable, transitive signatures and more. It includes being unable to distinguish a derived signature from a fresh one *even when given the original signature*. The inability to link derived signatures to their original sources prevents some practical privacy and linking attacks, which is a challenge not satisfied by most prior works.

Under this strong definition, we then provide generic constructions for all univariate and closed predicates, and specific efficient constructions for a broad class of natural predicates such as quoting, subsets, weighted sums, averages, and Fourier transforms. To our knowledge, these are the first efficient constructions for these predicates (excluding subsets) that provably satisfy this strong security notion.

^{*}Supported by NSF, DARPA, and AFOSR. Applying to all authors, the views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the Defense Advanced Research Projects Agency or the US government.

[†]This work has been funded by the European Community’s Seventh Framework Programme (FP7/2007-2013) under grant agreement no. 216483 (PrimeLife).

[‡]Supported by the Defense Advanced Research Projects Agency (DARPA) and the Air Force Research Laboratory (AFRL) under contract FA8750-11-2-0211, the Office of Naval Research under contract N00014-11-1-0470, a Microsoft Faculty Fellowship and a Google Faculty Research Award.

[§]Supported by NSF CNS-0845811 and TC-1018543, Defense Advanced Research Projects Agency (DARPA) and the Air Force Research Laboratory (AFRL) under contract FA8750-11-2-0211, and a Microsoft New Faculty Fellowship.

[¶]Supported by NSF CNS-0915361 and CNS-0952692, AFOSR Grant No: FA9550-08-1-0352, DARPA PROCEED, DARPA N11AP20006, Google Faculty Research award, the Alfred P. Sloan Fellowship, Microsoft Faculty Fellowship, and Packard Foundation Fellowship.

1 Introduction

In tandem with recent progress on computing *any function* on encrypted data, e.g., [29, 55, 52], this work explores computing on unencrypted signed data. In the past few years, several independent lines of research touched on this area:

- Quoting/redacting: [54, 35, 1, 42, 32, 19, 18, 20] Given Alice’s signature on some message m anyone should be able to derive Alice’s signature on a subset of m . Quoting typically applies to signed text messages where one wants to derive Alice’s signature on a substring of m . Quoting can also apply to signed images where one wants to derive a signature on a subregion of the image (say, a face or an object) and to data structures where one wants to derive a signature of a subset of the data structure such as a sub-tree of a tree.
- Arithmetic: [36, 61, 24, 15, 28, 14, 13, 59] Given Alice’s signature on vectors $\mathbf{v}_1, \dots, \mathbf{v}_k \in \mathbb{F}_p^n$ anyone should be able to derive Alice’s signature on a vector $\mathbf{v}$ in the linear span of $\mathbf{v}_1, \dots, \mathbf{v}_k$. Arithmetic on signed data is motivated by applications to secure network coding [27]. We show that these schemes can be used to compute authenticated linear operations such as computing an authenticated weighted sum of signed data and an authenticated Fourier transform. As a practical consequence of this, we show that an untrusted database storing signed data (e.g., employee salaries) can publish an authenticated average of the data without leaking any other information about the stored data. Recent constructions go beyond linear operations and support low degree polynomial computations [13].
- Transitivity: [47, 41, 6, 33, 7, 50, 60, 46] Given Alice’s signature on edges in a graph G anyone should be able to derive Alice’s signature on a pair of vertices (u, v) if and only if there is a path in G from u to v . The derived signature on the pair (u, v) must be indistinguishable from a fresh signature on (u, v) had Alice generated one herself [41]. This requirement ensures that the derived signature on (u, v) reveals no information about the path from u to v used to derive the signature.

In this paper, we put forth a general framework for computing on authenticated data that encompasses these lines of research and much more. While prior definitions mostly contained artifacts specific to the type of malleability they supported and, thus, were hard to compare to one another, we generalize and strengthen these disparate notions into a single definition. This definition can be instantiated with any predicate, and we allow repeated computation on the signatures (e.g., it is possible to quote from a quoted signature.) During our study, we realized that the “privacy” notions offered by many existing definitions are, in our view, insufficient for some practical applications. We therefore require a stronger (and seemingly a significantly more challenging to achieve) property called *context hiding*. Under this definition, we provide two generic solutions for computing signatures on any univariate, closed predicate; however, these generic constructions are not efficient. We also present efficient constructions for three problems: quoting substrings in Section 4, a subset predicate in Section 5, and a weighted average over data in Section 6 (which captures weighted sums and Fourier transforms). Our quoting substring construction is novel and significantly more efficient than the generic solutions. For the problems of subsets and weighted averages, we show somewhat surprising connections to respective existing solutions in attribute-based encryption and network coding signatures.

1.1 Overview

A general framework. Let $\mathcal{M}$ be some message space and let $2^{\mathcal{M}}$ be its powerset. Consider a predicate $P : 2^{\mathcal{M}} \times \mathcal{M} \rightarrow \{0, 1\}$ mapping a set of messages and a message to a bit. Loosely speaking we say that a signature scheme supports computations with respect to P if the following holds:

Let $M \subset \mathcal{M}$ be a set of messages and let m' be a *derived* message, namely m' satisfies $P(M, m') = 1$. Then there exists an efficient procedure that can derive Alice’s signature on m' from Alice’s independent signatures on all of the messages in M .

For the quoting application, the predicate P is defined as $P(M, m') = 1$ iff m' is a quote from the set of messages M . Here we focus on quoting from a single message m so that P is false whenever M contains more than one component¹, and thus use the notation $P(m, m')$ as shorthand for $P(\{m\}, m')$. The predicate P for arithmetic computations is defined in Appendix A and essentially says that $P((\mathbf{v}_1, \dots, \mathbf{v}_k), \mathbf{v})$ is true whenever $\mathbf{v}$ is in the span of $\mathbf{v}_1, \dots, \mathbf{v}_k$.

We emphasize that signature derivation can be iterative. For example, given a message-signature pair (m, σ) from Alice, Bob can publish a derived message-signature pair (m', σ') for an m' where $P(m, m')$ holds. Charlie, using (m', σ') , may further derive a signature σ'' on m'' . In the quoting application, Charlie is quoting from a quote which is perfectly fine.

Security. We give a clean security definition that captures two properties: unforgeability and context hiding. We briefly discuss each in turn and give precise definitions in the next section.

- Unforgeability captures the idea that an attacker may be given various derived signatures (perhaps iteratively derived) on messages of his choice. The attacker should be unable to produce a signature on a message that is not derivable from the set of signed messages at his possession. E.g., suppose Alice generates (m, σ) and gives it to Bob who then publishes a derived signature (m', σ') . Then an attacker given (m', σ') should be unable to produce a signature on m or on any other message m'' such that $P(m', m'') = 0$.
- Context hiding captures an important privacy property: a signature should reveal nothing more than the message being signed. In particular, if a signature on m' was derived from a signature on m , an attacker should not learn anything about m other than what can be inferred from m' . This should be true even if the original signature on m is revealed. For example, a signed quote should not reveal anything about the message from which it was quoted, including its length, the position of the quote, whether its parent document is the same as another quote, whether it was derived from a given signed message or generated freshly, etc.

Defining context hiding is an interesting and subtle task. In the next section, we give a definition that captures a very strong privacy requirement. We discuss earlier attempts at defining privacy following our definition in Section 2.3; while many prior works use a similar sounding *intuition* as we give above, most contain a fundamental difference to ours in their *formalization*.

We note that notions such as group or ring signatures [25, 5, 21, 11, 49] have considered the problem of hiding the identity of a signer among a set of users. Context hiding ensures privacy for the data rather than the signer. Our goal is to hide the legacy of how a signature was created.

¹We leave it for future work to construct systems for securely quoting from two messages (or possibly more) as defined next.

Efficiency. We require that the size of a signature, whether fresh or derived, depend only on the size of the object being signed. This rules out solutions where the signature grows with each derivation.

Generic Approaches. We begin with two generic constructions that can be inefficient. They apply to *closed, univariate* predicates, namely predicates $P(M, m')$ where M contains a single message (P is false when $|M| > 1$) and where if $P(a, b) = P(b, c) = 1$ then $P(a, c) = 1$. The first construction uses any standard signature scheme S where the signing algorithm is deterministic. (One can enforce determinism using PRFs [30].) To sign a message $m \in \mathcal{M}$, one uses S to sign each message m' such that $P(m, m') = 1$. The signature consists of all these signature components. To verify a signature for m , one checks the signature component corresponding to the message m . To derive a signature m' from m , one copies the signature components for all m'' such that $P(m', m'') = 1$. Soundness of the construction follows from the security of the underlying standard scheme S and context hiding from the fact that signing in S is deterministic.

Unfortunately, these signatures may become large consisting up to $|\mathcal{M}|$ signature components — effecting both the signing time and signature size. Our second generic construction alleviates the space burden by using an RSA accumulator. The construction works in a similar brute force fashion where a signature on m is an accumulator value on all m' such that $P(m, m') = 1$. While this produces short signatures, the time component of both verification and derivation are even worse than the first generic approach. Thus, these generic approaches are too expensive for most interesting predicates. We detail these generic approaches and proofs in Section 3, where we also discuss a generic construction using NIZK.

Our Quoting Construction. We turn to more efficient constructions. First, we set out to construct a signature for quoting *substrings*², which although conceptually simple is non-trivial to realize securely. As an efficiency baseline, we note that the brute force generic construction of the quoting predicate would result in n^2 components for a signature on n characters. So any interesting construction must perform more efficiently than this. We prove our construction selectively secure.³ In addition, we give some potential future directions for achieving adaptive security and removing the use of random oracles.

Our construction uses bilinear groups to link different signature components together securely, but in such a way that the context can be hidden by a re-randomizing step in the derivation algorithm. A signature in our system on a message of length n consists of $n \lg n$ group elements; intuitively organized as $\lg n$ group elements assigned to each character. To derive a new signature on a substring of ℓ characters, one roughly removes the group elements not associated with the new substring and then re-randomizes the remaining part of the signature. This results in a new signature of $\ell \lg \ell$ group elements. The technical challenge consists in simultaneously allowing re-randomization and preserving the “linking” between successive characters. In addition, there is a second option in our derive algorithm that allows for the derivation of a short signature of $\lg \ell$ group elements; however the derive procedure cannot be applied again to this short signature. *Thus, we support quoting from quotes, and also provide a compression option which produces a very short quote, but the price for this is that it cannot be quoted from further.*

²A substring of $x_1 \dots x_n$ is some $x_i \dots x_j$ where $i, j \in [1, n]$ and $i \leq j$. We emphasize that we are not considering *subsequences*. Thus, it is *not* possible, in this setting, to extract a signature on “I like fish” from one on “I do not like fish”.

³Following an analog of [22], selective security for signatures requires the attacker to give the forgery message before seeing the verification key.

Computing Signatures on Subsets and Weighted Averages. Our final two contributions are schemes for deriving signatures on subsets and weighted averages on signatures. Rather than create entirely new systems, we show connections to existing Attribute-Based Encryption schemes and Network Coding Signatures. Briefly, our subset construction extends the concept of Naor [12] who observed that every IBE scheme can be transformed into a standard signature scheme by applying the IBE KeyGen algorithm as a signing algorithm. Here we show an analog for known Ciphertext-Policy (CP) ABE schemes. The KeyGen algorithm which generates a key for a set S of attributes can be used as a signing algorithm for the set S . For known CP-ABE systems [8, 37, 58] it is straightforward to derive a key for a subset S' of S and to re-randomize the signature/key. To verify a signature on S we can apply Naor's signature-from-IBE idea and encrypt a random message X to a policy that is an AND of all the attributes in S and see if the signature can be used as an ABE key to decrypt to X . Signatures for subsets have been previously considered in [33, §6.4], but without context hiding requirements. We provide further details in Section 5. Our construction for weighted sums is presented in Section 6, where we discuss how this applies to Fourier transforms.

2 Definitions

Definition 2.1 (Derived messages) Let $\mathcal{M}$ be a message space and let $P : 2^{\mathcal{M}} \times \mathcal{M} \rightarrow \{0, 1\}$ be a predicate from sets over $\mathcal{M}$ and a message in $\mathcal{M}$ to a bit. We say that a message m' is **derivable** from the set $M \subseteq \mathcal{M}$ if $P(M, m') = 1$. We denote by $P^*(M)$ the set of messages derivable from M by repeated derivation. That is, let $P^0(M)$ be the set of messages derivable from M and for $i > 0$ let $P^i(M)$ be the set of messages derivable from $P^{i-1}(M)$. Then $P^*(M) := \cup_{i=0}^{\infty} P^i(M)$.

We define the closure of P , denoted P^* , as the predicate defined by $P^*(M, m) = 1$ iff $m \in P^*(M)$.

A P -homomorphic signature scheme Π for message space $\mathcal{M}$ and predicate P is a triple of PPT algorithms:

KeyGen(1^λ): the key generation algorithm outputs a key pair (pk, sk) . We treat the secret key sk as a signature on the empty tuple $\varepsilon \in \mathcal{M}^*$. We also assume that pk is embedded in sk .

SignDerive($pk, (\{\sigma_m\}_{m \in M}, M), m', w$): the algorithm takes as input the public key, a set of messages $M \subseteq \mathcal{M}$ and corresponding signatures $\{\sigma_m\}_{m \in M}$, a derived message $m' \in \mathcal{M}$, and possibly some auxiliary information w . It produces a new signature σ' or a special symbol $\perp$ to represent failure. For complicated predicates P , the auxiliary information w serves as a witness that $P(M, m') = 1$. To simplify the notation we often drop w as an explicit argument.

As shorthand we write $\mathbf{Sign}(sk, m) := \mathbf{SignDerive}(pk, (sk, \varepsilon), m, \cdot)$ to denote that any message can be derived when the original signature is the signing key. For a set of messages $M = \{m_1, \dots, m_k\} \subset \mathcal{M}^*$ it is convenient to let $\mathbf{Sign}(sk, M)$ denote independently signing each of the k messages, namely:

$$\mathbf{Sign}(sk, M) := (\mathbf{Sign}(sk, m_1), \dots, \mathbf{Sign}(sk, m_k)) .$$

Verify(pk, m, σ): given a public key, message, and purported signature σ , the algorithm returns 1 if the signature is valid and 0 otherwise.

We assume that testing $m \in \mathcal{M}$ can be done efficiently, and that **Verify** returns 0 if $m \notin \mathcal{M}$.

Correctness. We require that for all key pairs (sk, pk) generated by $\mathbf{KeyGen}(1^n)$ and for all $M \in \mathcal{M}^*$ and $m' \in \mathcal{M}$ we have:

- if $P(M, m') = 1$ then $\mathbf{SignDerive}(pk, (\mathbf{Sign}(sk, M), M), m') \neq \perp$, and
- for all signature tuples $\{\sigma_m\}_{m \in M}$ such that $\sigma' \leftarrow \mathbf{SignDerive}(pk, (\{\sigma_m\}_{m \in M}, M), m') \neq \perp$, we have $\mathbf{Verify}(pk, m', \sigma') = 1$.

In particular, correctness implies that a signature generated by $\mathbf{SignDerive}$ can be used as an input to $\mathbf{SignDerive}$ so that signatures can be further derived from derived signatures, if allowed by P .

Derivation efficiency. In many cases it is desirable that the size of a derived signature depend only on the size of the derived message. This rules out signatures that expand as one iteratively calls $\mathbf{SignDerive}$. All the constructions in this paper are derivation efficient in this sense.

Definition 2.2 (Derivation-Efficient) *A signature scheme is derivation-efficient if there exists a polynomial p such that for all $(pk, sk) \leftarrow \mathbf{KeyGen}(1^\lambda)$, set $M \subseteq \mathcal{M}^*$, signatures $\{\sigma_m\}_{m \in M} \leftarrow \mathbf{Sign}(sk, M)$ and derived messages m' where $P(M, m') = 1$, we have*

$$|\mathbf{SignDerive}(pk, \{\sigma_m\}_{m \in M}, M, m')| = p(\lambda, |m'|).$$

2.1 Security: Unforgeability

To define unforgeability, we extend the basic notion of existential unforgeability with respect to adaptive chosen-message attacks [31]. The definition captures the idea that if the attacker is given a set of signed messages (either primary or derived) then the only messages he can sign are derivations of the signed messages he was given. This is defined using a game between a challenger and an adversary $\mathcal{A}$ with respect to scheme Π over message space $\mathcal{M}$.

— Game $\mathbf{Unforg}(\Pi, \mathcal{A}, \lambda, P)$:

Setup: The challenger runs $\mathbf{KeyGen}(1^\lambda)$ to obtain (pk, sk) and sends pk to $\mathcal{A}$. The challenger maintains two sets T and Q that are initially empty.

Queries: Proceeding adaptively, the adversary issues the following queries to the challenger:

- $\mathbf{Sign}(m \in \mathcal{M})$: the challenger generates a unique handle h , runs $\mathbf{Sign}(sk, m) \rightarrow \sigma$ and places (h, m, σ) into a table T . It returns the handle h to the adversary.
- $\mathbf{SignDerive}(\vec{h} = (h_1, \dots, h_k), m')$: the oracle retrieves the tuples (h_i, σ_i, m_i) in T for $i = 1, \dots, k$, returning $\perp$ if any of them do not exist. Let $M := (m_1, \dots, m_k)$ and $\{\sigma_m\}_{m \in M} := \{\sigma_1, \dots, \sigma_k\}$. If $P(M, m')$ holds, then the oracle generates a new unique handle h' , runs $\mathbf{SignDerive}(pk, (\{\sigma_m\}_{m \in M}, M), m') \rightarrow \sigma'$ and places (h', m', σ') into T , and returns h' to the adversary.
- $\mathbf{Reveal}(h)$: Returns the signature σ corresponding to handle h , and adds (σ', m') to the set Q .

Output: Eventually, the adversary outputs a pair (σ', m') . The output of the game is 1 (i.e., the adversary wins the game) if:

- $\mathbf{Verify}(pk, m', \sigma') = 1$ and,

- let $M \subseteq \mathcal{M}$ be the set of messages in Q then $P^*(M, m') = 0$ where P^* is the closure of P from Definition 2.1.

Else, the output of the game is 0. Define $\mathbf{Forg}_{\mathcal{A}}$ as the probability that $\Pr[\mathbf{Unforg}(\Pi, \mathcal{A}, \lambda, P) = 1]$.

Interestingly, for some predicates it may be difficult to test if the adversary won the game. For all the predicates we consider in this paper, this will be quite easy.

Definition 2.3 (Unforgeability) *A P -homomorphic signature scheme Π is **unforgeable** with respect to adaptive chosen-message attacks if for all PPT adversaries $\mathcal{A}$, the function $\mathbf{Forg}_{\mathcal{A}}$ is negligible in λ .*

*A P -homomorphic signature scheme Π is **selective unforgeable** with respect to adaptive chosen-message attacks if for all PPT adversaries $\mathcal{A}$ who begin the above game by announcing the message m' on which they will forge, $\mathbf{Forg}_{\mathcal{A}}$ is negligible in λ .*

Properties of the definition. By taking P to be the equality oracle, namely $P(x, y) = 1$ iff $x = y$, we obtain the standard unforgeability requirement for signatures.

Notice that *Sign* and *SignDerive* queries return handles, but do not return the actual signatures. A system proven secure under this definition adequately rules out the following attack: suppose (m, σ) is a message signature pair and (m', σ') is a message-signature pair derived from it, namely $\sigma' = \mathbf{SignDerive}(pk, \sigma, m, m')$. For example, suppose m' is a quote from m . Then given (m', σ') it should be difficult to produce a signature on m and indeed our definition treats a signature on m as a valid forgery.

The unforgeability game imposes some constraints on P : (1) P must be reflexive, i.e. $P(m, m) = 1$ for all $m \in \mathcal{M}$, (2) P must be monotone, i.e. $P(M, m') \Rightarrow P(M', m')$ where $M \subseteq M'$. It is easy to see that predicates that do not satisfy these requirements cannot be realized under Definition 2.3.

2.2 Security: Context Hiding (a.k.a., Privacy)

Let M be some set and let m' be a derived message from M (i.e., $P(M, m') = 1$). Context hiding captures the idea that a signature on m' derived from signatures on M should reveal no information about M beyond what is revealed by m' . For example, in the case of quoting, a signature on a quote from m should reveal nothing more about m : not the length of m , not the position of the quote in m , etc. The same should hold even if the attacker is given signatures on multiple quotes from m .

We put forth the following powerful *statistical* definition of context hiding and discuss its implications following the definition. We were most easily able to leverage a statistical definition for our proofs, although we also give an alternative *computational* definition in Appendix A.

Definition 2.4 (Strong Context Hiding) *Let $M \subseteq \mathcal{M}^*$ and $m' \in \mathcal{M}$ be messages such that $P(M, m') = 1$. Let $(pk, sk) \leftarrow \mathbf{KeyGen}(1^\lambda)$ be a key pair. A signature scheme $(\mathbf{KeyGen}, \mathbf{SignDerive}, \mathbf{Verify})$ is strongly context hiding (for predicate P) if for all such triples $((pk, sk), M, m')$, the following two distributions are statistically close:*

$$\begin{aligned} & \left\{ (sk, \{\sigma_m\}_{m \in M} \leftarrow \mathbf{Sign}(sk, M), \mathbf{Sign}(sk, m')) \right\}_{sk, M, m'} \\ & \left\{ (sk, \{\sigma_m\}_{m \in M} \leftarrow \mathbf{Sign}(sk, M), \mathbf{SignDerive}(pk, (\{\sigma_m\}_{m \in M}, M), m')) \right\}_{sk, M, m'} \end{aligned}$$

The distributions are taken over the coins of $\mathbf{Sign}$ and $\mathbf{SignDerive}$. Without loss of generality, we assume that pk can be computed from sk .

The definition states that a derived signature on m' , from an honestly-generated original signature, is statistically indistinguishable from a fresh signature on m' . This implies that a derived signature on m' is indistinguishable from a signature generated independently of M . Therefore, the derived signature cannot (provably) reveal any information about M beyond what is revealed by m' . By a simple hybrid argument the same holds even if the adversary is given multiple derived signatures from M .

Moreover, Definition 2.4 requires that a derived signature look like a fresh signature even if the original signature on M is known. Hence, if for example someone quotes from a signed recommendation letter and somehow the original signed recommendation letter becomes public, it would be impossible to link the signed quote to the original signed letter. The same holds even if the signing key sk is leaked.

Thus, Definition 2.4 captures a broad range of privacy requirements for derived signatures. Earlier work in this area [35, 18, 20, 17] only considered weaker privacy requirements using more complex definitions. The simplicity and breadth of Definition 2.4 is one of our key contributions.

Definition 2.4 uses statistical indistinguishability meaning that even an unbounded adversary cannot distinguish derived signatures from newly created ones. In Appendix A, we give a definition using computational indistinguishability which is considerably more complex since the adversary needs to be given signing oracles. In the unbounded case of Definition 2.4 the adversary can simply recover a secret key sk from the public key and answer its own signature queries which greatly simplifies the definition of context hiding. All the signature schemes in this paper satisfy the statistical Definition 2.4.

As mentioned above, the context-hiding guarantee applies to all derivations that begin with an honestly-generated signature. One might imagine a scenario where a malicious signer creates a signature that passes the verification algorithm, but contains a “watermark” that allows the signer to detect if other signatures are derived from it. To prevent such attacks from malicious signers, we could alter the definition so that indistinguishability holds for any derivative that results from a signature that passed the verification algorithm.

A simpler approach to proving unforgeability. For systems that are strongly context hiding, unforgeability follows from a simpler game than that of Section 2.1. In particular, it suffices to just give the adversary the ability to obtain top level signatures signed by sk . In Appendix A, we define this simpler unforgeability game and prove equivalence to Definition 2.3 using strong context hiding.

2.3 Related Work

Early work on quotable signatures [54, 35, 44, 43, 32, 19, 23, 17] supports quoting from a single document, but does not achieve the privacy or unforgeability properties we are aiming for. For example, if *simple quoting* of messages is all that is desired, then the following folklore solution would suffice: simply sign the Merkle hash of a document. A quote represents some sub-tree of the Merkle hash; so a quoter could include enough intermediate hash nodes along with the original signature in any quote. A verifier could simply hash the quote, and then build the Merkle hash tree using the computed hash and the intermediate hashes, and compare with the original signature. Notice, however, that every quote in this scheme reveals information about the original source document. In particular, each quote reveals information about *where in the document* it appears. Thus, this simple quoting scheme is not *context hiding* in our sense.

The work whose definition is closest to what we envision is the recent work on redacted signatures of Chang et al. [23] and Brzuska et al. [17] (see also Naccache [45, p. 63] and Boneh-Freeman [14,

13] ⁴). However, there is a subtle, but fundamental difference between their definition and the privacy notion we are aiming for. In our formulation, a quoted signature should be indistinguishable from a fresh signature, even when the distinguisher is given the original signature. (We capture this by an even stronger game where a derived signature is distributed statistically close to a fresh signature.) In contrast, the definitions of [23, 17, 14, 13] do not provide the distinguisher with the original signature. Thus, it may be possible to link a quoted document to its original source (and indeed it is in the constructions of [23, 17, 14, 13]), which can have negative privacy implications. Overcoming such document linkage while maintaining unforgeability is a real technical challenge. This requires moving beyond techniques that use *nonces* to link parts of messages.

Indeed, in most prior constructions, such as [23, 17], nonces are used to prevent “mix-and-match” attacks (e.g., forming a “quote” using pieces of two different messages.) Unfortunately, these nonces reveal the history of derivation, since they cannot change during each derivation operation. Arguably, much of the technical difficulty in our current work comes precisely from the effort to meet our definition and hide the lineage. We introduce new techniques in this work which link pieces together using randomness that can be re-randomized in controlled ways.

Another line of work studies computing on authenticated data by holders of secret information. Examples include *sanitizable* signatures [44, 1, 42, 20, 18] that allow a proxy to compute signatures on related messages, but requires the proxy to have a secret key, and *incremental* signatures [4], where the signer can efficiently make small edits to his signed data. In contrast, our proposal is more along the lines of homomorphic encryption and Rivest’s vision [47], where *anyone* can compute on the authenticated data.

3 Generic Constructions for Simple Predicates

Let $\mathcal{M}$ be a finite message space. We say that a predicate $P : \mathcal{M}^* \times \mathcal{M} \rightarrow \{0, 1\}$ is a *simple* predicate if the following properties hold:

1. P is false whenever its left input is a tuple of length greater than 1,
2. P is a closed predicate (i.e., P is equal to its closure P^* ; see Section 2.1.)
3. For all $m \in \mathcal{M}$, $P(m, m) = 1$.

In this section, we present and discuss generic approaches to computing on authenticated data with respect to *any* simple predicate P . Note that the quoting of substrings or subsequences (i.e., redacting) are examples of simple predicates.

We begin with two inefficient constructions. The first takes a brute force approach that constructs long signatures that are easy to verify. The second takes an accumulator approach that constructs shorter signatures at the cost of less efficient verification. We conclude by discussing the limitations of a generic NIZK proof of knowledge approach.

⁴As acknowledged in Section 2.2 of Boneh-Freeman [13], our definitional notion is stronger than and predates the “weak context hiding” notion of [13]. Indeed, the fact that [13] uses our framework lends support to its generality, and the fact that they could not achieve our context hiding notion highlights its difficulty. Their “weak” definition, which is equivalent to [17], only ensures privacy when the original signatures remain hidden. In their system, signature derivation is deterministic and therefore once the original signatures become public it is easy to tell where the derived signature came from. Our signatures achieve full context hiding so that derived signatures remain private no matter what information is revealed. This is considerably harder and is not known how to do for the lattice-based signatures in Boneh-Freeman.

3.1 A Brute Force Construction From Any Signature Scheme

Let (G, S, V) be a signature scheme with a deterministic signing algorithm.⁵ One can construct a P -homomorphic signature scheme for any simple predicate P as follows:

KeyGen (1^λ) : The setup algorithm runs $G(1^\lambda) \rightarrow (pk, sk)$ and outputs this key pair.

Sign $(sk, m \in \mathcal{M})$: While **Sign** is simply a special case of the **SignDerive** algorithm, we will explicitly provide both algorithms here for clarity purposes.

The signature σ is the tuple $(S(sk, m), U = \{S(sk, m') \mid m' \in P^0(\{m\})\})$.

SignDerive (pk, σ, m, m') : The derived signature is computed as follows. First check that $P(m, m') = 1$. If not, then output $\perp$. Otherwise, parse $\sigma = (\sigma_1, \dots, \sigma_k)$ where σ_i corresponds to message m_i . If for any i , $V(pk, m_i, \sigma_i) = 0$, then output $\perp$. Otherwise, the signature is comprised as the set containing σ_i for all m_i such that $P(m', m_i) = 1$. Again, by default, let the first sub-signature of the output be the signature on m' .

Verify (pk, m, σ) : Parse $\sigma = (\sigma_1, \dots, \sigma_k)$. Output $V(pk, m, \sigma_1)$.

Efficiency Discussion The efficiency of the above approach depends on the message space and the predicate P . For instance, the brute force approach for signing a message of n characters, where $P(m, m')$ outputs 1 if and only if m' is a substring of m , will result in $O(n^2)$ sub-signatures (one for each of the $O(n^2)$ substrings). If one wanted to “quote” subgraphs from a graph, this approach is intractable, as a graph of n nodes will generate an exponential in n number of subgraphs.

Theorem 3.1 (Security from Any Signature) *If (G, S, V) is a secure deterministic signature scheme, then the above signature scheme is unforgeable and context-hiding.*

Proof of the above theorem is rather straightforward. The context-hiding property follows from the uniqueness of the signatures generated by the honest signing algorithms. The unforgeability property follows from the fact that an adversary cannot obtain a signature on any message not derivable from those she queried or one could use this signature to directly break the regular unforgeability of the underlying signature scheme. The correctness property is actually the most complex to verify: it requires the two restrictions on the predicate P made above.

3.2 An Accumulator-based Construction

Assumption 3.2 (RSA [48]) *Let k be the security parameter. Let a positive integer N be the product of two random k -bit primes p, q . Let e be a randomly chosen positive integer less than and relatively prime to $\phi(N) = (p - 1)(q - 1)$. Then no PPT algorithm given (N, e) and a random $y \in \mathbb{Z}_N^*$ as input can compute x such that $x^e \equiv y \pmod{N}$ with non-negligible probability.*

Lemma 3.3 (Shamir [51]) *Given $x, y \in \mathbb{Z}_n$ together with $a, b \in \mathbb{Z}$ such that $x^a = y^b$ and $\gcd(a, b) = 1$, there is an efficient algorithm for computing $z \in \mathbb{Z}_n$ such that $z^a = y$.*

⁵Given a signature scheme with a probabilistic signing algorithm, one can convert it to a scheme with a deterministic signing algorithm by: (1) including a pseudorandom function (PRF) seed as part of the secret key and (2) during the signing algorithm, applying this PRF to the message and using the output as the randomness in the signature. Given any signature scheme, one can also construct a PRF.

Theorem 3.4 (Prime Number Theorem) Define $\pi(x)$ as the number of primes no larger than x . For $x > 1$,

$$\pi(x) > \frac{x}{\lg x}.$$

Consider the following RSA accumulator solution which supports short signatures, but the computation required to derive a new signature is expensive. Let P be any univariate predicate with the above restrictions.

We now describe the algorithms. While **Sign** is simply a special case of the **SignDerive** algorithm, we will explicitly provide both algorithms here for clarity purposes.

KeyGen(1^λ) : The setup algorithm chooses N as a 20λ -bit RSA modulus and a random value $a \in \mathbb{Z}_N$. It also chooses a hash function H_p that maps arbitrary strings to 2λ -bit prime numbers, e.g., [34], which we treat as a random oracle.⁶ Output the public key $pk = (H_p, N, a)$ and keep as the secret key sk , the factorization of N .

Sign($sk, m \in \mathcal{M}$) : Let $U = P^0(\{m\}) = \{m' \mid m' \in \mathcal{M} \text{ and } P(m, m') = 1\}$. Compute and output the signature as

$$\sigma := a^{1/(\prod_{u_i \in U} H_p(u_i))} \mod N.$$

SignDerive(pk, σ, m, m') : The derivation is computed as follows. First check that $P(m, m') = 1$. If not, then output $\perp$. Otherwise, let $U' = P^0(\{m'\})$. Compute and output the signature as

$$\sigma' := \sigma^{\prod_{u_i \in U - U'} H_p(u_i)} \mod N.$$

Thus, the signature is of the form $a^{1/\prod_{u_i \in U'} H_p(u_i)} \mod N$.

Verify(pk, m, σ) : Accept if and only if $a = \sigma^{\prod_{u_i \in U} H_p(u_i)} \mod N$ where $U = P^0(m)$.

Efficiency Discussion In the above scheme, signatures require only one element in $\mathbb{Z}_N^*$. However, the cost of signing depends on P and the size of the message space. For example, computing an ℓ -symbol quote from an n -symbol message requires $O(n(n - \ell))$ evaluations of $H_p()$ and $O(n(n - \ell))$ modular exponentiations. The prime search component of H_p will likely be the dominating factor. Verification requires $O(\ell^2)$ evaluations of $H_p()$ and $O(\ell^2)$ modular exponentiations, for an ℓ -symbol quote. Thus, this scheme optimizes on space, but may require significant computation.

Theorem 3.5 (Security under RSA) If the RSA assumption holds, then the above signature scheme is unforgeable and context-hiding in the random oracle model.

We provide a proof of above theorem by showing the following lemmas.

Lemma 3.6 (Context-Hiding) The homomorphic signature scheme from §3.2 is strongly context-hiding.

Proof. This property is derived from the fact that a signature on any given message is deterministic. Let the public key PK be (H_p, N, a) and challenge be any m, m' where $P(m, m') = 1$. Let $U =$

⁶We choose our modulus and hash output lengths to obtain λ -bit security based on the recent estimates of [53].

$P^0(m)$ and $U' = P^0(m')$. Observe that

$$\begin{aligned}
\mathbf{Sign}(sk, m) &= \sigma = a^{1/\prod_{u \in U} H_p(u)} \mod N \\
\mathbf{Sign}(sk, m') &= \sigma_0 = a^{1/\prod_{u' \in U'} H_p(u')} \mod N \\
\mathbf{SignDerive}(pk, (\sigma, m), m') &= \sigma^{\prod_{u \in U - U'} H_p(u)} \mod N \\
&= \left[a^{1/\prod_{u \in U} H_p(u)} \right]^{\prod_{u \in U - U'} H_p(u)} \mod N \\
&= a^{1/\prod_{u' \in U'} H_p(u')} \mod N \\
&= \sigma_0
\end{aligned}$$

Because $\mathbf{Sign}(sk, m')$ and $\mathbf{SignDerive}(pk, (\sigma, m), m')$ are identical, for any adversary $\mathcal{A}$, the probability that $\mathcal{A}$ distinguishes the two is exactly $1/2$, and so the advantage in the strong context hiding game is 0. $\square$

Lemma 3.7 (Unforgeability) *If the RSA assumption holds, then the Section 3.2 homomorphic signature scheme is unforgeable in the **Unforg** game in the random oracle model.*

Proof. Our reduction only works on certain types of RSA challenges, as in [34]. In particular, this reduction only attempts to solve RSA challenges (N, e^*, y) where e^* is an odd prime. Fortunately, good challenges will occur with non-negligible probability. We know that e^* is less than and relatively prime to $\phi(N) < N$, which implies it cannot be 2. We also know, by Theorem 3.4, that the number of primes that are less than N is at least $\frac{N}{\lg N}$. Thus, a loose bound on the probability of e^* being a prime is $\geq (\frac{N}{\lg N})/N = \frac{1}{\lg N} = \frac{1}{20\lambda}$.

Now, we describe the reduction. Our proof first applies Lemma A.4, which allows us to only consider adversaries $\mathcal{A}$ that ask queries to *Sign* oracle in the **NHU** game. Moreover, suppose adversary $\mathcal{A}$ queries the random oracle H_p on at most s unique inputs. Without loss of generality, we will assume that all queries to this deterministic oracle are unique and that whenever *Sign* is called on message M , then H_p is automatically called with all unique substrings of M . Suppose an adversary $\mathcal{A}$ can produce a forgery with probability ϵ in the **NHU** game; then we can construct an adversary $\mathcal{B}$ that breaks the RSA assumption (with odd prime e^*) with probability ϵ/s minus a negligible amount as follows.

On input an RSA challenge (N, e^*, y) , $\mathcal{B}$ proceeds as follows:

Setup $\mathcal{B}$ chooses 2λ -bit distinct prime numbers $e_1, e_2, \dots, e_{s-1}$ at random, where all $e_i \neq e^*$. Denote this set of primes as E . Next, $\mathcal{B}$ makes a random guess of $i^* \in [1, s]$ and saves this value for later. Then it sets

$$a := y^{\prod_{e_i \in E} e_i}.$$

Finally, $\mathcal{B}$ give the public key $PK = (N, a)$ to $\mathcal{A}$ and will answer its queries to random oracle H_p interactively as described below.

Queries Proceeding adaptively, $\mathcal{B}$ answers the oracle and sign queries made by $\mathcal{A}$ as follows:

1. $H_p(x)$: When $\mathcal{A}$ queries the random oracle for the j th time, $\mathcal{B}$ responds with e^* if $j = i^*$, with e_j if $j < i^*$ and e_{j-1} otherwise. Recall that we stipulated that each call to H_p was unique. Denote x^* as the input where $H_p(x^*) = e^*$.

2. $Sign(M)$: Let $U = P^0(M)$. If $x^* \in U$, then $\mathcal{B}$ aborts the simulation. Otherwise, $\mathcal{B}$ calls H_p on all elements of U not previously queried to H_p . Let $\mathbf{primes}(U)$ denote the set of primes derived by calling H_p on the strings of U . Then, it computes the signature as $\sigma := y^{\prod_{e_i \in (E - \mathbf{primes}(U))} e_i} \bmod N$ and returns (M, σ) .

Response Eventually, $\mathcal{A}$ outputs a valid message-signature pair (M, σ) , where M is not a derivative of an element returned by $Sign$. If M was not queried to H_p or if $M \neq x^*$, then $\mathcal{B}$ aborts the simulation. Otherwise, let $U = P^0(x^*) - \{x^*\}$ and $\mathbf{primes}(U)$ denote the set of primes derived by calling H_p on the strings of U . It holds that $a^{1/\prod_{e_i \in \mathbf{primes}(U)} e_i} = y^{\prod_{e_i \in E - \mathbf{primes}(U)} e_i} = \sigma^{e^*} \bmod N$. Since $y, \sigma \in \mathbb{Z}_N$ and $\gcd(e^*, \prod_{e_i \in E - \mathbf{primes}(U)} e_i) = 1$ (recall, they are all distinct primes), then $\mathcal{B}$ can apply the efficient algorithm from Lemma 3.3 to obtain a value $z \in \mathbb{Z}_N$ such that $z^{e^*} = y \bmod N$. $\mathcal{B}$ outputs z as the solution to the RSA challenge.

Analysis We now argue that any successful adversary $\mathcal{A}$ against our scheme will have success in the game presented by $\mathcal{B}$. To do this, we first define a sequence of games, where the first game models the real security game and the final game is exactly the view of the adversary when interacting with $\mathcal{B}$. We then show via a series of claims that if $\mathcal{A}$ is successful against Game j , then it will also be successful against Game $j + 1$.

Game 1: The same as Game **NHU**, with the exception that at the beginning of the game $\mathcal{B}$ guesses an index $1 \leq i^* \leq s$ and e^* is the response of the i^* th query to H_p .

Game 2: The same as Game 1, with the exception that $\mathcal{A}$ fails if any output of H_p is repeated.

Game 3: The same as Game 2, with the exception that $\mathcal{A}$ fails if it outputs a valid forgery (M, σ) where M was not queried to H_p .

Game 4: The same as Game 3, with the exception that $\mathcal{A}$ fails if it outputs a valid forgery (M, σ) where $M \neq x^*$.

Notice that Game 4 is exactly the view of the adversary when interacting with $\mathcal{B}$. We complete this argument by linking the probability of $\mathcal{A}$'s success in these games via a series of claims. The only non-negligible probability gap comes between Games 3 and 4, where there is a factor $1/s$ loss.

Define $\mathbf{Adv}_{\mathcal{A}}[\text{Game } x]$ as the advantage of adversary $\mathcal{A}$ in Game x .

Claim 3.8 *If H_p is a truly random function, then*

$$\mathbf{Adv}_{\mathcal{A}}[\text{Game } 1] = \mathbf{Adv}_{\mathcal{A}}[\text{Game } \mathbf{NHU}].$$

Proof. The value e^* was chosen independently at random by the RSA challenger, just as H_p would have done. $\square$

Claim 3.9 *If H_p is a truly random function, then*

$$\mathbf{Adv}_{\mathcal{A}}[\text{Game } 2] = \mathbf{Adv}_{\mathcal{A}}[\text{Game } 1] - \frac{2s^2\lambda}{2^{2\lambda}}.$$

Proof. Consider the probability of a repeat occurring when s 2λ -bit primes are chosen at random. By Theorem 3.4, we know that there are at least $2^{2\lambda}/(2\lambda)$ 2λ -bit primes. Thus, a repeat will occur with probability $< \sum^s s/(2^{2\lambda}/2\lambda) = 2s^2\lambda/2^{2\lambda}$, which is negligible since s must be polynomial in λ . $\square$

Claim 3.10 *If H_p is a truly random function, then*

$$\mathbf{Adv}_{\mathcal{A}}[\text{Game 3}] = \mathbf{Adv}_{\mathcal{A}}[\text{Game 2}] - \frac{2\lambda}{2^{2\lambda}}.$$

Proof. If M was never queried to H_p , then σ can only be a valid forgery if $\mathcal{A}$ guessed the 2λ -bit prime that H_p would respond with on input M . By Theorem 3.4, there are at least $2^{2\lambda}/2\lambda$ such primes and thus the probability of $\mathcal{A}$'s correct guess is at most $2\lambda/2^{2\lambda}$, which is negligible. $\square$

Claim 3.11

$$\mathbf{Adv}_{\mathcal{A}}[\text{Game 4}] = \frac{\mathbf{Adv}_{\mathcal{A}}[\text{Game 3}]}{s}.$$

Proof. At this point in our series of games, we conclude that $\mathcal{A}$ forges on one of the s queries to H_p and that $1 \leq i^* \leq s$ was chosen at random. Thus, the probability that $\mathcal{A}$ forges on the i^* th query is $1/s$. $\square$

This completes our proof. $\square$

3.3 On the Limitations of Using a Generic NIZK Proof of Knowledge Approach

Another general approach that one might be tempted to try is to use an NIZK [9] proof of knowledge system to generate a signature on m' by proving that one knows a signature on some m such that $P(m, m')$ holds. Unfortunately, this approach has the standard drawback of generality in that it requires circuit-based (non black-box) reductions. In particular, the statements to prove in non-interactive zero-knowledge require transforming the circuits of the signature scheme and the quoting predicate into an instance of Hamiltonian circuit or 3-SAT. Even if one were to tailor an NIZK proof of knowledge for these specific statements and therefore avoid costly reductions, another problem emerges with re-quoting. When a quote is re-quoted, then the same process happens for both the original signature scheme circuit, the predicate, *and* the proof system. Aside from the inefficiency, using standard NIZKPoK systems would leak information about the size of the original message and quotes, and therefore would not satisfy our context hiding property⁷.

4 A Powers-of-2 Construction for Quoting Substrings

We begin by describing our algebraic setting.

4.1 Bilinear Groups and the CDH Assumption

Bilinear Groups and the CDH Assumption. Let $\mathbb{G}$ and $\mathbb{G}_T$ be groups of prime order p . A *bilinear map* is an efficient mapping $\mathbf{e} : \mathbb{G} \times \mathbb{G} \rightarrow \mathbb{G}_T$ which is both: (*bilinear*) for all $g \in \mathbb{G}$ and $a, b \leftarrow \mathbb{Z}_p$, $\mathbf{e}(g^a, g^b) = \mathbf{e}(g, g)^{ab}$; and (*non-degenerate*) if g generates $\mathbb{G}$, then $\mathbf{e}(g, g) \neq 1$. We will focus on the Computational Diffie-Hellman assumption in these groups.

Assumption 4.1 (CDH [26]) *Let g generate a group $\mathbb{G}$ of prime order $p \in \Theta(2^\lambda)$. For all PPT adversaries $\mathcal{A}$, the following probability is negligible in λ : $\Pr[a, b, \leftarrow \mathbb{Z}_p; z \leftarrow \mathcal{A}(g, g^a, g^b) : z = g^{ab}]$.*

⁷Using non-interactive CS-proofs [40] in the random oracle model may reduce the size of the proof, but we do not know how to avoid leaking the size of the theorem statement which also violates the context hiding property.

4.2 The Quoting Construction

We now provide our main construction for quoting substrings in a text document. It achieves the best time/space efficiency trade-off to our knowledge for this problem. We will have two different types of signatures called Type I and Type II, where a Type I signature can be quoted down to another Type I or Type II signature. A Type II signature cannot be quoted any further, but will be a shorter signature. The quoting algorithm will allow us to quote anything that is a substring of the original message. We point out that the Type I, II signatures of this system conform to the general framework given in Section 2. In particular, we can view a message M as a pair $(t, m) \in \{0, 1\}, \{0, 1\}^*$. The bit t will identify the message as being Type I or Type II (assume $t = 1$ signifies Type I signatures) and m will be the quoted substring. The predicate

$$P(M = (t, m), M' = (t', m')) = \begin{cases} 1 & \text{if } t = 1 \text{ and } m' \text{ is a substring of } m; \\ 0 & \text{otherwise.} \end{cases}$$

The bit t' will indicate whether the new message is Type I or II (i.e., whether the system can quote further.) We note that this description allows an attacker to distinguish between any Type I signature from any Type II signature since the “type bit” of the messages will be different and thus they will technically be two different messages even if the substring components are equal. For this reason we will only need to prove context hiding between messages of Type I or Type II, but not across types. In general, flipping the bit t will not result in a valid signature of a different type on the same core message, because the format will be wrong; however, moving from a Type I to a Type II on the same core message is not considered a forgery since Type II signatures can be legally derived from Type I.

For presentational clarity, we will split the description of our quoting algorithm into two quoting algorithms for quoting to Type I and to Type II signatures; likewise we will split the description of our verification algorithm into two separate verification algorithms, one for each type of signature. The type of signature used or created (i.e., bit t) will be implicit in the description.

Notation: We use notation $m_{i,j}$ to denote the substring of m of length j starting at position i .

Intuition: We begin by giving some intuition. We design Type I signatures that allow re-quoting and Type II signatures that cannot be further quoted, but are ultra-short. For an original message of length n , our signature structure should be able to accommodate starting at any position $1 \leq i \leq n$ and quoting any length $1 \leq \ell \leq (n - i + 1)$ substring.⁸

To (roughly) see how this works for a message of length n , visualize $(n + 1)$ columns with $(\lceil \lg n \rceil + 2)$ rows as in Figure 1. The columns correspond to the characters of the message, so if the 14-character message is “abcdefghijklmn” then there are 15 columns, with a character in between each column. The rows correspond to the numbers $\lg n$ down to 0, plus an extra row at the bottom.⁹ Each location in the matrix (except along the bottom-most row) contains one or more out-going arrows. We’ll establish rules for when these arrows exist and where each arrow ends shortly.

A Type II quote will trace a $(\lg n + 1)$ -length path on these arrows through this matrix starting in a row (with outgoing arrows) of the column that begins the quote and ending in the lowest row of the first column after the quote ends. The starting row corresponds to the largest power of two less than or equal to the length of the desired quote. E.g., to quote “bcdef”, start in row 2 immediately to the left of ‘b’ (because $2^2 = 4$ is the largest power of two less than 5) and end in

⁸Technically, our predicate $P(m, m')$ will take the quote from the first occurrence of substring m' in m , but for the moment imagine that we allowed quoting from anywhere in m .

⁹The lowest row is intentionally not assigned a number. The second lowest row is row 0. We do this so that row i can correspond to a jump of length 2^i .

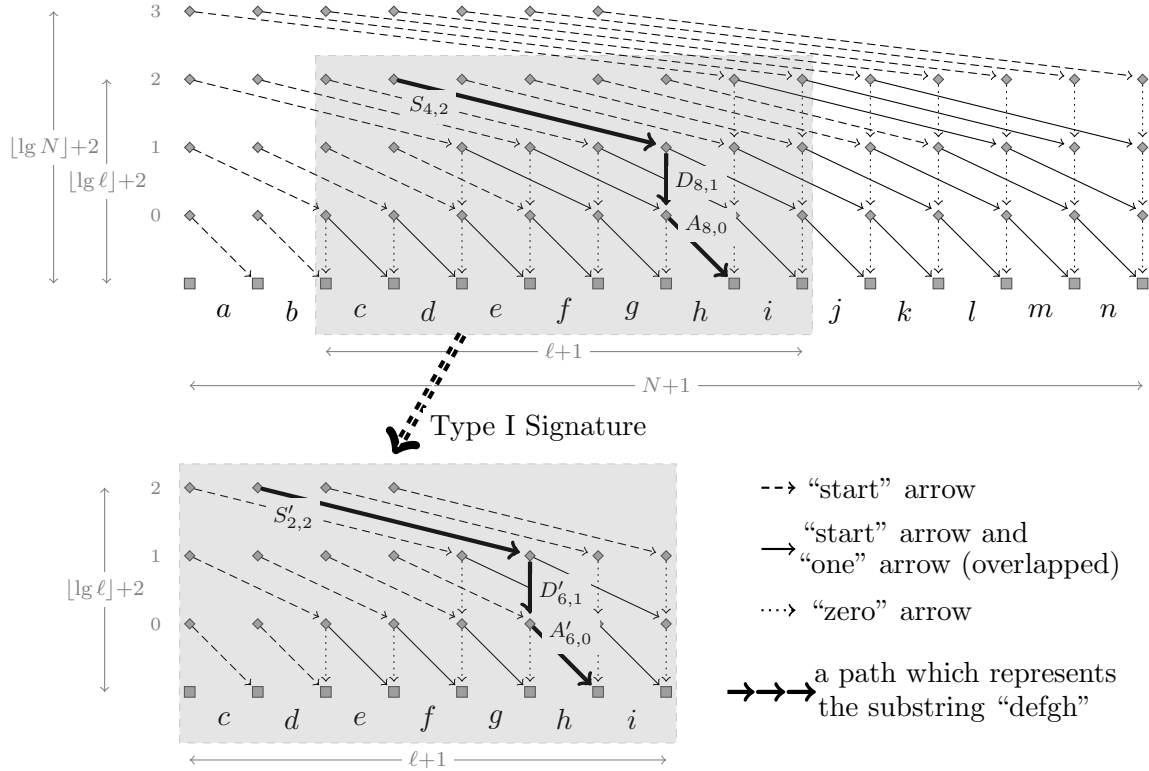


Figure 1: The top diagram represents a signature on “abcdefghijklmn” with length $N = 14$. Each arrow corresponds to some group elements in the construction. Logically, whenever the elements corresponding to an arrow are included in a quoted signature, the characters underneath this arrow are included in the quoted message. The bold path through the top diagram shows how to construct a Type II signature on “defgh”; it is very short, but cannot be re-quoted. The gray box in this figure shows how to construct a Type I signature on “cdefghi” of length $\ell = 7$; it includes all the arrows in the lower figure and can be re-quoted. A technical challenge is to enforce that following the arrows is the only way to form a valid signature. Details are below.

row 0 immediately to the right of ‘f’. Intuitively, taking an arrow over a character includes it in the quote. A Type II quote on “defgh” is illustrated in Figure 1.

A technical challenge is to make this a $O(\lg n)$ -length path rather than a $O(n)$ -length path. To do this, the key insight is to view the length of any possible quote as the sum of powers of two and to allow arrows that correspond to covering the quote in pieces of size corresponding to one operand of the sum at a time. Each location (i_c, i_r) in the matrix (except the bottom-most row) contains:

- a “start” arrow: an arrow that goes down one row and over 2^{i_r} columns ending in $(i_c + 2^{i_r}, i_r - 1)$, if this end point is in the matrix. This adds all characters from position i_c to $i_c + 2^{i_r} - 1$ to the quoted substring; effectively adding the largest power-of-two-length prefix of the quote characters. This arrow indicates that the quote starts here. These are represented as $S_{i,j}, \widehat{S}_{i,j}$ pairs in our construction.
- a “one” arrow: operate similarly to start arrows and used to include characters after a start arrow includes the quote prefix. These are represented as $A_{i,j}, \widehat{A}_{i,j}$ pairs in our construction.

- a “zero” arrow: an arrow that goes straight down one row ending in $(i_c, i_r - 1)$. This does not add any characters to the quoted substring. These are represented as $D_{i,j}, \widetilde{D_{i,j}}$ pairs in our construction.

A Type II quote always starts with a start arrow and then contains one and zero arrows according to the binary representation of the length of the quote. In our example of original message “abcdefghijklmn”, we have 15 columns and 5 rows. We will logically divide our desired substring of “bcdef” (length $5 = 2^2 + 2^0 = 4 + 1$) into its powers-of-two components “bcde” (length $4 = 2^2$) and “f” (length $1 = 2^0$). To form the Type II quote, we start in row 2 (since $4 = 2^2$) of column 2 (to the left of ‘b’) and take the start arrow ($S_{2,2}$) to row 1 of column 7, take the zero arrow ($D_{7,1}$) to row 0 of column 7, and then take the one arrow ($A_{7,0}$) to the lowest row of column 8. The arrows “pass over” the characters “bcdef”. Figure 1 illustrates this for quote “defgh”.

For a quote of length ℓ , the elements on this $O(\lg \ell)$ -length path of arrows form a very short Type II signature. For Type I signatures, we include all the elements corresponding to all arrows that make connections within the columns corresponding to the quote. We illustrate this in Figure 1. This allows quoting of quotes with a signature size of $O(\ell \lg \ell)$.

It is essential for security that the signature structure and data algorithm enforce that the quoting algorithm be used and not allow an attacker to “splice” together a quote from different parts of the signature. We realize this by adding in random “chaining” variables. In order to cancel these out and get a well formed Type II quote a user must intuitively follow the prescribed procedure (i.e., following the arrows is the only way to form a valid quote.)

The Construction: We now describe our algorithms. While **Sign** is simply a special case of the **SignDerive** algorithm, we will explicitly provide both algorithms here for clarity purposes.

KeyGen(1^λ) : The algorithm selects a bilinear group $\mathbb{G}$ of prime order $p > 2^\lambda$ with generator g . Let L be the maximum message length supported and denote $n = \lfloor \lg(L) \rfloor$. Let $H : \{0, 1\}^* \rightarrow \mathbb{G}$ and $H_s : \{0, 1\}^* \rightarrow \mathbb{G}$ be the description of two hash functions that we model as random oracles. Choose random $z_0, \dots, z_{n-1}, \alpha \in \mathbb{Z}_p$. The secret key is $(z_0, \dots, z_{n-1}, \alpha)$ and the public key is:

$$PK = (H, H_s, g, g^{z_0}, \dots, g^{z_{n-1}}, \mathbf{e}(g, g)^\alpha).$$

Sign($sk, M = (t, m) \in \{0, 1\} \times \Sigma^{\ell \leq L}$) : If $t = 1$, signatures produced by this algorithm are Type I as described below. If $t = 0$, the Type II signature can be obtained by running this algorithm and then running the Quote-Type II algorithm below to obtain a quote on the entire message. The message space is treated as $\ell \leq L$ symbols from alphabet Σ .

Recall: we use notation $m_{i,j}$ to denote the substring of m of length j starting at position i .

For $i = 3$ to $\ell + 1$ and $j = 0$ to $\lfloor \lg(i - 1) - 1 \rfloor$, choose random values $x_{i,j} \in \mathbb{Z}_p$. These will serve as our random “chaining” variables, and they should all “cancel” each other out in our short Type II signatures. By definition, set $x_{i,-1} := 0$ for all $i = 1$ to $\ell + 1$.

A signature is comprised of the following values for $i = 1$ to ℓ and $j = 0$ to $\lfloor \lg(\ell - i + 1) \rfloor$, for randomly chosen values $r_{i,j} \in \mathbb{Z}_p$:

$$\begin{aligned} & \text{[start arrow: start and include power } j] \\ S_{i,j} &= g^\alpha g^{-x_{i+2j,j-1}} H_s(m_{i,2j})^{r_{i,j}} \quad , \quad \widetilde{S_{i,j}} = g^{r_{i,j}} \end{aligned}$$

Together with the following values for $i = 3$ to ℓ and $j = 0$ to $\min(\lfloor \lg(i-1) \rfloor - 1, \lfloor \lg(\ell-i+1) \rfloor)$, for randomly chosen values $r'_{i,j} \in \mathbb{Z}_p$:

[one arrow: include power j and decrease j]

$$A_{i,j} = g^{x_{i,j}} g^{-x_{i+2j,j-1}} H(m_{i,2j})^{r'_{i,j}}, \quad \widetilde{A_{i,j}} = g^{r'_{i,j}}$$

Together with the following values for $i = 3$ to $\ell+1$ and $j = 0$ to $\lfloor \lg(i-1) \rfloor - 1$, for randomly chosen values $r''_{i,j} \in \mathbb{Z}_p$:

[zero arrow: decrease j]

$$D_{i,j} = g^{x_{i,j}} g^{-x_{i,j-1}} g^{z_j r''_{i,j}}, \quad \widetilde{D_{i,j}} = g^{r''_{i,j}}$$

We provide an example of how to form Type II signatures from this construction shortly. To see why our $A_{i,j}$ and $D_{i,j}$ values start at $i = 3$, note that Type II quotes at position i of length $2^0 = 1$ symbol include only the $S_{i,0}$ value, where the $x_{i,0-1}$ term is 0 by definition. Type II quotes at position i of length $2^1 = 2$ symbols include the $S_{i,1}$ value plus an additional $D_{i+2,0}$ term to cancel out the $x_{i+2,0}$ value (leaving only $x_{i+2,-1} = 0$.) Quotes at position i of length $2^1 + 1 = 3$ symbols include the $S_{i,1}$ value plus an additional $A_{i+2,0}$ term to cancel out the $x_{i+2,0}$ value (leaving only $x_{i+3,-1} = 0$.) Since we index strings from position 1, the first position to include an $A_{i,j}$ or $D_{i,j}$ value is $i + 2 = 3$.

SignDerive($pk, \sigma, M = (t, m), M' = (t', m')$) : If $P(M, M') = 0$, output $\perp$. Otherwise, if $t' = 1$, output Quote-Type I(PK, σ, m, m'); if $t' = 0$, output Quote-Type II(PK, σ, m, m'), where these algorithms are defined below.

Quote-Type I(pk, σ, m, m') : The quote algorithm takes a Type I signature and produces another Type I signature that maintains the ability to be quoted again. Intuitively, this operation will simply find a substring m' in m , keep only the components associated with this substring and re-randomize them all (both the $x_{i,j}$ and $r_{i,j}$ terms in every component.)

If m' is not a substring of m , then output $\perp$. Otherwise, let $\ell' = |m'|$. Determine the first index k at which substring m' occurs in m . Parse σ as a collection of $S_{i,j}, \widetilde{S_{i,j}}, A_{i,j}, \widetilde{A_{i,j}}, D_{i,j}, \widetilde{D_{i,j}}$ values, exactly as would come from **Sign** with $\ell = |m|$.

First, we choose re-randomization values (to re-randomize the $x_{i,j}$ terms of σ .) For $i = 2$ to $\ell' + 1$ and $j = 0$ to $\lfloor \lg(i-1) \rfloor - 1$, choose random values $y_{i,j} \in \mathbb{Z}_p$. Set $y_{i,-1} := 0$ for all $i = 1$ to $\ell' + 1$. Later, we will choose $t_{i,j}$ values to re-randomize the $r_{i,j}$ terms of σ .

The quote signature σ' is comprised of the following values:

For $i = 1$ to ℓ' and $j = 0$ to $\lfloor \lg(\ell' - i + 1) \rfloor$, for randomly chosen $t_{i,j} \in \mathbb{Z}_p$:

$$S'_{i,j} = S_{i+k-1,j} \cdot g^{-y_{i+2j,j-1}} H_s(m_{i+k-1,2j})^{t_{i,j}}, \quad \widetilde{S'_{i,j}} = \widetilde{S_{i+k-1,j}} \cdot g^{t_{i,j}}$$

Together with the following values for $i = 3$ to ℓ' and $j = 0$ to $\min(\lfloor \lg(i-1) \rfloor - 1, \lfloor \lg(\ell' - i + 1) \rfloor)$, for randomly chosen $t'_{i,j} \in \mathbb{Z}_p$:

$$A'_{i,j} = A_{i+k-1,j} \cdot g^{y_{i,j}} g^{-y_{i+2j,j-1}} H(m_{i+k-1,2j})^{t'_{i,j}}, \quad \widetilde{A'_{i,j}} = \widetilde{A_{i+k-1,j}} \cdot g^{t'_{i,j}}$$

Together with the following values for $i = 3$ to $\ell' + 1$ and $j = 0$ to $\lfloor \lg(i-1) - 1 \rfloor$, for randomly chosen $t''_{i,j} \in \mathbb{Z}_p$:

$$D'_{i,j} = D_{i+k-1,j} \cdot g^{y_{i,j}} g^{-y_{i,j-1}} g^{z_j t''_{i,j}}, \quad \widetilde{D'}_{i,j} = \widetilde{D_{i+k-1,j}} \cdot g^{t''_{i,j}}$$

Quote-Type II(pk, σ, m, m') : The quote algorithm takes a Type I signature and produces a Type II signature. If $P(m, m') \neq 1$, then output $\perp$.

A quote is computed from one start value and logarithmically many subsequent pieces depending on the bits of $|m'|$. All signature pieces must be re-randomized to prevent content-hiding attacks.

Consider the length ℓ' written as a binary string. Let β' be the largest index of $\ell' = |m'|$ that is set to 1, where *we start counting with zero as the least significant bit*. That is, set $\beta' = \lfloor \lg(\ell') \rfloor$. Select random values $v, v_{\beta'-1}, \dots, v_0 \in \mathbb{Z}_p$. Set the start position as $B := S_{k,\beta'}$ and

$k' := k + 2^{\beta'}$. Then, from $j = \beta' - 1$ down to 0, proceed as follows:

- If the j th bit of ℓ' is 1, set $B := B \cdot A_{k',j} \cdot H(m_{k',2^j})^{v_j}$, set $k' := k' + 2^j$, and $Z_j := \widetilde{A_{k',j}} \cdot g^{v_j}$;
- If the j th bit of ℓ' is 0, set $B := B \cdot D_{k',j} \cdot g^{z_j v_j}$ and $Z_j := \widetilde{D_{k',j}} \cdot g^{v_j}$.

To end, re-randomize as $B := B \cdot H_s(m_{k,2^\beta})^v$ and $\widetilde{S} := \widetilde{S_{k,\beta}} \cdot g^v$; output the quote as

$$\sigma' = (B, \widetilde{S}, Z_{\beta-1}, \dots, Z_0)$$

Verify($pk, M = (t, m), \sigma$) : If $t = 1$, output Verify-Type I(pk, m, σ). Otherwise, output Verify-Type II(pk, m, σ), where these algorithms are defined immediately below.

Verify-Type I(pk, m, σ) : Parse σ as the set of $S_{i,j}, \widetilde{S_{i,j}}, A_{i,j}, \widetilde{A_{i,j}}, D_{i,j}, \widetilde{D_{i,j}}$. Let $\ell = |m|$.

Let $X_{i,j}$ denote $e(g, g)^{x_{i,j}}$. We can compute these values as follows. The value $X_{i,-1} = 1$, since for all $i = 1$ to $\ell + 1$, $x_{i,-1} = 0$. For $i = 3$ to $\ell + 1$ and $j = 0$ to $\lfloor \lg(i-1) - 1 \rfloor$, we compute $X_{i,j}$ in the following manner: Let $I = i - 2^{j+1}$ and $J = j + 1$. Next, compute $X_{i,j} = (e(g, g)^\alpha \cdot e(H_s(m_{I,2^J}), \widetilde{S_{I,J}})) / e(S_{I,J}, g)$. The verification accepts if and only if all of the following hold:

- for $i = 3$ to ℓ and $j = 0$ to $\min(\lfloor \lg(i-1) - 1 \rfloor, \lfloor \lg(\ell - i + 1) \rfloor)$,

$$e(A_{i,j}, g) = X_{i,j} / X_{i+2^j, j-1} \cdot e(H(m_{i,2^j}), \widetilde{A_{i,j}})$$

- and for $i = 3$ to $\ell + 1$ and $j = 0$ to $\lfloor \lg(i-1) - 1 \rfloor$, $e(D_{i,j}, g) = X_{i,j} / X_{i,j-1} \cdot e(g^{z_j}, \widetilde{D_{i,j}})$.

Verify-Type II(pk, m, σ) : We give the verification algorithm for Type II signatures. Parse σ as $(B, \widetilde{S}, Z_{\beta-1}, \dots, Z_0)$. Let $\ell = |m|$ and β be the index of the highest bit of ℓ that is set to 1. If σ does not include exactly β Z_i values, reject. Set $C := 1$ and $k = 1$. From $j = \beta - 1$ down to 0, proceed as follows:

- If the j th bit of ℓ is 1, set $C := C \cdot e(H(m_{k,2^j}), Z_j)$ and $k := k + 2^j$;
- If the j th bit of ℓ is 0, set $C := C \cdot e(g^{z_j}, Z_j)$.

Accept if and only if $e(B, g) = e(g, g)^\alpha \cdot e(H_s(m_{1,2^\beta}), \widetilde{S}) \cdot C$.

Theorem 4.2 (Security under CDH) *If the CDH assumption holds in $\mathbb{G}$, then the above quotable signature scheme is selectively quote unforgeable and context-hiding in the random oracle model.*

Efficiency Discussion This construction presents the best known balance between time and space complexity. The quotable (Type I) signatures require $O(\ell \lg \ell)$ elements in $\mathbb{G}$ for a message of length ℓ . The group elements in both types of signatures are elements of $\mathbb{G}$, and not the target group $\mathbb{G}_T$. Typically, elements of the base group are significantly smaller than elements of the target group. Computing quotes requires $O(\ell \lg \ell)$ modular exponentiations for a quote of length ℓ for re-randomization. Similarly, verification also requires $O(\ell \lg \ell)$ pairings.

The non-quotable (Type II) signatures require only $O(\lg \ell)$ elements in $\mathbb{G}$. Computing quotes is very efficient as it requires only $O(\lg \ell)$ modular exponentiations for a quote of length ℓ for re-randomization. Similarly, verification requires only $O(\lg \ell)$ pairings.

On Removing the Random Oracle and Obtaining Full Security The quoting construction above is provably selectively secure in the random oracle model. We now suggest a few potential avenues for adapting the above construction to full security in the standard model. First, with an eye to remove the random oracle, we observe that our signatures share many properties with the private keys of hierarchical identity-based encryption (HIBE) schemes. To remove the random oracle, while remaining under a selective definition, one might use the Boneh-Boyen techniques [10] to instantiate $H(m) = g^m h$, where $h \in \mathbb{G}$ is added to the public key and there is a method for mapping the message space to $\mathbb{Z}_p$. Similarly, one might remove the random oracle by instantiating H with the Waters hash [56] and applying his proof techniques. This can be viewed as a full security construction with a reduction to the concrete security parameter by roughly a factor of $(1/O(q))^{\lg \ell}$, where q is the number of signing queries and ℓ is the length of the quote. A direction for achieving full security could be the recent “Dual System” techniques introduced by Waters [57]. One obstacle in adapting the Waters system is that it contains “tags” in the private key structure, which would likely make our re-randomization step difficult for our context hiding property. Lewko and Waters [38] recently removed the tags, which may make their techniques and construction more suitable for our application. One drawback in using their HIBE techniques to construct signatures is that even the signatures resulting from their construction require (slightly non-standard) *decisional* complexity assumptions. Thus, it is unknown how to balance time/space efficiently while achieving full security in the standard model from a simple computational assumption such as CDH.

4.3 Security Analysis

We now provide a proof of Theorem 4.2 by showing the following lemmas.

Lemma 4.3 (Strong Context-Hiding) *The Section 4 quotable signature scheme is strongly context-hiding.*

Proof. Given any two challenge messages $M = (t, m)$, $M' = (t', m')$ such that $P(M, M') = 1$, we claim that whether $t' = 1$ or 0, **SignDerive**(pk, σ, M', M) has an identical distribution to that of **Sign**(sk, M), which implies that the two distributions are statistically close.

$$\begin{aligned} & \{(SK, \sigma \leftarrow \mathbf{Sign}(SK, M), \mathbf{Sign}(SK, M')\}_{SK, M, M'} \\ & \{(SK, \sigma \leftarrow \mathbf{Sign}(SK, M), \mathbf{SignDerive}(PK, \sigma, M, M')\}_{SK, M, M'} \end{aligned}$$

Let ℓ, ℓ' denote $|m|$ and $|m'|$ respectively. Let $\Gamma = \min(\lfloor \lg(i-1) - 1 \rfloor, \lfloor \lg(\ell - i + 1) \rfloor)$. **Sign**(SK, M) is composed of the following values:

$$\begin{aligned} S_{i,j} &= g^\alpha g^{-x_{i+2j,j-1}} H_s(m_{i,2j})^{r_{i,j}}, & \widetilde{S}_{i,j} &= g^{r_{i,j}}, & \text{for } i = 1 \text{ to } \ell \text{ and } j = 0 \text{ to } \lfloor \lg(\ell - i + 1) \rfloor \\ A_{i,j} &= g^{x_{i,j}} g^{-x_{i+2j,j-1}} H(m_{i,2j})^{r'_{i,j}}, & \widetilde{A}_{i,j} &= g^{r'_{i,j}}, & \text{for } i = 3 \text{ to } \ell \text{ and } j = 0 \text{ to } \Gamma \\ D_{i,j} &= g^{x_{i,j}} g^{-x_{i,j-1}} g^{z_j r''_{i,j}}, & \widetilde{D}_{i,j} &= g^{r''_{i,j}}, & \text{for } i = 3 \text{ to } \ell + 1 \text{ and } j = 0 \text{ to } \lfloor \lg(i-1) - 1 \rfloor \end{aligned}$$

for randomly chosen $r_{i,j}, r'_{i,j}, r''_{i,j}, x_{i,j} \in \mathbb{Z}_p$.

Case where $t' = 1$ (Type I Signatures). Let $\Gamma' = \min(\lfloor \lg(i-1) - 1 \rfloor, \lfloor \lg(\ell' - i + 1) \rfloor)$. When $t' = 1$, **Sign**(SK, M') is composed of the following values:

$$\begin{aligned} S''_{i,j} &= g^\alpha g^{-x'_{i+2j,j-1}} H_s(m'_{i,2j})^{v_{i,j}}, & \widetilde{S''_{i,j}} &= g^{v_{i,j}}, & \text{for } i = 1 \text{ to } \ell' \text{ and } j = 0 \text{ to } \lfloor \lg(\ell' - i + 1) \rfloor \\ A''_{i,j} &= g^{x'_{i,j}} g^{-x'_{i+2j,j-1}} H(m'_{i,2j})^{v'_{i,j}}, & \widetilde{A''_{i,j}} &= g^{v'_{i,j}}, & \text{for } i = 3 \text{ to } \ell' \text{ and } j = 0 \text{ to } \Gamma' \\ D''_{i,j} &= g^{x'_{i,j}} g^{-x'_{i,j-1}} g^{z_j v''_{i,j}}, & \widetilde{D''_{i,j}} &= g^{v''_{i,j}}, & \text{for } i = 3 \text{ to } \ell' + 1 \text{ and } j = 0 \text{ to } \lfloor \lg(i-1) - 1 \rfloor \end{aligned}$$

for randomly chosen $v_{i,j}, v'_{i,j}, v''_{i,j}, x'_{i,j} \in \mathbb{Z}_p$.

And **SignDerive**(PK, σ, M, M') is Quote-Type I(PK, σ, m, m'), which is comprised of the following:

$$\begin{aligned} S'_{i,j} &= g^\alpha g^{-w_{i+2j,j-1}} H_s(m'_{i,2j})^{r_{I,j}+t_{i,j}}, & \widetilde{S'_{i,j}} &= g^{r_{I,j}+t_{i,j}}, & \text{for } i = 1 \text{ to } \ell' \text{ and } j = 0 \text{ to } \lfloor \lg(\ell' - i + 1) \rfloor \\ A'_{i,j} &= g^{w_{i,j}} g^{-w_{i+2j,j-1}} H(m'_{i,2j})^{r'_{I,j}+t'_{i,j}}, & \widetilde{A'_{i,j}} &= g^{r'_{I,j}+t'_{i,j}}, & \text{for } i = 3 \text{ to } \ell' \text{ and } j = 0 \text{ to } \Gamma' \\ D'_{i,j} &= g^{w_{i,j}} g^{-w_{i,j-1}} g^{z_j(r''_{I,j}+t''_{i,j})}, & \widetilde{D'_{i,j}} &= g^{r''_{I,j}+t''_{i,j}}, & \text{for } i = 3 \text{ to } \ell' + 1 \text{ and } j = 0 \text{ to } \lfloor \lg(i-1) - 1 \rfloor \end{aligned}$$

for randomly chosen $t_{i,j}, t'_{i,j}, t''_{i,j}, y_{i,j} \in \mathbb{Z}_p$, where m' occurs at position k as a substring of m , $I = i + k - 1$ and $w_{i,j} = x_{I,j} + y_{i,j}$.

Since all exponents have been independently re-randomized, one can see by inspection that **SignDerive**(pk, σ, M', M) has identical distribution as that of **Sign**(sk, M').

Case where $t' = 0$ (Type II Signatures). Parse $m' = m'_\beta m'_{\beta-1} \dots m'_0$ where m'_j is of length 2^j or a null string where $\beta = \lfloor \lg(\ell') \rfloor$. ℓ'_i denotes i -th bit of ℓ' when we start counting with zero as the least significant bit. m' occurs at position k of m . **Sign**(SK, M') = $(B, \tilde{S}, Z_{\beta-1}, \dots, Z_0)$ is the following, for random $u, u_i \in \mathbb{Z}_p$:

$$\begin{aligned} B &= g^\alpha \cdot H_s(m'_\beta)^u \prod_{j < \beta, \ell'_j=1} H(m'_j)^{u_j} \prod_{j' < \beta, \ell'_{j'}=0} g^{z_{j'} u_{j'}} \\ \tilde{S} &= g^u, \quad Z_j = g^{u_j} \end{aligned}$$

Let each m'_j start at position s_j in m' . **SignDerive**(PK, σ, M, M') = Quote-Type II(PK, σ, m, m') is $(B', \tilde{S}', Z'_{\beta-1}, \dots, Z'_0)$ such that

$$\begin{aligned} B' &= g^\alpha \cdot H_s(m'_\beta)^{r_{k,\beta}+v} \prod_{j < \beta, \ell'_j=1} H(m'_j)^{r'_{k+s_j-1,j}+v_j} \prod_{j' < \beta, \ell'_{j'}=0} g^{z_{j'}(r''_{k+s_{j'}-1,j'}+v_{j'})} \\ \tilde{S}' &= g^{r_{k,\beta}+v}, \quad Z'_j = g^{r''_{k+s_j-1,j}+v_j} \end{aligned}$$

for randomly chosen $v, v_j \in \mathbb{Z}_p$. Since all exponents have been independently re-randomized, one can see by inspection that **SignDerive**(PK, σ, M, M') has identical distribution as that of **Sign**(sk, M').

Thus, the our powers-of-2 construction is strongly context-hiding. $\square$

Lemma 4.4 (Unforgeability) *If the CDH assumption holds in $\mathbb{G}$, then the Section 4 quotable signature scheme is selectively unforgeable in the **Unforg** game in the random oracle model.*

Proof. We first apply Lemma A.4, which allows us to only consider adversaries $\mathcal{A}$ that asks queries to *Sign* oracle in the simpler **NHU** game.

Suppose an adversary $\mathcal{A}$ can produce a forgery with probability ϵ in the selective **NHU** unforgeability game; then we can construct an adversary $\mathcal{B}$ that breaks the CDH assumption with probability ϵ plus a negligible amount.

We are now ready to describe $\mathcal{B}$ which solves the CDH problem. On input the CDH challenge (g, g^a, g^b) , $\mathcal{B}$ begins to run $\mathcal{A}$ and proceeds as follows:

Selective Disclosure $\mathcal{A}$ first announces the message M^* on which he will forge.

Setup Let L be the maximum size of any message and let $n = \lfloor \lg(L) \rfloor$. Let $M^* = (t^*, m^*)$ and $\ell^* = |m^*|$ and let β be the highest bit of ℓ^* set to 1 (numbering the least significant bit as zero). Set $\mathbf{e}(g, g)^\alpha := \mathbf{e}(g^a, g^b)$, which implicitly sets the secret key $\alpha = ab$.

For $i = 0$ to $n - 1$, choose a random $v_i \in \mathbb{Z}_p$ and set

$$g^{z_i} = \begin{cases} g^{bv_i} & \text{if the } i\text{th bit of } \ell^* \text{ is 1;} \\ g^{v_i} & \text{otherwise.} \end{cases}$$

Finally, $\mathcal{B}$ give the public key $PK = (g, g^{z_0}, \dots, g^{z_{n-1}}, \mathbf{e}(g, g)^\alpha)$ to $\mathcal{A}$ and will answer its queries to random oracles H and H_s interactively as described below.

Random Oracle Queries Proceeding adaptively, $\mathcal{A}$ may make any of the following queries which $\mathcal{B}$ will answer as follows:

1. $H(x)$: The random oracle is answered as follows. If the query has been made before, return the same response as before. Otherwise, imagine dividing up m^* into a sequence of segments whose lengths are decreasing powers of two; that is, the first segments would be of length 2^β where β is the largest power of two less than ℓ^* , the second segment would contain the next largest power of two, etc. Let $m_{(j)}^*$ denote the segment of m^* corresponding to power j . If no such segment exists, let $m_{(j)}^* = \perp$. Select a random $\gamma \in \mathbb{Z}_p$ and return the response as:

$$H(x) = \begin{cases} g^\gamma & \text{if } |x| = 2^j \text{ and } j < \beta \text{ and } m_{(j)}^* = x \\ & (x \text{ is on the selective path}); \\ g^{b\gamma} & \text{otherwise} \\ & (x \text{ is not on the selective path}). \end{cases}$$

Note that $H(m_{(j)}^*)$ is set according to the first method for all segments of m^* *except* the first segment $m_{(\beta)}^*$.

2. $H_s(x)$: The random oracle is answered as follows. If the query has been made before, return the same response as before. Select a random $\delta \in \mathbb{Z}_p$ and return the response as:

$$H_s(x) = \begin{cases} g^\delta & \text{if } |x| = 2^\beta \text{ and } m_{(\beta)}^* = x; \\ g^{b\delta} & \text{otherwise.} \end{cases}$$

Note that $H_s(m_{(j)}^*)$ is set according to the first method *only* for the first segment of m^* .

Signature and Quote Queries

Sign (M): Let $M = (t, m)$ and $\ell = |m|$. Recall that β^* is highest bit of ℓ^* set to 1 and that we are counting up from zero as the least significant bit.

We describe how to create signatures.

1. When $t = 1$ and m^* is not a substring of m (Type I Signature Generation):

Here $m_{i,j}$ denotes the substring m of length j starting at position i . It will help us to first establish the variables $X_{i,j}$, which will be set to 1 if on the selective forgery path and 0 otherwise. We give a set of “rules” defining terms and make a few observations. Then we describe how the reduction algorithm creates the signatures.

Rules.

For $i = 1$ up to $\ell + 1$,

For $j = \lfloor \lg(\ell - i + 1) \rfloor$ down to -1 ,

- (a) If $j + 1 = \beta^*$ and $m_{i-2^{j+1}, 2^{j+1}} = m_{(j+1)}^*$, then set $X_{i,j} = 1$.
- (b) Else, if $j + 1 < \beta^*$ and $(j + 1)$ th bit of ℓ^* is 1 and $m_{i-2^{j+1}, 2^{j+1}} = m_{(j+1)}^*$ and $X_{i-2^{j+1}, j+1} = 1$, then set $X_{i,j} = 1$.
- (c) Else if $j + 1 < \beta^*$ and $(j + 1)$ th bit of ℓ^* is 0 and $X_{i, j+1} = 1$, then set $X_{i,j} = 1$.
- (d) Else set $X_{i,j} = 0$.

Observations. Before we show how $\mathcal{B}$ will simulate the signatures, we make a set of useful observations.

- (a) For all i and $j \geq \beta^*$, $X_{i,j} = 0$.
- (b) For all i , $X_{i,-1} = 0$. Otherwise, $m_{i-\ell^*, \ell^*} = m^*$.
- (c) For all i, j , if $X_{i,j} = 1$ and $X_{i, j-1} = 0$, then the j th bit of ℓ^* is 1. If the j th bit were 0, then $X_{i, j-1}$ would have been set to 1 by Rule 1c.
- (d) For all i, j , if $X_{i,j} = 0$ and $X_{i, j-1} = 1$, then the j th bit of ℓ^* is 1. If the j th bit were 0, then the only way to set $X_{i, j-1}$ to 1 would be by Rule 1c, however, $X_{i,j} = 0$ so Rule 1c does not apply.
- (e) For all i, j , if $X_{i,j} = 1$ and $X_{i+2^j, j-1} = 0$, then $H(m_{i, 2^j}) = g^{b\gamma}$ for some known $\gamma \in \mathbb{Z}_p$. Otherwise, $X_{i+2^j, j-1}$ would have been set by Rule 1b to be 1.
- (f) For all i, j , if $X_{i,j} = 0$ and $X_{i+2^j, j-1} = 1$, then $H(m_{i, 2^j}) = g^{b\gamma}$ for some known $\gamma \in \mathbb{Z}_p$. If $X_{i+2^j, j-1} = 1$ and $X_{i,j} = 0$, then $X_{i+2^j, j-1}$ was set to be 1 either by Rule 1a or Rule 1c. If it were Rule 1a, then $j = \beta^*$ and it follows from the programming of the random oracle that $H(m_{i, 2^j}) = g^{b\gamma}$. If it were Rule 1c, then the j th bit of ℓ^* is 0, meaning $m_{(j)}$ cannot be on the selective path and therefore again $H(m_{i, 2^j}) = g^{b\gamma}$.
- (g) For all i, j , if $X_{i+2^j, j-1} = 0$, then $H_s(m_{i, 2^j}) = g^{b\delta}$ for some known $\delta \in \mathbb{Z}_p$. If $j \neq \beta^*$, this follows immediately from the programming of the random oracle. Otherwise, if $j = \beta^*$, then the only way for $X_{i+2^j, j-1} = 0$ would be if $m_{(\beta)} \neq m_{(\beta)}^*$ by Rule 1a. Thus, it also follows that $H_s(m_{i, 2^j}) = g^{b\delta}$.

Signature Components. Next, for $i = 1$ to $\ell + 1$ and $j = 0$ to $\lfloor \lg(\ell - i + 1) \rfloor$, choose a random $x'_{i,j} \in \mathbb{Z}_p$ and logically set $x_{i,j} := x'_{i,j} + X_{i,j} \cdot (ab)$. For $i = 1$ to $\ell + 1$, set $x_{i,-1} := 0$ (as consistent with Observation 1b.)

A signature is comprised of the following values:

Start. For $i = 1$ to ℓ and $j = 0$ to $\lfloor \lg(\ell - i + 1) \rfloor$:

- (a) If $X_{i+2j,j-1} = 0$, then it follows by Observation 1g that $H_s(m_{i,2j}) = g^{b\delta}$ for some known $\delta \in \mathbb{Z}_p$, so choose random $s_{i,j} \in \mathbb{Z}_p$, implicitly set $r_{i,j} := -a/\delta + s_{i,j}$ and set

$$\begin{aligned} S_{i,j} &= g^{-x_{i+2j,j-1}} g^{b\delta s_{i,j}} \\ &= g^\alpha g^{-x_{i+2j,j-1}} H_s(m_{i,2j})^{r_{i,j}} \\ \widetilde{S_{i,j}} &= g^{-a/\delta + s_{i,j}} = g^{r_{i,j}} \end{aligned}$$

- (b) Else $X_{i+2j,j-1} = 1$, so choose random $r_{i,j} \in \mathbb{Z}_p$ and with $x_{i+2j,j-1} := x'_{i+2j,j-1} + ab$ set

$$\begin{aligned} S_{i,j} &= g^{-x'_{i+2j,j-1}} H_s(m_{i,2j})^{r_{i,j}} \\ &= g^\alpha g^{-x_{i+2j,j-1}} H_s(m_{i,2j})^{r_{i,j}} \\ \widetilde{S_{i,j}} &= g^{r_{i,j}} \end{aligned}$$

Across. Together with the following values for $i = 3$ to ℓ and $j = 0$ to $\min(\lfloor \lg(i-1) - 1 \rfloor, \lfloor \lg(\ell - i + 1) \rfloor)$:

- (a) If $X_{i,j} = 1$ and $X_{i+2j,j-1} = 1$, choose random $r'_{i,j} \in \mathbb{Z}_p$ with implicitly set $x_{i,j} = x'_{i,j} + ab$ and $x_{i+2j,j-1} = x'_{i+2j,j-1} + ab$ and set

$$\begin{aligned} A_{i,j} &= g^{x'_{i,j}} g^{-x'_{i+2j,j-1}} H(m_{i,2j})^{r'_{i,j}} \\ &= g^{x_{i,j}} g^{-x_{i+2j,j-1}} H(m_{i,2j})^{r'_{i,j}} \\ \widetilde{A_{i,j}} &= g^{r'_{i,j}} \end{aligned}$$

- (b) Else, if $X_{i,j} = 1$ and $X_{i+2j,j-1} = 0$, then $H(m_{i,2j}) = g^{b\gamma}$ for some known $\gamma \in \mathbb{Z}_p$ by Observation 1e. Choose random $s'_{i,j} \in \mathbb{Z}_p$ with implicitly set $x_{i,j} = x'_{i,j} + ab$, $x_{i+2j,j-1} = x'_{i+2j,j-1}$ and $r'_{i,j} := -a/\gamma + s'_{i,j}$ and set

$$\begin{aligned} A_{i,j} &= g^{x'_{i,j}} g^{-x_{i+2j,j-1}} g^{b\gamma s'_{i,j}} \\ &= g^{x_{i,j}} g^{-x_{i+2j,j-1}} H(m_{i,2j})^{r'_{i,j}} \\ \widetilde{A_{i,j}} &= g^{r'_{i,j}} \end{aligned}$$

- (c) Else, if $X_{i,j} = 0$ and $X_{i+2j,j-1} = 1$, then $H(m_{i,2j}) = g^{b\gamma}$ for some known $\gamma \in \mathbb{Z}_p$ by Observation 1f. Choose random $s'_{i,j} \in \mathbb{Z}_p$ with implicitly set $x_{i,j} = x'_{i,j}$, $x_{i+2j,j-1} = x'_{i+2j,j-1} + ab$ and $r'_{i,j} := a/\gamma + s'_{i,j}$ and set

$$\begin{aligned} A_{i,j} &= g^{x_{i,j}} g^{-x'_{i+2j,j-1}} g^{b\gamma s'_{i,j}} \\ &= g^{x_{i,j}} g^{-x_{i+2j,j-1}} H(m_{i,2j})^{r'_{i,j}} \\ \widetilde{A_{i,j}} &= g^{r'_{i,j}} \end{aligned}$$

- (d) Else, $X_{i,j} = 0$ and $X_{i+2j,j-1} = 0$, so choose random $r'_{i,j} \in \mathbb{Z}_p$ and set

$$A_{i,j} = g^{x_{i,j}} g^{-x_{i+2j,j-1}} H(m_{i,2j})^{r'_{i,j}}, \quad \widetilde{A_{i,j}} = g^{r'_{i,j}}$$

Down. Together with the following values for $i = 3$ to $\ell + 1$ and $j = 0$ to $\lfloor \lg(i-1) - 1 \rfloor$:

- (a) If $X_{i,j} = 1$ and $X_{i,j-1} = 1$, choose random $r''_{i,j} \in \mathbb{Z}_p$ with implicitly set $x_{i,j} = x'_{i,j} + ab$ and $x_{i,j-1} = x'_{i,j-1} + ab$ and set

$$\begin{aligned} D_{i,j} &= g^{x'_{i,j}} g^{-x'_{i,j-1}} g^{z_j r''_{i,j}} = g^{x_{i,j}} g^{-x_{i,j-1}} g^{z_j r''_{i,j}} \\ \widetilde{D_{i,j}} &= g^{r''_{i,j}} \end{aligned}$$

- (b) Else, if $X_{i,j} = 1$ and $X_{i,j-1} = 0$, then the j th bit of ℓ^* is 1 by Observation 1c. Thus $z_j = bv_j$, so choose random $s''_{i,j} \in \mathbb{Z}_p$ with implicitly set $x_{i,j} = x'_{i,j} + ab$, $x_{i,j-1} = x'_{i,j-1}$ and $r''_{i,j} := -a/v_j + s''_{i,j}$ and set

$$\begin{aligned} D_{i,j} &= g^{x'_{i,j}} g^{-x_{i,j-1}} g^{bv_j s''_{i,j}} = g^{x_{i,j}} g^{-x_{i,j-1}} g^{z_j r''_{i,j}} \\ \widetilde{D_{i,j}} &= g^{-a/v_j + s''_{i,j}} = g^{r''_{i,j}} \end{aligned}$$

- (c) Else, if $X_{i,j} = 0$ and $X_{i,j-1} = 1$, then the j th bit of ℓ^* is 1 by Observation 1d. Thus $z_j = bv_j$, so choose random $s''_{i,j} \in \mathbb{Z}_p$ with implicitly set $x_{i,j} = x'_{i,j}$, $x_{i,j-1} = x'_{i,j-1} + ab$ and $r''_{i,j} := a/v_j + s''_{i,j}$ and set

$$\begin{aligned} D_{i,j} &= g^{x'_{i,j}} g^{-x_{i,j-1}} g^{bv_j s''_{i,j}} = g^{x_{i,j}} g^{-x_{i,j-1}} g^{z_j r''_{i,j}} \\ \widetilde{D_{i,j}} &= g^{a/v_j + s''_{i,j}} = g^{r''_{i,j}} \end{aligned}$$

- (d) Else, $X_{i,j} = 0$ and $X_{i,j-1} = 0$, so choose random $r''_{i,j} \in \mathbb{Z}_p$ and set

$$D_{i,j} = g^{x_{i,j}} g^{-x_{i,j-1}} g^{z_j r''_{i,j}}, \quad \widetilde{D_{i,j}} = g^{r''_{i,j}}$$

2. When $t = 0$ and $m \neq m^*$ (Type II Signature Generation):

Let $\ell = |m|$, and $\beta = \lfloor \lg(\ell) \rfloor$. ℓ_i^* denotes i -th bit of ℓ^* when we start counting with zero as the least significant bit, and ℓ_i denotes i -th bit of ℓ .

Parse m^* as $m_{\beta^*}^* m_{\beta^*-1}^* \dots m_0^*$ where m_i^* is a string of length 2^i or a null string. m_i is of length 2^i if $\ell_i = 0$, and is null otherwise. Similarly, parse m as $m_{\beta} m_{\beta-1} \dots m_0$.

$\mathcal{B}$ constructs $(B, \tilde{S}, Z_{\beta-1}, \dots, Z_0)$ in the following way:

- If $m_{\beta} \neq m_{\beta^*}^*$, then $H_s(m_{\beta}) = g^{b\delta}$ for a δ which is known to $\mathcal{B}$.
 - (a) $\mathcal{B}$ sets $\tilde{S} := g^{-a/\delta+r}$ for a randomly chosen r and $B := g^{b\delta r}$.
 - (b) For $j = \beta - 1$ down to 0, $Z_j := g^{r_j}$ for a randomly chosen r_j , and
 - If $\ell_j = 1$, then $B := B \cdot H(m_j)^{r_j}$.
 - If $\ell_j = 0$, then $B := B \cdot g^{z_j r_j}$.
- Otherwise, if $\beta = \beta^*$ and $m_{\beta} = m_{\beta^*}^*$, there exists $j_s < \beta$ such that
 - $\ell_{j_s} \neq \ell_{j_s}^*$, or
 - $\ell_{j_s} = \ell_{j_s}^* = 1$ and $H(m_{j_s}) \neq H(m_{j_s}^*)$.

so $\mathcal{B}$ can construct a signature $(B, \tilde{S}, Z_{\beta-1}, \dots, Z_0)$ in the following way.

- (a) $\mathcal{B}$ sets $\tilde{S} := g^{r_c}$ for a randomly chosen r_c and $B := g^{\delta r_c}$.
- (b) For $j = \beta - 1$ down to $j_s + 1$ and $j = j_s - 1$ to 0, $Z_j := g^{r_j}$ for randomly chosen r_j , and
 - If $\ell_j = 1$, then $B := B \cdot H(m_j)^{r_j}$.
 - If $\ell_j = 0$, then $B := B \cdot g^{z_j r_j}$.
- (c) For $j = j_s$,

- If $\ell_j = 1$, whether $\ell_j^* = 0$ or not, $\mathcal{B}$ knows γ such that $H(m_j) = g^{b\gamma}$. $\mathcal{B}$ sets $Z_j = g^{-a/\gamma+r_j}$ for a randomly chosen r_j , and $B := B \cdot g^{b\gamma r_j}$.
- If $\ell_j = 0$ and $\ell_j^* = 1$, then $\mathcal{B}$ knows v such that $g^{z_j} = g^{bv}$. $\mathcal{B}$ sets $Z_j = g^{-a/v+r_j}$ for a randomly chosen r_j , and $B := B \cdot g^{bv r_j}$.

$\mathcal{B}$ returns $(B, \tilde{S}, Z_{\beta-1}, \dots, Z_0)$.

Response Eventually, $\mathcal{A}$ outputs a valid signature σ^* on $M^* = (t^*, m^*)$. Recall that $\ell^* = |m^*|$ and $\beta = \lfloor \lg(\ell^*) \rfloor$. Here ℓ_i^* denotes i -th bit of ℓ^* when we start counting with zero as the least significant bit. Parse m^* as $m_\beta^* m_{\beta-1}^* \dots m_0^*$ where m_i^* is a string of length 2^i (when $\ell_i^* = 1$) or a null string (when $\ell_i^* = 0$).

Because of the selective disclosure and setup, $\mathcal{B}$ knows the following exponents:

- γ such that $H_s(m_\beta^*) = g^\gamma$.
- δ_j such that $H(m_{s_j, 2^j}^*) = g^{\delta_j}$ when $\ell_j^* = 1$ and $j \neq \beta$.
- z_j when $\ell_j^* = 0$.

t^* is either 1 or 0.

- If $t^* = 1$,
 s_i denotes the position where m_i^* starts. $\mathcal{B}$ can compute the information of some $x_{i,j}$ with the following components of σ^* .

$$- S_{1,\beta} = g^\alpha g^{-x_{1+2\beta, \beta-1}} H_s(m_\beta^*)^{r_c}, \quad \widetilde{S_{1,\beta}} = g^{r_{1,\beta}}$$

$\mathcal{B}$ knows γ such that $H_s(m_\beta^*) = g^\gamma$, so $\mathcal{B}$ can compute $g^\alpha g^{-x_{1+2\beta, \beta-1}} = S_{1,\beta} / \widetilde{S_{1,\beta}}^\gamma$.

- For $j = \beta - 1$ down to 0,

$$* \text{ when } \ell_j = 1, \quad A_{s_j, j} = g^{x_{s_j, j}} g^{-x_{s_{j-1}, j-1}} H(m_j^*)^{r'_{s_j, j}}, \quad \widetilde{A_{s_j, j}} = g^{r'_{s_j, j}}$$

$\mathcal{B}$ knows δ such that $H(m_j^*) = g^\delta$, so $\mathcal{B}$ can compute $g^{x_{s_j, j}} g^{-x_{s_{j-1}, j-1}} = A_{s_j, j} / \widetilde{A_{s_j, j}}^\delta$.

$$* \text{ when } \ell_j = 0, \quad D_{s_j, j} = g^{x_{s_j, j}} g^{-x_{s_{j-1}, j-1}} g^{z_j r''_{s_j, j}}, \quad \widetilde{D_{s_j, j}} = g^{r''_{s_j, j}}$$

$\mathcal{B}$ knows z_j , so $\mathcal{B}$ can compute $g^{x_{s_j, j}} g^{-x_{s_{j-1}, j-1}} = D_{s_j, j} / \widetilde{D_{s_j, j}}^{z_j}$.

so $\mathcal{B}$ can compute $g^{x_{s_j, j}} g^{-x_{s_{j-1}, j-1}}$.

$\mathcal{B}$ has the values of $g^{x_{s_j, j}} g^{-x_{s_{j-1}, j-1}}$ for $j = \beta - 1$ down to 0 and $g^\alpha g^{-x_{1+2\beta, \beta-1}}$, so can compute

$$g^\alpha g^{-x_{1+2\beta, \beta-1}} \prod_{j=0}^{\beta-1} g^{x_{s_j, j}} g^{-x_{s_{j-1}, j-1}} = g^\alpha g^{-x_{s_{-1}, -1}} = g^\alpha$$

- If $t^* = 0$,
 $\mathcal{B}$ parses σ^* as $(B, \tilde{S}, Z_{\beta-1}, \dots, Z_0)$, with

$$\tilde{S} = g^c, \quad Z_{\beta-1} = g^{c_{\beta-1}}, \quad \dots, \quad Z_0 = g^{c_0}$$

for some $c, c_{\beta-1}, \dots, c_0 \in \mathbb{Z}_p$.

$$B = g^\alpha \cdot H_s(m_\beta^*)^c \prod_{j < \beta, \ell_j^* = 1} H(m_j^*)^{c_j} \prod_{j' < \beta, \ell_{j'}^* = 0} (g^{z_{j'}})^{c_{j'}}$$

because the signature is valid.

- $\mathcal{B}$ knows γ such that $H_s(m_\beta^*) = g^\gamma$. $\mathcal{B}$ sets $C := \tilde{S}^\gamma$.

- From $j = \beta - 1$ down to 0, $\mathcal{B}$ proceeds as:
 - * If $\ell_j = 1$, $\mathcal{B}$ knows δ_j such that $H(m_j^*) = g^{\delta_j}$. $\mathcal{B}$ sets $C := C \cdot Z_j^{\delta_j}$;
 - * If $\ell_j = 0$, $\mathcal{B}$ knows z_j . $\mathcal{B}$ sets $C := C \cdot Z_j^{z_j}$.

Then

$$C = H_s(m_\beta^*)^c \prod_{j < \beta, \ell_j^* = 1} H(m_j^*)^{c_j} \prod_{j' < \beta, \ell_{j'}^* = 0} (g^{z_{j'}})^{c_{j'}}$$

so $\mathcal{B}$ can compute $B/C = g^\alpha$.

Thus, whether t^* is 1 or 0, $\mathcal{B}$ can solve for $g^\alpha = g^{ab}$ and correctly answer to the CDH challenge.

Analysis The distribution of the above game and the security game are identical. Thus, whenever $\mathcal{A}$ is successful in a forgery against our scheme, $\mathcal{B}$ will solve the CDH challenge. $\square$

5 A Construction for Subset Predicates based on ABE

The Subset Predicate. We now point out a surprising connection to Attribute Based Encryption (ABE). We show that existing constructions for Ciphertext-Policy ABE [8, 37, 58] naturally lead to context hiding quotable signatures for arbitrary message *subsets* (as opposed to the *substring* predicate considered in the previous section). In particular, let U be a set of strings over an arbitrary alphabet. These strings can be used to encode elements for different types of sets. A message will be a set of strings from U . A general way to define the subset predicate would be $P(M, m') = 1$ iff $m' \subseteq m_i$ for some $m_i \in M$. Recall from Section 2 that M is a set of messages, which might have been independently authenticated. Here, we want to disallow “collusions” between two different signatures where m' is a subset of the union of multiple messages in M , but not any single one. (Otherwise, this would be trivially realizable from standard signatures schemes.) In other words, our focus here is extracting a subset from a *single* signed set. Thus, we will restrict our attention in this section to the simple predicate $P(m, m') = 1$ iff $m' \subseteq m$.

The Construction at a High-Level. Our main tool is an observation of Naor that shows that secret keys in Identity Based Encryption [12] can function as signatures. Recall that in (ciphertext-policy) *attribute based encryption* an authority provides secret keys to a user based on the user’s list of attributes. The main challenge in building such systems is preventing collusion attacks: two (or more) users with distinct sets of attributes should be unable to create a secret key for a combination of their attributes.

If we treat elements in a message $m \subseteq U$ as attributes, that is, we treat a message $m = \{a_1, \dots, a_\ell\} \in U^\ell$ as a set of attributes $a_1, \dots, a_\ell$, then we can define the signature on m as a set of ℓ secret keys corresponding to the ℓ attributes in the message. Verifying the signature can be done by trying to decrypt some test ciphertext using the secret keys in the signature. Now, given a signature on m we derive a signature on a subset of the elements in m by simply removing the secret keys corresponding to elements not in the subset. For context hiding, we need to re-randomize the resulting set of secret keys. (Not all CP-ABE schemes may support the removal and re-randomization of secret keys in this manner, but the schemes of [8, 37, 58] do.)

Since ABE security prevents collusion attacks, it is straight forward to show that these signatures are unforgeable in the sense of Definition 2.3. Moreover, due to the re-randomization of secret keys,

a derived signature is sampled from the same distribution as a fresh signature and is independent from the given signature. This implies strong context hiding in sense of Definition 2.4.

This unexpected connection between quoting and ABE leads to the following theorem, stated first informally.

Theorem 5.1 (informal) *The Ciphertext-Policy ABE systems in [8, 37, 58] translate using Naor’s transformation into a signature scheme supporting quoting for arbitrary subsets of a message. (Selective) security of the CP-ABE systems imply (selective) unforgeability and context hiding.*

In other words, when the ABE scheme provides adaptive (resp, selective) security, then the resulting signature scheme achieves adaptive (resp., selective) unforgeability. The (third) ABE scheme of Waters [58] provides selective security from the Decisional Bilinear Diffie Hellman assumption. Adaptive security is proven for the Bethencourt et al. construction [8], but only in the generic group model. The construction of Lewko et al. [37] proves adaptive security under certain static assumptions using composite order groups.

5.1 The Subset Construction from Existing CP-ABE Schemes

We now formalize the intuition and claims of the previous section.

5.1.1 Background: Ciphertext-Policy ABE

Definition 1 (Access Structure [3]) *Let $\{P_1, P_2, \dots, P_n\}$ be a set of parties. A collection $\Gamma \subseteq 2^{\{P_1, P_2, \dots, P_n\}}$ is monotone if $\forall B, C : \text{if } B \in \Gamma \text{ and } B \subseteq C \text{ then } C \in \Gamma$. An access structure (respectively, monotone access structure) is a collection (resp., monotone collection) Γ of non-empty subsets of $\{P_1, P_2, \dots, P_n\}$, i.e., $\Gamma \subseteq 2^{\{P_1, P_2, \dots, P_n\}} \setminus \{\emptyset\}$. The sets in Γ are called the authorized sets, and the sets not in Γ are called the unauthorized sets.*

In the context of CP-ABE, the role of the parties is taken by the attributes. Thus, the access structure Γ will contain the authorized sets of attributes. We restrict our attention to monotone access structures.

Definition 5.2 (CP-ABE Algorithm Specification) *A ciphertext-policy attribute-based encryption system for message space $\mathcal{M}$ and access structure space $\mathcal{G}$ is a tuple of the following algorithms:*

Setup $(\lambda, U) \rightarrow (\text{PK}, \text{MK})$. *The setup algorithm takes as input a security parameter λ and a universe description U , which defines the set of allowed attributes in the system. It outputs the public parameters PK and the master secret key MK.*

Encrypt $(\text{PK}, m, \Gamma) \rightarrow \text{CT}$. *The encryption algorithm takes as input the public parameters PK, a message m and an access structure Γ and outputs a ciphertext CT associated with the access structure.*

KeyGen $(\text{MK}, S) \rightarrow sk$. *The key generation algorithm takes as input the master secret key MK and a set of attributes S and outputs a private key sk associated with the attributes.*

Decrypt $(sk, \text{CT}) \rightarrow m$. *The decryption algorithm takes as input a secret key sk associated with attributes S and a ciphertext CT associated with access structure Γ and outputs a message m if S satisfies Γ or the error message $\perp$ otherwise.*

*The correctness property requires that for all sufficiently large $\lambda \in \mathbb{N}$, all universe descriptions U , all $(\text{PK}, \text{MK}) \in \text{Setup}(\lambda, U)$, all $S \subseteq U$, all $sk \in \text{KeyGen}(\text{MK}, S)$, all $m \in \mathcal{M}$, all $\Gamma \in \mathcal{G}$ and all $\text{CT} \in \text{Encrypt}(\text{PK}, m, \Gamma)$, if S satisfies Γ , then **Decrypt** (sk, CT) outputs m .*

Security Model for CP-ABE Let $\Pi = (\text{Setup}, \text{Encrypt}, \text{KeyGen}, \text{Decrypt})$ be a CP-ABE scheme for message space $\mathcal{M}$ and access structure space $\mathcal{G}$, and consider the following experiment for an adversary Adv , parameter λ and attribute universe U :

The CP-ABE experiment $\text{CP-ABE-Exp}_{\text{Adv}, \Pi}(\lambda, U)$:

Start. $\text{Setup}(\lambda, U)$ is run to obtain the public parameters PK and master secret key MK.

Phase 1. Adversary Adv is given PK and access to the oracle $\text{KeyGen}(\text{MK}, \cdot)$, which generates a private key corresponding to an attribute set of the adversary's choosing.

Challenge. The adversary outputs two messages $m_0, m_1 \in \mathcal{M}$ and a challenge access structure Γ^* such that none of the sets of attributes queried during Phase 1 satisfy it. A random bit b is chosen and $\text{Encrypt}(\text{PK}, m_b, \Gamma^*)$ is run to produce CT^* , which is then given to the adversary.

Phase 2. The adversary is given access to the oracle $\text{KeyGen}(\text{MK}, \cdot)$, with the restriction that it cannot query the oracle on any set of attributes that satisfy Γ^* .

Guess. The adversary outputs a guess b' of b . The output of the experiment is defined to be 1 if and only if $b' = b$.

Definition 5.3 (CP-ABE Security) A CP-ABE scheme Π is secure for attribute universe U if for all probabilistic polynomial-time adversaries Adv , there exists a negligible function negl such that:

$$\Pr[\text{CP-ABE-Exp}_{\text{Adv}, \Pi}(\lambda, U) = 1] \leq \text{negl}(\lambda).$$

We say that a system is *selectively* secure if we add an Init stage before Start where the adversary outputs the challenge access structure Γ^* (instead of waiting until Challenge to do so).

5.1.2 CP-ABE with Key Reduction

Our construction requires that the holder of a private key can efficiently “remove” attributes from his private key and then re-randomize the remaining private key. We formalize this as follows.

Definition 5.4 (CP-ABE with key reduction) We say that a CP-ABE system for attribute universe U supports key reduction if there exists an efficient algorithm

$\text{KeyReduce}(\text{PK}, sk, S, S') \rightarrow sk'$. The key reduction algorithm takes as input the public parameters PK with a private key sk associated with attribute set S and outputs a private key sk' associated with attribute set S' , if $S' \subseteq S$, and $\perp$ otherwise.

such that if $(\text{PK}, \text{MK}) \in \text{Setup}(\lambda, U)$ and $S' \subseteq S \subseteq U$, then for all such tuples (MK, S, S') , the following two distributions are statistically close:

$$\begin{aligned} & \{(\text{MK}, sk \leftarrow \text{KeyGen}(\text{MK}, S), \text{KeyGen}(\text{MK}, S'))\}_{\text{MK}, S, S'} \\ & \{(\text{MK}, sk \leftarrow \text{KeyGen}(\text{MK}, S), \text{KeyReduce}(\text{PK}, sk, S, S'))\}_{\text{MK}, S, S'} \end{aligned}$$

The distributions are taken over the coins of KeyGen and KeyReduce .

It is not a coincidence that this definition strongly resembles the context hiding definition presented earlier. Fortunately, we observed that several existing CP-ABE schemes support key reduction.

Claim 5.5 *The Ciphertext-Policy ABE systems in [8, Section 4.2],[37, Section 2.3.1],[58, Section 6] support key reduction.*

Proof. We argue this claim by providing a key reduction algorithm for each scheme. In all cases, the output is perfectly indistinguishable from the normal key generation algorithm.

The BSW Construction [8, Section 4.2]

- **Setup**(λ, U) $\rightarrow$ (PK, MK): The algebraic setting is a bilinear group $\mathbb{G}$ of prime order p with generator g . The public parameters PK are $\mathbb{G}, g, p, h = g^\beta, f = g^{a/\beta}, e(g, g)^\alpha$, where $\beta, \alpha \in \mathbb{Z}_p$, and the description of a hash function $H : \{0, 1\}^* \rightarrow \mathbb{G}$. The master secret key MK is $(\text{PK}, \beta, g^\alpha)$.
- **KeyGen**(MK, S) $\rightarrow sk$: The key generation algorithm chooses random $r, r_i \in \mathbb{Z}_p$ for each attribute $i \in S$. The private key sk is:

$$S, D = g^{(\alpha+r)/\beta}, D_j = g^r H(j)^{r_j}, D'_j = g^{r_j} \forall j \in S.$$

- **KeyReduce**(PK, sk, S, S') $\rightarrow sk'$: The key reduction algorithm chooses random r', r'_i for each attribute $i \in S'$ and outputs the private key sk' as:

$$S', D' = Dg^{r'/\beta} = g^{(\alpha+r+r')/\beta}, \\ D'_j = D_j g^{r'_j} H(j)^{r_j} = g^{r+r'} H(j)^{r_j+r'_j}, D'_j = D_j g^{r'_j} = g^{r_j+r'_j} \forall j \in S'.$$

The LOSTW Construction [37, Section 2.3.1]

- **Setup**(λ, U) $\rightarrow$ (PK, MK): The algebraic setting is a bilinear group $\mathbb{G}$ of order $N = p_1 p_2 p_3$ (3 distinct primes). We let $\mathbb{G}_{p_i}$ denote the subgroup of order p_i in $\mathbb{G}$. The public parameters PK are $N, g, g^a, e(g, g)^\alpha, T_i = g^{s_i}$ for all attributes $i \in U$, where $g \in \mathbb{G}_{p_1}$ and $a, \alpha, s_i \in \mathbb{Z}_N$. The master secret key MK is PK, α and a generator $X_3 \in \mathbb{G}_{p_3}$.
- **KeyGen**(MK, S) $\rightarrow sk$: The key generation algorithm chooses a random $t \in \mathbb{Z}_N$ and random elements $R_0, R'_0, R_i \in \mathbb{G}_{p_3}$. The private key sk is:

$$S, K = g^\alpha g^{at} R_0, L = g^t R'_0, K_i = T_i^t R_i \forall i \in S.$$

- **KeyReduce**(PK, sk, S, S') $\rightarrow sk'$: The key reduction algorithm chooses a random $t' \in \mathbb{Z}_N$ and random elements $Z_0, Z'_0, Z_i \in \mathbb{G}_{p_3}$ and outputs the new private key sk' as:

$$S', K' = K g^{at'} Z_0 = g^\alpha g^{a(t+t')} R_0 Z_0, L' = L g^{t'} Z'_0 = g^{t+t'} R'_0 Z'_0, \\ K'_i = K_i T_i^{t'} Z'_i = T_i^{t+t'} R_i Z_i \forall i \in S'.$$

The Waters Construction [58, Section 6]

- **Setup**(λ, U) $\rightarrow$ (PK, MK): The algebraic setting is a bilinear group $\mathbb{G}$ of prime order p with generator g . Let n_{max} be the maximum number of nodes in an access formula and let $|U|$ be the number of attributes in U . The public parameters PK are $\mathbb{G}, g, p, g^a, e(g, g)^\alpha, (h_{1,1}, \dots, h_{1,U}), \dots, (h_{n_{max},1}, \dots, h_{n_{max},U})$, where all $h_{i,j}$ values are elements in $\mathbb{G}$. The master secret key MK is (PK, g^α) .

- **KeyGen**(MK, S) $\rightarrow sk$: The key generation algorithm chooses random $t_1, \dots, t_{n_{max}} \in \mathbb{Z}_p$. The private key sk is:

$$S, K = g^\alpha g^{at_1}, L_1 = g^{t_1}, \dots, L_{n_{max}} = g^{t_{n_{max}}}, \\ \forall x \in S, K_x = \prod_{j=1}^{n_{max}} h_{j,x}^{t_j}.$$

- **KeyReduce**(PK, sk, S, S') $\rightarrow sk'$: The key reduction algorithm chooses random $t'_1, \dots, t'_{n_{max}} \in \mathbb{Z}_p$. The private key sk' is:

$$S', K' = K g^{at'_1} = g^\alpha g^{a(t_1+t'_1)}, L'_1 = L_1 g^{t'_1} = g^{t_1+t'_1}, \dots, L'_{n_{max}} = L_{n_{max}} g^{t'_{n_{max}}} = g^{t_{n_{max}}+t'_{n_{max}}}, \\ \forall x \in S', K'_x = K_x \prod_{j=1}^{n_{max}} h_{j,x}^{t'_j} = \prod_{j=1}^{n_{max}} h_{j,x}^{t_j+t'_j}.$$

□

5.1.3 The Subset Signature Construction

Let $\Pi = (\mathbf{Setup}_{ABE}, \mathbf{Encrypt}_{ABE}, \mathbf{KeyGen}_{ABE}, \mathbf{Decrypt}_{ABE})$ be a CP-ABE scheme that supports key reduction with the algorithm **KeyReduce**_{ABE}. Let Π have an arbitrary, finite and efficiently-samplable¹⁰ message space $\mathcal{M}$ and access structure space $\mathcal{G}$ that supports AND gates. Let U be a set of strings over an arbitrary alphabet. We construct a signature scheme that supports computing on authenticated subsets of U as follows.

KeyGen(1^λ) : Run **Setup**_{ABE}($1^\lambda, U$) to obtain the key pair (pk, sk) , which will serve as the public and secret keys of the signature scheme.

Sign($sk, m \subseteq U$) : Run **KeyGen**_{ABE}(sk, m) to obtain an ABE private key which will be treated as the signature σ .

SignDerive(pk, σ, m, m') : First, check if $P(m, m') = 1$. If not, output $\perp$. Otherwise, run **KeyReduce**_{ABE}(pk, σ, m, m') to obtain the new signature σ' and output it.

Verify(pk, m, σ) : Choose a random value $x \in \mathcal{M}$ and a random access structure $\Gamma \in \mathcal{G}$. Run **Encrypt**_{ABE}(pk, x, Γ) to obtain CT. Output 1 if and only if **Decrypt**_{ABE}(σ, CT) = x .

5.1.4 Security Analysis

Theorem 5.6 *If Π is (resp., selectively) secure for attribute universe U with respect to Definition 5.3, then the above subset signature scheme is (resp., selectively) unforgeable with respect to Definition 2.3 and strongly context hiding with respect to Definition 2.4.*

Proof. We argue this theorem in two parts.

Lemma 5.7 (Strong Context Hiding) *If Π is a CP-ABE scheme that supports key reduction, then the above subset signature scheme is strongly context hiding under Definition 2.4.*

¹⁰We mean that it is possible to efficiently sample elements from the set uniformly at random.

Proof. This follows directly from the key reduction property of the CP-ABE scheme. Let $(pk, sk) \leftarrow \mathbf{KeyGen}(1^\lambda)$ be a key pair. A signature scheme $(\mathbf{KeyGen}, \mathbf{SignDerive}, \mathbf{Verify})$ is strongly context hiding for the simple subset predicate P if for all such triples $((pk, sk), m, m')$ where $P(m, m') = 1$, the following two distributions are statistically close:

$$\begin{aligned} & \{(sk, \sigma \leftarrow \mathbf{Sign}(sk, m), \mathbf{Sign}(sk, m'))\}_{sk, m, m'} \\ & \{(sk, \sigma \leftarrow \mathbf{Sign}(sk, m), \mathbf{SignDerive}(pk, \sigma, m, m'))\}_{sk, m, m'} \end{aligned}$$

where the distributions are taken over the random coins of $\mathbf{Sign}$ and $\mathbf{SignDerive}$. If we substitute the signature algorithms for their underlying CP-ABE algorithms, we have the following two distributions:

$$\begin{aligned} & \{(sk, \sigma \leftarrow \mathbf{KeyGen}_{ABE}(sk, m), \mathbf{KeyGen}_{ABE}(sk, m'))\}_{sk, m, m'} \\ & \{(sk, \sigma \leftarrow \mathbf{KeyGen}_{ABE}(sk, m), \mathbf{KeyReduce}_{ABE}(pk, \sigma, m, m'))\}_{sk, m, m'} \end{aligned}$$

where the distributions are taken over the random coins of $\mathbf{KeyGen}_{ABE}$ and $\mathbf{KeyReduce}_{ABE}$. The statistical closeness of these distributions is directly guaranteed by the key reduction specification when the predicate is satisfied, i.e., $m' \subseteq m$. $\square$

Lemma 5.8 (Unforgeability) *If Π is a (resp., selectively) secure CP-ABE scheme that supports key reduction, then the above subset signature scheme is (resp., selectively) unforgeable in the **Unforg** game.*

Proof. We first apply Lemma A.4, which allows us to only consider adversaries $\mathcal{A}$ that asks queries to *Sign* oracle in the simpler **NHU** game.

We deal with the non-selective case first. Suppose an adversary $\mathcal{A}$ can produce a forgery with probability ϵ in the **NHU** selective unforgeability game; then we can construct an adversary $\mathcal{B}$ that breaks the selective security of the CP-ABE scheme with key reduction with probability $1/2 + \epsilon/2$. $\mathcal{B}$ begins to run $\mathcal{A}$ and proceeds as follows:

Setup When $\mathcal{B}$ receives the public parameters pk from its challenger, it passes these on as the signature public key to $\mathcal{A}$.

Sign When $\mathcal{A}$ queries $\mathbf{Sign}(m)$, $\mathcal{B}$ queries its key generation oracle on m and returns the response to $\mathcal{A}$.

Response If $\mathcal{A}$ does not output a valid forgery, then $\mathcal{B}$ simply chooses and outputs a random bit. If $\mathcal{A}$ outputs a valid message-signature pair (m^*, σ^*) where m^* is not a subset of any message queried to $\mathbf{Sign}$, then $\mathcal{B}$ then picks two arbitrary messages m_0, m_1 in the message space of the CP-ABE scheme. It outputs these to its challenger together with a challenge access structure, which is the AND of all attributes in m^* . (Recall that in this signature scheme messages are sets of strings.) This challenge access structure is allowed, under the CP-ABE security experiment, because none of the other private keys created by the oracle can satisfy it. Once the challenge ciphertext CT^* is returned, $\mathcal{B}$ simply uses the private key σ^* to decrypt CT^* and to then state which of the two challenge messages was encrypted.

The Selective Case Suppose an adversary $\mathcal{A}$ can produce a forgery with probability ϵ in the **NHU** *selective* unforgeability game; then we can construct an adversary $\mathcal{B}$ that breaks the *selective* security of the CP-ABE scheme with key reduction with probability ϵ plus a negligible amount. $\mathcal{B}$ behaves as above, except that prior to **Setup** there is a selective disclosure phase where $\mathcal{A}$ first announces the message m^* on which he will forge. $\mathcal{B}$ then announces to its challenger that its challenge access structure will be the AND of all attributes in m^* . This information is latter used, as before, in $\mathcal{B}$'s Response phase, where now $\mathcal{A}$ will only output σ^* .

Analysis The following analysis applies to both the selective and non-selective cases. The view of $\mathcal{A}$ when interacting with $\mathcal{B}$ is identical to its view when interacting with a real **NHU** game challenger. Whenever $\mathcal{A}$ correctly produces a forgery, then $\mathcal{B}$ correctly identified the challenge message. Whenever $\mathcal{A}$ fails to produce a forgery, then $\mathcal{B}$ guesses the challenge message with probability $1/2$. Thus, if $\mathcal{A}$ succeeds with probability ϵ , then $\mathcal{B}$ succeeds with probability $\epsilon + (1 - \epsilon)/2 = 1/2 + \epsilon/2$. $\square$

6 Computing Weighted Averages and Fourier Transforms

So far we only constructed schemes for *univariate* predicates P . We now give an example where one computes on multiple signed messages. Let p be a prime, n a positive integer, and $\mathcal{T}$ a set of tags. The message space $\mathcal{M}$ consists of pairs:

$$\mathcal{M} := \mathcal{T} \times \mathbb{F}_p^n$$

Now, define the predicate P as follows: $P(\varepsilon, m) = 1$ for all $m \in \mathcal{M}$ and¹¹

$$P\left(\left((t_1, \mathbf{v}_1), \dots, (t_k, \mathbf{v}_k) \right), (t, \mathbf{v}) \right) = 1 \iff \begin{cases} t = t_1 = \dots = t_k, \text{ and} \\ \mathbf{v} \in \text{span}(\mathbf{v}_1, \dots, \mathbf{v}_k) \end{cases}$$

Thus, given signatures on vectors $\mathbf{v}_1, \dots, \mathbf{v}_k$ grouped together by the tag t , anyone can create a signature on a linear combination of these vectors. This can be done iteratively so that given signed linear combinations, new signed linear combinations can be created. Unforgeability means that if the adversary obtains signatures on vectors $\mathbf{v}_1, \dots, \mathbf{v}_k$ for particular tag $t \in \mathcal{T}$ then he cannot create a signature on a vector outside the linear span of $\mathbf{v}_1, \dots, \mathbf{v}_k$.

Signature schemes for this predicate P are presented in [15, 14, 13, 16, 2] while schemes over $\mathbb{Z}$ (rather than $\mathbb{F}_p$) are presented in [28]. These schemes were originally designed to secure network coding where context hiding is not needed since there are no privacy requirements for the sender (in fact, the sender is explicitly transmitting all his data to the recipient). The question then is how to construct a system for predicate P above that is both unforgeable and context hiding. Fortunately, we do not need to look very far. The linearly homomorphic signature schemes in [15] can be shown to be context hiding. We state this in the following theorem.

Theorem 6.1 *If the CDH assumption holds in group $\mathbb{G}$, then the signature scheme $\mathbf{NCS}_1$ from [15] is unforgeable and context-hiding in the random oracle model, assuming tags are generated independently at random by the unforgeability challenger when responding to Sign queries.*

¹¹Recall, the signature on ϵ is the output the KeyGen algorithm.

Unforgeability is Theorem 6 in [15], which holds only when tags $t_i \in \mathcal{T}$ are generated independently at random by the signer. While context hiding has not been considered before for this scheme, it is not difficult to show that the scheme is context hiding. The scheme is unique in the sense that every vector $\mathbf{v}$ has a unique valid signature.¹² It is easy to show that any homomorphic unique signature must be context hiding and hence $\mathbf{NCS}_1$ is context hiding.

Viewed in this way, the scheme $\mathbf{NCS}_1$ gives the ability to carry out authenticated addition on signed data. Consider a server that stores signed data samples $s_1, \dots, s_n$ in $\mathbb{F}_p$. The signature on sample s_i is actually a signature on the vector $(s_i | \mathbf{e}_i) \in \mathbb{F}_p^{n+1}$, where $\mathbf{e}_i$ is the i th unit vector in $\mathbb{F}_p^n$. The server stores (i, s_i) and a signature on $(s_i | \mathbf{e}_i)$. (The vector $\mathbf{e}_i$ need not be stored with the data and can be reconstructed from i when needed.) Using **SignDerive**, the server can compute a signature σ on the sum $(\sum_{i=1}^n s_i, 1, \dots, 1)$. Since the schemes are context hiding, the server can publish the sum $\sum_{i=1}^n s_i \bmod p$ and the signature σ on the sum and (provably) reveal no other information on the original data. The “augmentation” $(1, \dots, 1)$ proves that the published message really is the claimed sum of the original samples (the tag t prevents mix-and-match attacks between different data sets). We can similarly publish a sum of any required subset.

More generally, the server can publish an authenticated inner product of the samples $\mathbf{s} := (s_1, \dots, s_n)$ with any public vector $\mathbf{c} \in \mathbb{F}_p^n$ without leaking additional information about the samples. This is needed, for example, to publish a weighted average of the original data set. Similarly, recall that the Fourier transform of the data $(s_1, \dots, s_n)$ is a specific linear operator (represented by a specific $n \times n$ matrix) applied to this vector. Therefore, we can publish signed Fourier coefficients of the data. If we only publish a subset of the Fourier coefficients then, by context hiding, we are guaranteed that no other information about $(s_1, \dots, s_n)$ is revealed.

7 Conclusion and Open Problems

In summary, the goal of this work is the study of computing on authenticated data. We presented one unified framework for a variety of distinct concepts in this area, including arithmetic, homomorphic, quotable, redactable and transitive signatures. The definition we provide tackles unforgeability as well as a strong notion of privacy, where an adversary is unable to distinguish a derived signature from a fresh one even when given the original signature. In this setting, we provide generic constructions for all univariate and closed predicates, as well as specific efficient constructions for predicates such as quoting, subsets, weighted sums, averages and Fourier transforms.

This work leaves open a host of interesting problems. First, one can imagine predicates that support more complex operations on signed messages. One natural set of examples are spreadsheet operations such as median, standard deviation, and rounding on signed data (satisfying unforgeability and context hiding). Other examples include graph algorithms such as computing a signature on a perfect matching in a signed bipartite graph. Still other examples involve efficiently expanding quoting/redacting to more complex data types, such as (potentially compressed) graphical images.

A first step in this direction may be to improve upon some of the constructions for basic predicates presented herein. For example, as discussed at the end of Section 4.2, for quoting/redacting on simple text, it is still unknown how to balance time and space efficiently while achieving full security in the standard model from a simple computational assumption.

¹²Recall that in *unique* signatures [39] in addition to the regular unforgeability requirement there is an additional uniqueness property: for any honestly-generated public key pk and any message m in the message space, there do not exist values σ_1, σ_2 such that $\sigma_1 \neq \sigma_2$ and yet $\mathbf{Verify}(pk, m, \sigma_1) = \mathbf{Verify}(pk, m, \sigma_2) = 1$.

References

- [1] Giuseppe Ateniese, Daniel H. Chou, Breno de Medeiros, and Gene Tsudik. Sanitizable signatures. In *ESORICS '05*, volume 3679 of LNCS, pages 159–177, 2005.
- [2] Nuttapon Attrapadung and Benoit Libert. Homomorphic network coding signatures in the standard model. In *Public Key Cryptography - PKC 2011*, volume 6571, page 17, 2011.
- [3] Amos Beimel. *Secure Schemes for Secret Sharing and Key Distribution*. PhD thesis, Israel Institute of Technology, Technion, Haifa, Israel, 1996.
- [4] Mihir Bellare, Oded Goldreich, and Shafi Goldwasser. Incremental cryptography: The case of hashing and signing. In *CRYPTO '94*, volume 839 of LNCS, pages 216–233, 1994.
- [5] Mihir Bellare, Daniele Micciancio, and Bogdan Warinschi. Foundations of group signatures: Formal definitions, simplified requirements, and a construction based on general assumptions. In *EUROCRYPT*, pages 614–629, 2003.
- [6] Mihir Bellare and Gregory Neven. Transitive signatures based on factoring and RSA. In *ASIACRYPT '02*, volume 2501 of LNCS, pages 397–414, 2002.
- [7] Mihir Bellare and Gregory Neven. Transitive signatures: New schemes and proofs. *IEEE Transactions on Information Theory*, 51:2133–2151, 2005.
- [8] John Bethencourt, Amit Sahai, and Brent Waters. Ciphertext-policy attribute-based encryption. In *IEEE Symposium on Security and Privacy*, pages 321–334, 2007.
- [9] Manuel Blum, Alfredo De Santis, Silvio Micali, and Giuseppe Persiano. Noninteractive zero-knowledge. *SIAM Journal of Computing*, 20(6):1084–1118, 1991.
- [10] Dan Boneh and Xavier Boyen. Efficient selective-ID secure identity-based encryption without random oracles. In *Advances in Cryptology – EUROCRYPT '04*, volume 3027, pages 223–238, 2004.
- [11] Dan Boneh, Xavier Boyen, and Hovav Shacham. Short group signatures. In *CRYPTO '04*, volume 3152 of LNCS, pages 45–55, 2004.
- [12] Dan Boneh and Matthew K. Franklin. Identity-based encryption from the Weil pairing. *SIAM J. Comput.*, 32(3), 2003.
- [13] Dan Boneh and David Freeman. Homomorphic signatures for polynomial functions. In *Proc. of Eurocrypt*, 2011. Cryptology ePrint Archive, Report 2011/018.
- [14] Dan Boneh and David Freeman. Linearly homomorphic signatures over binary fields and new tools for lattice-based signatures. In *Proc. of PKC*, volume 6571 of LNCS, pages 1–16, 2011. Cryptology ePrint Archive, Report 2010/453.
- [15] Dan Boneh, David Freeman, Jonathan Katz, and Brent Waters. Signing a linear subspace: Signature schemes for network coding. In *Public-Key Cryptography — PKC '09*, volume 5443 of *Springer LNCS*, pages 68–87, 2009.
- [16] Dan Boneh and Michael Hamburg. Generalized identity based and broadcast encryption schemes. In *ASIACRYPT*, pages 455–470, 2008.

- [17] Christina Brzuska, Heike Busch, Özgür Dagdelen, Marc Fischlin, Martin Franz, Stefan Katzenbeisser, Mark Manulis, Cristina Onete, Andreas Peter, Bertram Poettering, and Dominique Schröder. Redactable signatures for tree-structured data: definitions and constructions. In *Applied Cryptography and Network Security (ACNS) '08*, volume 6123 of LNCS, pages 87–104, 2010.
- [18] Christina Brzuska, Marc Fischlin, Tobias Freudenreich, Anja Lehmann, Marcus Page, Jakob Schelbert, Dominique Schröder, and Florian Volk. Security of sanitizable signatures revisited. In *Public Key Cryptography*, volume 5443 of LNCS, pages 317–336, 2009.
- [19] Christina Brzuska, Marc Fischlin, Anja Lehmann, and Dominique Schröder. Santizable signatures: How to partially delegate control for authenticated data. In *BIOSIG 2009*, pages 117–128, 2009.
- [20] Christina Brzuska, Marc Fischlin, Anja Lehmann, and Dominique Schröder. Unlinkability of sanitizable signatures. In *Public Key Cryptography (PKC) '10*, volume 6056 of LNCS, pages 444–461, 2010.
- [21] Jan Camenisch and Anna Lysyanskaya. Signature schemes and anonymous credentials from bilinear maps. In *Advances in Cryptology – CRYPTO '04*, volume 3152, pages 56–72, 2004.
- [22] Ran Canetti, Shai Halevi, and Jonathan Katz. A forward-secure public-key encryption scheme. In *EUROCRYPT*, pages 255–271, 2003.
- [23] Ee-Chien Chang, Chee Liang Lim, and Jia Xu. Short redactable signatures using random trees. In *CT-RSA '09: Proceedings of the The Cryptographers' Track at the RSA Conference 2009 on Topics in Cryptology*, pages 133–147, 2009.
- [24] Denis Charles, K Jain, and K Lauter. Signatures for network coding. *International Journal of Information and Coding Theory*, 1(1):3–14, 2009.
- [25] David Chaum and Eugène van Heyst. Group signatures. In *EUROCRYPT*, volume 547 of LNCS, pages 257–265, 1991.
- [26] Whitfield Diffie and Martin Hellman. New directions in cryptography. *IEEE Transactions on Information Theory*, 22:644–654, 1976.
- [27] Christina Fragouli and Emina Soljanin. *Network Coding Fundamentals*. Now Publishers Inc., Hanover, MA, USA, 2007.
- [28] Rosario Gennaro, Jonathan Katz, Hugo Krawczyk, and Tal Rabin. Secure network coding over the integers. In *Public Key Cryptography — PKC '10*, volume 6056 of *Springer LNCS*, pages 142–160, 2010.
- [29] Craig Gentry. *A fully homomorphic encryption scheme*. PhD thesis, Stanford University, 2009.
- [30] Oded Goldreich, Shafi Goldwasser, and Silvio Micali. How to construct random functions (extended abstract). In *FOCS*, pages 464–479, 1984.
- [31] Shafi Goldwasser, Silvio Micali, and Ronald L. Rivest. A digital signature scheme secure against adaptive chosen-message attacks. *SIAM Journal of Computing*, 17(2):281–308, 1988.

- [32] Stuart Haber, Yasuo Hatano, Yoshinori Honda, William Horne, Kunihiko Miyazaki, Tomas Sander, Satoru Tezoku, and Danfeng Yao. Efficient signature schemes supporting redaction, pseudonymization, and data deidentification. In *ASIACCS '08*, pages 353–362, 2008.
- [33] Alejandro Hevia and Daniele Micciancio. The provable security of graph-based one-time signatures and extensions to algebraic signature schemes. In *ASIACRYPT '02*, volume 2501 of LNCS, pages 379–396, 2002.
- [34] Susan Hohenberger and Brent Waters. Realizing hash-and-sign signatures under standard assumptions. In *EUROCRYPT '09*, volume 5479 of LNCS, pages 333–350, 2009.
- [35] Robert Johnson, David Molnar, Dawn Song, and David Wagner. Homomorphic signature schemes. In *CT-RSA*, pages 244–262. Springer-Verlag, 2002.
- [36] M. Krohn, M. Freedman, and D. Mazieres. On-the-fly verification of rateless erasure codes for efficient content distribution. In *Proc. of IEEE Symposium on Security and Privacy*, pages 226–240, 2004.
- [37] Allison B. Lewko, Tatsuaki Okamoto, Amit Sahai, Katsuyuki Takashima, and Brent Waters. Fully secure functional encryption: Attribute-based encryption and (hierarchical) inner product encryption. In *EUROCRYPT*, 2010.
- [38] Allison B. Lewko and Brent Waters. New techniques for dual system encryption and fully secure HIBE with short ciphertexts. In *TCC '10*, volume 5978 of LNCS, pages 455–479, 2010.
- [39] Anna Lysyanskaya. Unique signatures and verifiable random functions from the DH-DDH separation. In *CRYPTO*, pages 597–612, 2002.
- [40] Silvio Micali. Computationally sound proofs. *SIAM J. Comput.*, 30(4):1253–1298, 2000.
- [41] Silvio Micali and Ronald L. Rivest. Transitive signature schemes. In *CT-RSA '02*, volume 2271 of LNCS, pages 236–243, 2002.
- [42] Kunihiko Miyazaki, Goichiro Hanaoka, and Hideki Imai. Digitally signed document sanitizing scheme based on bilinear maps. In *ASIACCS '06: Proceedings of the 2006 ACM Symposium on Information, computer and communications security*, pages 343–354, 2006.
- [43] Kunihiko Miyazaki, Mitsuru Iwamura, Tsutomu Matsumoto, Ryoichi Sasaki, Hiroshi Yoshiura, Satoru Tezuka, and Hideki Imai. Digitally signed document sanitizing scheme with disclosure condition control. *IEICE Transactions on Fundamentals*, E88-A(1):239–246, 2005.
- [44] Kunihiko Miyazaki, Seiichi Susaki, Mitsuru Iwamura, Tsutomu Matsumoto, Ryoichi Sasaki, and Hiroshi Yoshiura. Digital document sanitizing problem. *IEICE Technical Report*, 103(195(ISEC2003 12-29)):61–67, 2003.
- [45] David Naccache. Is theoretical cryptography any good in practice? CHES 2010 invited talk, 2010. available at www.iacr.org/workshops/ches/ches2010.
- [46] Gregory Neven. A simple transitive signature scheme for directed trees. *Theoretical Computer Science*, 396(1-3):277–282, 2008.
- [47] Ronald Rivest. Two signature schemes. Slides from talk given at Cambridge University, 2000. <http://people.csail.mit.edu/rivest/Rivest-CambridgeTalk.pdf>.

- [48] Ronald L. Rivest, Adi Shamir, and Leonard Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Comm. of the ACM*, 21(2):120–126, February 1978.
- [49] Ronald L. Rivest, Adi Shamir, and Yael Tauman. How to leak a secret: Theory and applications of ring signatures. In *Essays in Memory of Shimon Even*, pages 164–186, 2006.
- [50] Siamak Fayyaz Shahandashti, Mahmoud Salmasizadeh, and Javad Mohajer. A provably secure short transitive signature scheme from bilinear group pairs. In *Security and Communication Networks*, volume 3352 of *LNCS*, page 6076, 2005.
- [51] Adi Shamir. On the generation of cryptographically strong pseudorandom sequences. *ACM Transaction on Computer Systems*, 1:38–44, 1983.
- [52] N. P. Smart and F. Vercauteren. Fully homomorphic encryption with relatively small key and ciphertext sizes. In *Public Key Cryptography — PKC ’10*, volume 6056 of *Springer LNCS*, pages 420–443, 2010.
- [53] Nigel Smart. ECRYPT2 Yearly Report on Algorithms and Keysizes (2008-2009), Revision 1.0, July 27, 2009. Edited by Smart. Available at <http://www.ecrypt.eu.org/documents/D.SPA.7.pdf>.
- [54] Ron Steinfeld, Laurence Bull, and Yuliang Zheng. Context extraction signatures. In *Information Security and Cryptology (ICISC)*, volume 2288 of *LNCS*, pages 285–304, 2001.
- [55] Marten van Dijk, Craig Gentry, Shai Halevi, and Vinod Vaikuntanathan. Fully homomorphic encryption over the integers. In *Advances in Cryptology — EUROCRYPT ’10*, volume 6110 of *Springer LNCS*, pages 24–43, 2010.
- [56] Brent Waters. Efficient identity-based encryption without random oracles. In *Advances in Cryptology – EUROCRYPT ’05*, volume 3494, pages 320–329, 2005.
- [57] Brent Waters. Dual system encryption: Realizing fully secure IBE and HIBE under simple assumptions. In *Advances in Cryptology – CRYPTO ’09*, volume 5677, pages 619–636, 2009.
- [58] Brent Waters. Ciphertext-policy attribute-based encryption: an expressive, efficient, and provably secure realization. In *Public Key Cryptography — PKC ’11*, pages 53–70, 2011.
- [59] Lei Wei, Scott E. Coull, and Michael K. Reiter. Bounded vector signatures and their applications. In *ASIACCS ’11*, pages 277–285, 2011.
- [60] Xun Yi. Directed transitive signature scheme. In *CT-RSA ’07*, volume 4377 of *LNCS*, page 129144, 2007.
- [61] Fang Zhao, Ton Kalker, Muriel Médard, and Keesook Han. Signatures for content distribution with network coding. In *Proc. Intl. Symp. Info. Theory (ISIT)*, 2007.

A A Computational Definition of Context Hiding

Let $(\mathbf{KeyGen}, \mathbf{SignDerive}, \mathbf{Verify})$ be a P -homomorphic signature scheme for predicate P and message $\mathcal{M}$. Consider the following game to model context hiding:

Setup: The challenger runs the algorithm $(pk, sk) \leftarrow \mathbf{KeyGen}(1^\lambda)$ to obtain the public key pk and the secret key sk , and gives pk to the adversary.

Query Phase 1: Proceeding adaptively, the adversary may query any of the three oracles from the unforgeability game:

- $\text{Sign}(m \in \mathcal{M})$: (same as in the unforgeability game)
- $\text{SignDerive}(i \in \mathbb{Z}, m')$: (same as in the unforgeability game)
- $\text{Reveal}(i \in \mathbb{Z})$: (same as in the unforgeability game)

Challenge: At some point, the adversary issues a challenge (m, m') where $P(m, m') = 1$ for any $m, m' \in \mathcal{M}$. The challenger computes the following three values: $\sigma \leftarrow \mathbf{Sign}(sk, m)$, $\sigma_0 \leftarrow \mathbf{Sign}(sk, m')$ and $\sigma_1 \leftarrow \mathbf{SignDerive}(pk, \sigma, m, m')$. The challenger then picks a random $b \in \{0, 1\}$ and returns (σ, σ_b) to the adversary. Note: there are no restrictions on m, m' other than that they be in the message space; in particular, they could be equal and one or both could have been previously signed.

Query Phase 2: Proceeding adaptively, the adversary may query the oracles from Phase 1.

Output: Eventually, the adversary will output a bit b' and is said to win if $b = b'$.

We define $\mathbf{Adv}_{\mathcal{A}}^{\text{CH}}$ to be the probability that adversary $\mathcal{A}$ wins in the above game minus $\frac{1}{2}$.

Definition A.1 (Context Hiding) For a predicate P and message space $\mathcal{M}$, a P -homomorphic signature scheme $(\mathbf{KeyGen}, \mathbf{Sign}, \mathbf{SignDerive}, \mathbf{Verify})$ is context hiding if for all probabilistic polynomial time adversaries $\mathcal{A}$, $\mathbf{Adv}_{\mathcal{A}}^{\text{CH}}$ is negligible in λ .

A.1 Relation to Strong Context Hiding

Lemma A.2 A homomorphic signature scheme that is strongly context hiding is context hiding.

Proof. (Sketch) Let $\Pi = (\mathbf{KeyGen}, \mathbf{SignDerive}, \mathbf{Verify})$ be a homomorphic signature scheme and let A be an adversary that has advantage $\mathbf{Adv}_A^{\text{CH}}(\Pi) = p(\lambda)$ in the context-hiding game. The advantage probability for A is taken over the random coins of the key generation, random coins of the Sign and SignDerive operations used in the first query phase, the random coins used by algorithm A , and the random coins used by the rest of the experiment. Therefore by an averaging argument, there must exist *some particular* key pair $(PK, SK) \leftarrow \mathbf{KeyGen}(1^\lambda; z_1)$, *some particular* random tape z_q for the $\mathbf{Sign}$ and $\mathbf{SignDerive}$ operations used in the first query phase, *some particular* random coins z_A for A , and *some particular* message pair (m, m') output by A over which the probability of A winning the context-hiding game in this case is at least $p(\lambda)$. Let the values of the random tapes be given as non-uniform advice.

We show how this information can be used to construct a (non-uniform) adversary A' that distinguishes $\{(SK, \sigma, \mathbf{Sign}(SK, m'))\}$ from $\{(SK, \sigma, \mathbf{SignDerive}(PK, \sigma, m, m'))\}$ with probability $p(\lambda)$ for the triple $((PK, SK), m, m')$. Thus, if Π is strongly context hiding, then $p(\lambda)$ must be exponentially small, and so Π must also be context-hiding.

The adversary A' works as follows: On input the challenge tuple (SK, σ, σ') , A' begins to run the context-hiding experiment for $A(PK; z_A)$. A' answers the queries that A asks by using SK and the random tape z_q to run $\mathbf{Sign}$ and $\mathbf{SignDerive}$. When A outputs a challenge message pair (m, m') (which must occur by construction), then A' answers with (σ, σ') . A' answers the second-phase queries of A using SK and fresh random coins. Finally, when A outputs b' , A' echoes this answer as output and halts.

First observe that A' performs a perfect simulation of the context-hiding game. When the input pair (σ, σ') corresponds to $(\mathbf{Sign}(SK, m), \mathbf{Sign}(SK, m'))$, then A' simulates the context-hiding

game for $b = 0$. In the other case, A' simulates the context-hiding game for $b = 1$. Therefore, A' distinguishes

$$\begin{aligned} & \{(SK, \mathbf{Sign}(SK, m), \mathbf{Sign}(SK, m'))\}_{SK, m, m'} \\ & \{(SK, \mathbf{Sign}(SK, m), \mathbf{SignDerive}(PK, \sigma, m, m'))\}_{SK, m, m'} \end{aligned}$$

with probability $p(\lambda)$. $\square$

A.2 Simplified Unforgeability Under Strong Context Hiding

We now show how the strong context hiding property can help simplify the security argument for unforgeability. In particular, we introduce a weaker notion of unforgeability in which the adversary only makes calls to the **Sign** oracle and immediately receives a signature.

— Game **NHU**($\Pi, \mathcal{A}, \lambda, P$): This game is the same as the **Unforg**($\Pi, \mathcal{A}, \lambda, P$) game with the exception that only the following query is allowed:

— $\text{Sign}(m \in \mathcal{M})$: the oracle computes $\sigma \leftarrow \mathbf{Sign}(SK, m)$, adds m to Q and returns σ .

Note, the only difference between game **NHU** and the standard unforgeability game for a signature scheme is that in this game, the adversary only wins if it produces a forgery on a signature m^* such that for all $m \in Q$, $P(m, m^*) = 0$, whereas in the standard unforgeability game, the adversary wins if it produces a signature on *any* message that is not in Q .

Definition A.3 A quoteable signature scheme Π is **NHU-unforgeable** if for all efficient adversaries $\mathcal{A}$, it holds that $\Pr[\mathbf{NHU}(\Pi, \mathcal{A}, \lambda, P) = 1] < \text{negl}(\lambda)$ for some negligible function λ .

Lemma A.4 A signature scheme that is **NHU-unforgeable** and strongly context hiding is **Unforg-unforgeable**.

Proof. Our plan is to present a series of hybrid experiments that are meant to simplify the quotable unforgeability game.

Hybrid $H_1(\Pi, \mathcal{A}, \lambda, P)$ Consider the first hybrid experiment H_1 which is the same as the unforgeability game **Unforg**($\Pi, \mathcal{A}, \lambda, P$), with the exception that all Sign and SignDerive queries are lazily evaluated. That is, when $\mathcal{A}$ makes a query, the experiment responds in the following way:

- $\text{Sign}(m)$: generate a handle i and record information $(i, ?, m, \epsilon)$ in T and return i
- $\text{SignDerive}(i, m')$: retrieve $(i, z, m, \cdot)$ from T , return $\perp$ if it does not exist or if $P(m, m') \neq 1$, generate a new handle i' , record $(i', ?, m', i)$ in T , and return i'
- $\text{Reveal}(i)$: retrieve (i, z, m, i_1) from T (returning $\perp$ if it does not exist). If $z \neq ?$, then return z . Otherwise, if $i_1 = \epsilon$, then compute $\sigma \leftarrow \mathbf{Sign}(SK, m)$, replace the entry (i, z, m, ϵ) with (i, σ, m, ϵ) , and return σ . Finally, if $i_1 \neq \epsilon$, then recursively call $z_1 \leftarrow \text{Reveal}(i_1)$, obtain $(i_1, \cdot, m_1, \cdot)$ from T and compute $\sigma \leftarrow \mathbf{SignDerive}(PK, z_1, m_1, m)$. Replace the entry with (i, σ, m, i_1) , and return σ .

Claim A.5 $\Pr[H_1(\Pi, \mathcal{A}, \lambda, P) = 1] = \Pr[\mathbf{Unforg}(\Pi, \mathcal{A}, \lambda, P) = 1]$.

This claim follows by inspection. For any query that is eventually revealed, the same operations are performed in both H_1 and the original game. For any query that is never revealed, no operation in H_1 is performed; but this does not affect the view of the adversary, and therefore does not affect the output of the adversary.

Hybrid $H_{2,i}(\Pi, \mathcal{A}, \lambda, P)$ The second hybrid is the same as H_1 except that the first i queries to *Reveal* are answered using *Reveal₂* described below, and the remaining queries are answered as per H_1 : (The only difference is that **Sign**(SK, m_1) is used in place of **SignDerive**(PK, z_1, m_1, m) in the second to last sentence.)

- *Reveal₂*(i): retrieve (i, z, m, i_1) from T (returning $\perp$ if it does not exist). If $z \neq ?$, then return z . Otherwise, if $i_1 = \epsilon$, then compute $\sigma \leftarrow \mathbf{Sign}(SK, m)$, replace the entry (i, z, m, ϵ) with (i, σ, m, ϵ) , and return σ . Finally, if $i_1 \neq \epsilon$, then recursively call $z_1 \leftarrow \mathbf{Reveal}(i_1)$, obtain $(i_1, \cdot, m_1, \cdot)$ from T and compute $\sigma \leftarrow \mathbf{Sign}(SK, m_1)$. Replace the entry with (i, σ, m, i_1) , and return σ .

Claim A.6 $H_{2,0}(\Pi, \mathcal{A}, \lambda, P)$ is identically distributed to $H_1(\Pi, \mathcal{A}, \lambda, P)$.

By inspection.

Claim A.7 $H_{2,i}(\Pi, \mathcal{A}, \lambda, P)$ is identically distributed to $H_{2,i-1}(\Pi, \mathcal{A}, \lambda, P)$ for $i \geq 1$.

This claim follows via the strong context-hiding property of the signature scheme because this property guarantees **Sign**(SK, m') and **SignDerive**(PK, σ, m, m') are statistically close.

Suppose that $\mathcal{A}$ makes $\ell = \text{poly}(\lambda)$ queries. Observe that $H_{2,\ell}(\Pi, \mathcal{A}, \lambda, P)$ only evaluates **Sign**, and only does so on messages that are immediately returned to the adversary. Thus, $H_{2,\ell}$ is syntactically equivalent to the **NHU** game. Since the $H_{2,\ell}$ game enables $\mathcal{A}$ to produce a forgery with the same probability as **Unforg**($\Pi, \mathcal{A}, \lambda, P$), we have that **Unforg**($\Pi, \mathcal{A}, \lambda, P$) = **NHU**($\Pi, \mathcal{A}, \lambda, P$) which completes the lemma. $\square$

Universal Signature Aggregators

Susan Hohenberger*
Johns Hopkins University
susan@cs.jhu.edu

Venkata Koppula
University of Texas at Austin
kvenkata@cs.utexas.edu

Brent Waters†
University of Texas at Austin
bwaters@cs.utexas.edu

Abstract

We introduce the concept of universal signature aggregators. In a universal signature aggregator system, a third party, using a set of common reference parameters, can aggregate a collection of signatures produced from *any* set of signing algorithms (subject to a chosen length constraint) into one short signature whose length is independent of the number of signatures aggregated. In prior aggregation works, signatures can only be aggregated if all signers use the same signing algorithm (e.g., BLS) and shared parameters. A universal aggregator can aggregate across schemes even in various algebraic settings (e.g., BLS, RSA, ECDSA), thus creating novel opportunities for compressing authentication overhead. It is especially compelling that *existing* public key infrastructures can be used and that the signers do not have to alter their behavior to enable aggregation of their signatures.

We provide multiple constructions and proofs of universal signature aggregators based on indistinguishability obfuscation and other supporting primitives. We detail our techniques as well as the tradeoffs in features and security of our solutions.

1 Introduction

An aggregate signature system, as introduced by Boneh, Gentry, Lynn and Shacham [BGLS03], allows a party to bundle a set of signatures together into a single short cryptographic signature. Aggregate signatures are motivated by applications where one needs to simultaneously verify several signatures from different users on different messages in environments with communication or storage resource constraints. For example, Boneh et al. [BGLS03] proposed applying aggregate signatures to Secure BGP [KLMS00] path authentication; later this idea was empirically evaluated by Zhao et al. [ZSN05].

Over the past several years many solutions to aggregate signatures [LOS⁺06, GR06, BGOY07, BNN07, AGH10] have been proposed that have explored tradeoffs regarding computational cost, security models, features (e.g. identity-based), limitations (e.g. sequential signing), and cryptographic assumptions. However, all of these constructions have one thing in common in that they require *all signers to adopt a common signature system and shared parameters*.

In practice, the common scheme and parameter requirements can be a large barrier to adoption. Existing users will already have established signing keys and algorithms which are entrenched in an existing public key infrastructure. The overhead of changing and re-certifying one's public keys may very well overwhelm

*Supported by the National Science Foundation (NSF) CNS-1154035 and CNS-1228443; the Defense Advanced Research Projects Agency (DARPA) and the Air Force Research Laboratory (AFRL) under contract FA8750-11-2-0211, the Office of Naval Research under contract N00014-14-1-0333, and a Microsoft Faculty Fellowship.

†Supported by NSF CNS-0952692, CNS-1228599 and CNS-1414082. DARPA through the U.S. Office of Naval Research under Contract N00014-11-1-0382, Google Faculty Research award, the Alfred P. Sloan Fellowship, Microsoft Faculty Fellowship, and Packard Foundation Fellowship.

the perceived benefit of creating signatures that can be aggregated by a third party. Indeed the original signer might not even be incentivized to allow aggregation in the first place when the benefits fall to the aggregating party or verifier of the signatures. Furthermore, even if a user moved from one signature system to an aggregate signature system, all previously created signatures would be unaggregatable.¹

Universal Signature Aggregators We introduce the concept of universal signature aggregators. In a universal signature aggregator system, a third party, using a set of common reference parameters, can aggregate a collection of signatures produced from *any* set of signing algorithms (subject to a chosen length constraint) into one short signature whose length is independent of the number of signatures aggregated. A verifier can use the common parameters to verify the aggregate signature. The system will be secure in the sense that it is hard to produce an aggregate signature on a verification algorithm, verification key, message tuple, $(\text{Verify}, \text{VK}, m)$ unless the holder of the corresponding secret key produced a signature on m . We emphasize that signers in the system need not do anything special to allow aggregation; indeed they could be unaware of the existence of such a system.

Our central challenge is to create a way to compress many signatures of varying types into one short object. Prior solutions required all signatures to reside in a common (often bilinear) group, where it was possible to leverage homomorphic properties of the group structure. Here we are afforded no such luxury as signatures will reside in different groups or even be based on a scheme with no algebraic structure.

Our approach will be to overcome these limitations by applying the tool of program obfuscation. At the highest level, a trusted setup routine will produce a pair of a global signature verification key for a universal signature aggregator and a shared obfuscated program. The job of the obfuscated program will be to take as input tuples of the form $(\text{Verify}, \text{VK}, m, \sigma)$ that respectively represent verification algorithm, verification key, message and signature 4-tuples. The program will first verify using algorithm Verify and key VK that σ is a signature on m . If this check passes, it will produce a signature using a master secret key on the message $\text{MSG} = (\text{Verify}, \text{VK}, m)$ — essentially transforming the arbitrary signature into one of an aggregatable form.

At first glance it might appear that obfuscation provides an open and close solution to our problem. Indeed, if we heuristically model the obfuscated program as an oracle to the program, the analysis is relatively straightforward. However, as noted by Hada [Had00] such a definition is impossible to achieve for any functionality. Our goal is to create probably secure constructions under a realizable definition of obfuscation — ideally indistinguishability obfuscation.

Achieving provable security under indistinguishability obfuscation (and without knowledge assumptions²) presents significant challenges. The primary technical challenge is how to design a construction and corresponding reduction that can extract a forgery on an arbitrary input signature scheme from an attacker that forges on the aggregate. We emphasize that without an oracle interface or knowledge assumption a reduction is not afforded the opportunity to simply “look at” the input signatures.

Universally Aggregating Unique Signatures We begin by exploring how to universally aggregate unique signatures — a unique signature system [GO93] is one where there is at most one signature that will verify per message. Notably, RSA based full domain hash [BR93, BR96] are unique signatures that form the basis of the widely deployed PKCS#1 standard [KS98]. As evidence of the wide scale deployment, Heninger et al. [HDWH12] performed an Internet-wide scan of machines responding on the TLS and SSH ports for IPv4 space and reported 3.9 million distinct RSA keys compared to only 1.9 thousand DSA keys.

Our construction will be parameterized by four polynomial functions over the security parameter: $\ell_{\text{ver}}(\lambda)$, $\ell_{\text{vk}}(\lambda)$, $\ell_{\text{msg}}(\lambda)$, $\ell_{\text{sig}}(\lambda)$. These respectively represent a bound on the size of verification circuits, verification keys, length of messages signed and size of signatures that are aggregated. While we are interested in signatures of arbitrary length messages, in practice almost all signature schemes will apply the “hash and

¹We note that the concept of integrating “special property” cryptography into existing keys is relatively unexplored, but has been considered in ring signatures [BKM08] and deniable encryption [SW14].

²A different direction is to attempt to build universal aggregation from succinct arguments of knowledge (SNARKs)[BCCT13]. We aim to achieve our goals without applying knowledge assumptions.

sign” paradigm where a longer message is first hashed down to a fixed size hash value (dependent on the security parameter). The core signature scheme then signs this value.

In our first construction (see Section 4), the **UniversalSetup** first chooses an RSA modulus N and exponent $e \leftarrow \mathbb{Z}_{\phi(N)}^*$. Next, it chooses a puncturable PRF [BW13, BGI13, KPTZ13, SW14] key K for a function F that takes inputs of the form $(\text{Verify}, \text{VK}, m) \in \{0, 1\}^{\ell_{\text{ver}}} \times \{0, 1\}^{\ell_{\text{vk}}} \times \{0, 1\}^{\ell_{\text{msg}}}$ (i.e., 3-tuples representing a verification circuit, verification key and message). The puncturable PRF will output into $\mathbb{Z}_N$.

Finally, the setup will publish (indistinguishability) obfuscations of two programs. The first is **Transform** $_{N,K}$. This program takes as input a 4-tuple $\text{Verify}, \text{VK}, m, \sigma$. It then computes $\text{Verify}(\text{VK}, m, \sigma)$, which verifies the signature under the algorithm. If the signature verifies, the program outputs $F(K, \text{Verify}, \text{VK}, m) \in \mathbb{Z}_N$. This can be thought of as a “transformed signature” where the obfuscated program maps the original signature into one over $\mathbb{Z}_N$. The second program is called **Transform-Image** $_{N,K,e}$. On input $(\text{Verify}, \text{VK}, m)$, it computes $F(K, \text{Verify}, \text{VK}, m)^e \pmod{N}$.

One can now aggregate a sequence of signatures $(\text{Verify}_i, \text{VK}_i, m_i, \sigma_i)$ by transforming each one by computing³ $s_i = \text{Transform}_{N,K}(\text{Verify}_i, \text{VK}_i, m_i, \sigma_i)$ and then aggregating into one element of $\mathbb{Z}_N$ as $\sigma_{\text{agg}} = \prod_i s_i$. To verify an aggregate signature, σ_{agg} , on $(\text{Verify}_i, \text{VK}_i, m_i)$ simply compute $t_i = \text{Transform}_{N,K}(\text{Verify}_i, \text{VK}_i, m_i, \sigma_i)$ and test whether $\sigma_{\text{agg}} \stackrel{?}{=} \prod_i t_i$.⁴ Essentially the **Transform** program maps an arbitrary signature to an RSA FullDomain hash type signature on the “message” $(\text{Verify}_i, \text{VK}_i, m_i)$.

We prove selective security where the attacker declares before seeing the public parameters a message m^* that they will forge on.⁵ Our security argument is centered around an alternative program **Transform-Reject** which is programmed to behave the same as **Transform** except on input $y = (\text{Verify}^*, \text{VK}^*, m^*)$ on which it always outputs $\perp$ *even if it is given a valid signature on m^** . It also uses a PRF key that is punctured at y .

Security follows from two primary arguments about the program. We first establish that if an attack algorithm, **Att**, is successful when given **Transform**, it must be almost as successful when given **Transform-Reject**; otherwise, the underlying unique signature scheme is broken. Suppose that there is an attacker, **Att**, with a non-negligible difference in advantage between these two games, then we can build a reduction algorithm that extracts the unique signature on m^* in a bit by bit fashion. The reduction algorithm runs as the challenger in the aggregate signature game and receives a challenge verification key from the challenger in the standard signature security game. It runs to the point in the security game where the input public key and parameters are established and saves the state of the game (including the state of **Att**). Then for each bit of the signature it performs the following process multiple times. It runs a third program **TransformAlt** $_{y,j}$. This program runs as **Transform**, but rejects if the j -th bit of the input signature is 1. For each j , it runs the experiment multiple times with fresh randomness. If the measured advantage of the attacker drops when using **TransformAlt** $_{y,j}$ then it guesses that the j -th bit of the signature is 1; otherwise it guesses that it is 0. It compiles all of these guesses together to output a forgery. (The amount of rewinding needed depends on the difference in advantage. In addition, our actual analysis addresses other technical details.)

Since signatures are unique, the program **TransformAlt** $_{y,j}$ is functionally equivalent to **Transform** if the j -th bit of the unique signature on m^* is 0 and thus by indistinguishability obfuscation the attacker’s advantage should be negligibly close in these two cases. Similarly, **TransformAlt** $_{y,j}$ is functionally equivalent to **Transform-Reject** if the j -th signature bit is 1 and again by indistinguishability obfuscation the advantage should be close to that of **Transform-Reject**.

After we have established that the advantage when given **Transform-Reject** is close to that of **Transform**, we show that an attacker that can win when given **Transform-Reject** will either break $i\mathcal{O}$, the punctured PRF or the RSA assumption and roughly follows [HSW14] using punctured programming [SW14] techniques. The main proof innovation is combining a rewinding argument with indistinguishability obfuscation to extract a unique signature.

We also show a variation of this idea in Appendix A that is a universal aggregator of unique signatures,

³We slightly abuse notation in the introduction for ease of exposition by using the names **Transform** and **Transform-Image** to refer both to the obfuscated and unobfuscated forms of the program. In the main body, we are careful about these distinctions.

⁴We require in verification that no 3-tuples are repeated. I.e., for all $i \neq j$, $(\text{Verify}_i, \text{VK}_i, m_i) \neq (\text{Verify}_j, \text{VK}_j, m_j)$.

⁵The usual complexity leveraging arguments for adaptive security can be applied here if we are willing to make sub-exponential hardness assumptions.

but where we avoid using the RSA assumption. (Indistinguishability obfuscation and punctured PRFs are still used.) The tradeoff is that there is an a priori bound n on the number of signatures that can be aggregated. In the construction, the parameters will grow polynomially with n , but the size of the signatures is independent of n . We also conjecture that in our main construction the RSA-type transformed signature can be replaced by a BLS [BLS01] type signature (in an analogous way to [HSW14]), but do not formally show this.

Universal Aggregation of arbitrary signatures using VBB Obfuscation While covering unique signatures achieves progress, we want to push toward our central goal of aggregating arbitrary signatures. Our next step is to show that a slight tweak to the previous construction gives us a universal aggregator of arbitrary signatures under a specific virtual black box (VBB) assumption. This appears in Section 5.

It might first seem that a solution proven under a VBB assumption is not better than the oracle heuristic outlined earlier. However, achieving a VBB proof provides both a sounder justification and is more technically challenging than the oracle heuristic. First, modeling an obfuscated program as an oracle is a heuristic — a piece of code is clearly a different object than an oracle. In contrast, a VBB assumption could be true for many functionalities even though there exists certain functionalities for which it cannot hold [BGI⁺01].

Proving our construction secure under a VBB definition presents an interesting technical barrier. A natural proof methodology is to first say that an obfuscator for a given circuit cannot be more successful than a simulator with oracle access to the same circuit using VBB. And then making further hybrid security arguments leveraging the fact that the simulator has oracle access. The primary problem with this strategy is that while the universal aggregator security game gives the attacker access to a signing oracle, there is no place to “put” this signing oracle when applying the VBB security game.

We overcome this obstacle by introducing a new technique that we call “oracle assimilation” which we believe might be of independent interest. In our construction, the **Transform-VBB** program behaves in almost the same way as **Transform** before except an extra mode bit is added to the input. If this mode bit b is set to 0, it indicates **normal** input and the **Transform-VBB** program operates roughly as described above. If the mode bit is set to 1, it indicates **query** input and the program outputs a rejecting $\perp$ on all inputs of this type. The **query** type input is only used in the proof and not in the construction.

Our proof of security proceeds by a sequence of games. In the initial security game, all **query** inputs output a rejecting $\perp$. The proof (in a couple of steps) then moves to a game where the **query** inputs will take a form of (a, m) and output a signature on m under the challenge input secret signing key if $\text{PRG}(a) = \text{PRG}(\alpha)$ for some value α chosen by the game, but hidden from the attacker. We can argue this change is indiscernable to the attacker by $i\mathcal{O}$ and pseudorandom generator security. At this point the security game will use the **query** interface of the obfuscated program to answer signing queries and we can say that the signing oracle was “assimilated” into the obfuscated program. Next, we can use VBB security to argue that there must exist a simulator with oracle access to the program that outputs 1 with close to the same probability that the attacker wins. Now that the input signing algorithm is accessed by an oracle we can use its security to argue that the game is indistinguishable from when the circuit refuses to transform on m^* , the challenge message.⁶ Finally, we use VBB again to reason about the attack algorithm’s advantage when given this second circuit that will not transform on m^* . From here, the proof follows as in the unique signature case.

Stepping back, the main innovation for this proof is to use punctured programming techniques to subliminally assimilate the signing oracle for one scheme into the obfuscated program, then use the VBB interface to execute the proof. We expect that this technique will be useful in other contexts. One interesting view is that we could apply either this VBB argument for arbitrary signatures or the previous $i\mathcal{O}$ argument for unique signatures to this single construction. So a user with any signature scheme would get VBB based security and if a user had a unique signature scheme, she would get the added benefit of $i\mathcal{O}$ based security.

⁶For ease of exposition, the proof in the main body proves selective security; however, we show how a minor transformation of the construction using admissible hash functions [BB04] gives adaptive security in Appendix B.

Aggregating arbitrary signatures using indistinguishability obfuscation For our final contribution, we return to our goal of aggregating arbitrary signatures using indistinguishability obfuscation. Our primary challenge again is how to extract an input forgery from the attacker in a proof. The previous two methods used the structure of a unique signature and an oracle interface, neither of which is available to us now.

We overview the main solution ideas and our proof approach. At a high level, we devise a means for being able to extract and check the validity of a single signature (from the aggregate) of our choice in the proof without the adversary being able to know which one we are “looking at”. Thus, we build our confidence in the validity of all the signatures by being able to check any given one of them. We call this an “enforce all by one” technique.

To do this, we first use additively (or singly) homomorphic encryption to combine the encryptions of several signatures together into one object t . Then we will have an obfuscated program generate a PRF-type signature component s on a message representing ciphertext tag t along with tuples $\{\text{Verify}_i, \text{VK}_i, m_i\}$ if the input contains valid signatures on each message. The output aggregate signature is $\sigma_{\text{agg}} = (t, s)$. Although the homomorphic ciphertext t will not be large enough to contain all of the input signatures, in the proof it can be used to remember one of the input signatures and thus provide us with an opportunity to extract a forgery on the input signature. The difficulty is in using $i\mathcal{O}$ to ensure that an attacker can only output a verifying $\sigma_{\text{agg}} = (t, s)$ on a ciphertext “tag” t that contains a proper forgery in the proof.

Diving in a little further in our solution, the setup algorithm will be parameterized by a polynomial $n(\cdot)$ that gives an a-priori bound on the number of signatures that can be verified. The size of the parameters will grow polynomially with n , but the signature size will be independent of it. The setup algorithm will output n ciphertexts $\{\text{count}_i \leftarrow \text{HE.enc}(\text{pk}, 0)\}_{i=1, \dots, n}$ each of which is an encryption of 0.

The universal aggregation algorithm takes input $\{\text{Verify}_i, \text{VK}_i, m_i, \sigma_i\}$. It then computes $t = \sum_i \text{count}_i \cdot \sigma_i$. Next it will input t and the tuples $\{\text{Verify}_i, \text{VK}_i, m_i, \sigma_i\}$ to an obfuscated program **AggSign** which will evaluate and output a punctured PRF on t and $\{\text{Verify}_i, \text{VK}_i, m_i\}$ if the input signatures verify. (We will return shortly to where the obfuscated program comes from.)

In proving security we perform a sequence of hybrids, where the first step of the hybrid is to guess an index j (incurring a $1/n$ loss) where the forgery occurs. Next, we change count_j to be an encryption of 1. This causes an honestly computed value t to be an encryption of the j -th signature that we will eventually use for extraction.

The challenge at this point is to come up with a formulation of the program **AggSign** for which we can prove security using indistinguishability arguments. We provide two approaches. In the first one (see Section 6), we allow **AggSign** to be created by a Universal Sampler (also called a Universal Parameters Scheme) as defined by Hofheinz et al. [HJK⁺14]. A Universal Sampler is allowed to adaptively sample from an arbitrary (efficiently computable) distribution. In this case we sample from an obfuscation of the **AggSign_t** program that is parameterized to only work with a given tag value t . As noted in [HJK⁺14], Universal Samplers are realizable in the random oracle model from indistinguishability obfuscation. So this solution will exist in the random oracle model as well. An advantage of Universal Samplers is that they can define the **AggSign_t** program adaptively.

We also propose a second variation of this solution in Section 7 that does not need the random oracle heuristic. Instead it applies complexity leveraging that requires assuming sub-exponential hardness of some of the underlying computational assumptions.

1.1 Summary of our results

Our results are summarized in the following table. The first column labels the construction. The remaining columns indicate: type of signatures that can be aggregated, selective or adaptive security, standard or random oracle model proofs, whether the aggregator is bounded or not, and finally, the cryptographic assumptions used in the security proof. In our assumptions, we prefix them with “subexp” to indicate if sub-exponential hardness is required for complexity leveraging. Since PRFs, PRGs, and (selectively-secure) puncturable PRFs are constructible from one-way functions, we list OWF as the assumption. UPS stands for a universal parameters scheme [HJK⁺14] (implied by $i\mathcal{O}$ in the random oracle model), HE stands for

singly homomorphic encryption, $i\mathcal{O}$ stands for indistinguishability obfuscation, and VBB stands for virtual black-box obfuscation, where we assume that VBB holds only for a certain limited family of circuits.

Construction	Type	Selective/ Adaptive	RO	Bounded Aggregator	Assumptions
Section 4	Unique	Selective	No	No	$i\mathcal{O}$, RSA, OWF
Section 5	Arbitrary	Selective ⁷	No	No	$i\mathcal{O}$, VBB, OWF
Section 6	Arbitrary	Adaptive	Yes	Yes	$i\mathcal{O}$, UPS, HE, OWF
Section 7	Arbitrary	Selective	No	Yes	subexp- $i\mathcal{O}$, HE, subexp-OWF

2 Preliminaries

2.1 Notations

For any set $\mathcal{X}$, $x \leftarrow \mathcal{X}$ denotes a uniformly random element drawn from $\mathcal{X}$. Given integers $\ell_{\text{ckt}}, \ell_{\text{inp}}, \ell_{\text{out}}$, let $\mathcal{C}[\ell_{\text{ckt}}, \ell_{\text{inp}}, \ell_{\text{out}}]$ denote the set of circuits that can be represented using ℓ_{ckt} bits, take ℓ_{inp} bits as input, and output ℓ_{out} bits.

2.2 Admissible Hash Functions

We recall the notion of *admissible hash functions* due to Boneh and Boyen [BB04]. Here we state a simplified definition from [HSW14].

Definition 2.1. Let l, n and θ be efficiently computable univariate polynomials. Let $h : \{0, 1\}^{l(\lambda)} \rightarrow \{0, 1\}^{n(\lambda)}$ be an efficiently computable function and AdmSample a PPT algorithm that takes as input 1^λ and an integer q , and outputs $u \in \{0, 1, \perp\}^{n(\lambda)}$. For any $u \in \{0, 1, \perp\}^{n(\lambda)}$, define $P_u : \{0, 1\}^{l(\lambda)} \rightarrow \{0, 1\}$ as follows: $P_u(x) = 0$ if for all $1 \leq j \leq n(\lambda)$, $h(x)_j \neq u_j$, else $P_u(x) = 1$ (where u_j denotes the j^{th} bit of u).

We say that $(h, \text{AdmSample})$ is θ -admissible if the following condition holds:

For any efficiently computable polynomial Q , for all $x^1, \dots, x^{Q(\lambda)}, x^* \in \{0, 1\}^{l(\lambda)}$, where $x^* \notin \{x^i\}_i$,

$$\Pr[(\forall i \leq Q(\lambda), P_u(x^i) = 1) \wedge P_u(x^*) = 0] \geq \frac{1}{\theta(Q(\lambda))}$$

where the probability is taken over $u \leftarrow \text{AdmSample}(1^\lambda, Q(\lambda))$.

Theorem 2.1 (Admissible Hash Function Family [BB04], simplified proof in [FHPS13]). For any efficiently computable polynomial l , there exist efficiently computable polynomials n, θ such that there exist θ -admissible function families mapping l bits to n bits.

2.3 Signature Schemes

A signature scheme $\mathcal{S}$ with message space $\mathcal{M}(\lambda)$, signature key space $\mathcal{SK}(\lambda)$ and verification key space $\mathcal{VK}(\lambda)$ consists of the following algorithms.

- $\text{Gen}(1^\lambda)$ The setup algorithm is a randomized algorithm that takes as input security parameter λ and outputs signing key $\text{SK} \in \mathcal{SK}$ and verification key $\text{VK} \in \mathcal{VK}$.
- $\text{Sign}(\text{SK}, m)$ The signing algorithm takes as input the signing key $\text{SK} \in \mathcal{SK}$ and a message $m \in \mathcal{M}$ and outputs a signature σ .
- $\text{Verify}(\text{VK}, m, \sigma)$ The verification algorithm takes as input a verification key $\text{VK} \in \mathcal{VK}$, message $m \in \mathcal{M}$ and signature σ and outputs either 0 or 1.

⁷In Appendix B, we modify this construction to achieve adaptive security without any additional assumptions.

Correctness For all $\lambda \in \mathbb{N}$, $(SK, VK) \leftarrow \text{Gen}(1^\lambda)$, messages $m \in \mathcal{M}(\lambda)$, we require that $\text{Verify}(VK, m, \text{Sign}(SK, m)) = 1$.

Security The security notion for signature schemes, formalized by Goldwasser, Micali and Rivest [GMR88], is based on the following game between an adversary $\mathcal{A}$ and a challenger.

1. **Setup Phase** Challenger chooses $(SK, VK) \leftarrow \text{Gen}(1^\lambda)$.
2. **Signing Phase** $\mathcal{A}$ sends signature query $m_i \in \mathcal{M}$ and receives $\sigma_i \leftarrow \text{Sign}(SK, m_i)$.
3. **Forgery Phase** $\mathcal{A}$ finally outputs a message m and signature σ .

$\mathcal{A}$ wins if m was not queried during the Signing Phase and $\text{Verify}(VK, m, \sigma) = 1$. Let $\text{Adv}_{\mathcal{A}}(\lambda) = \Pr[\mathcal{A} \text{ wins}]$.

Definition 2.2. A signature scheme $\mathcal{S} = (\text{Gen}, \text{Sign}, \text{Verify})$ is *existentially unforgeable under a chosen message attack* if for all PPT adversaries $\mathcal{A}$, $\text{Adv}_{\mathcal{A}}(\lambda)$ is negligible in λ .

Goldwasser and Ostrovsky [GO93] introduced the notion of *unique signature schemes*⁸. In a unique signature scheme, there is a unique valid signature corresponding to any message, verification key pair.

Definition 2.3 (Unique Signatures). A signature scheme $\mathcal{S} = (\text{Gen}, \text{Sign}, \text{Verify})$ is said to be *unique* if for all tuples $(VK, m, \sigma_1, \sigma_2)$, either $\sigma_1 = \sigma_2$ or $\text{Verify}(VK, m, \sigma_1) = 0$ or $\text{Verify}(VK, m, \sigma_2) = 0$.

In this work, we will be considering signature schemes where the messages, signatures and verification keys have bounded length, and the verification algorithm is deterministic. In practice, most signature schemes use a collision resistant hash function to compress an arbitrary length message to bounded length. We will be dealing with these ‘post-hash’ messages.

Definition 2.4 ($(\ell_{vk}, \ell_{msg}, \ell_{sig})$ -bounded length signature scheme). Let ℓ_{vk} , ℓ_{msg} and ℓ_{sig} be fixed polynomials. A signature scheme $\mathcal{S} = (\text{Gen}, \text{Sign}, \text{Verify})$ is said to be $(\ell_{vk}, \ell_{msg}, \ell_{sig})$ -bounded length if all verification keys output by $\text{Gen}(1^\lambda)$ have length at most $\ell_{vk}(\lambda)$, Sign takes as input messages of length at most $\ell_{msg}(\lambda)$ and outputs signatures of length bounded by $\ell_{sig}(\lambda)$.

Since the verification keys, messages and signatures have bounded length, we can view Verify as a circuit with three inputs- verification key VK , message m and signature σ . We assume every circuit can be represented as a binary string.

Definition 2.5 ($(\ell_{ver}, \ell_{vk}, \ell_{msg}, \ell_{sig})$ -length qualified signature scheme). Let ℓ_{ver} , ℓ_{vk} , ℓ_{msg} , ℓ_{sig} be fixed polynomials. A $(\ell_{vk}, \ell_{msg}, \ell_{sig})$ -bounded length signature scheme $\mathcal{S} = (\text{Gen}, \text{Sign}, \text{Verify})$ is said to be $(\ell_{ver}, \ell_{vk}, \ell_{msg}, \ell_{sig})$ -length qualified if the verification circuit Verify and signing circuit Sign can be represented as a binary string of length at most $\ell_{ver}(\lambda)$ bits.

Abusing notation, we will say that a tuple $(\text{Verify}, VK, m, \sigma)$ is a $(\ell_{ver}, \ell_{vk}, \ell_{msg}, \ell_{sig})$ -length qualified tuple if Verify is a circuit that can be represented using $\ell_{ver}(\lambda)$ bits, and VK, m, σ are of length at most $\ell_{vk}(\lambda)$, $\ell_{msg}(\lambda)$ and $\ell_{sig}(\lambda)$ respectively. Similarly, a tuple (Verify, VK, m) is $(\ell_{ver}, \ell_{vk}, \ell_{msg})$ -length qualified if Verify, VK and m have length at most $\ell_{ver}(\lambda)$, $\ell_{vk}(\lambda)$ and $\ell_{vk}(\lambda)$ respectively.

2.4 Additively Homomorphic Encryption

In this work, we will be using encryption schemes which allow us to perform additive operations on ciphertexts. Many encryption schemes [GM84, Ben87, NS98, OU98, Pai99, DJ03] have the ‘additive homomorphism’ property. We will now define the syntax and security definition for an additively homomorphic encryption scheme.

Let p be a prime⁹. An additively homomorphic encryption scheme $\mathcal{HE}$ with message space $\mathbb{F}_p$ and ciphertext space $\mathcal{C}_{\text{HE}}$ consists of the following algorithms.

⁸Also known as invariant signature schemes.

⁹The prime p is a property of the encryption scheme.

- $\text{HE.setup}(1^\lambda)$ The setup algorithm takes the security parameter as input and outputs public key pk , secret key sk .
- $\text{HE.enc}(\text{pk}, m)$ The encryption algorithm takes as input a public key pk and message $m \in \mathbb{F}_p$ and outputs a ciphertext $\text{ct} \in \mathcal{C}_{\text{HE}}$.
- $\text{HE.dec}(\text{sk}, \text{ct})$ The decryption algorithm takes as input a secret key sk , a ciphertext $\text{ct} \in \mathcal{C}_{\text{HE}}$ and either outputs an element in $\mathbb{F}_p$ or $\perp$.
- $\text{HE.add}(\text{pk}, \text{ct}_1, \text{ct}_2)$ The addition algorithm takes as input a public key pk and two ciphertexts $\text{ct}_1, \text{ct}_2 \in \mathcal{C}_{\text{HE}}$ and outputs a ciphertext ct .

For simplicity of notation, we will represent $\text{HE.add}(\text{pk}, \text{ct}_1, \text{ct}_2)$ as $\text{ct}_1 + \text{ct}_2$.

Correctness We require the following correctness property:

- Let p be any prime and q any polynomial in λ . For any $\lambda \in \mathbb{N}$, $(\text{pk}, \text{sk}) \leftarrow \text{HE.setup}(1^\lambda)$, q messages $m_1, \dots, m_q \in \mathbb{F}_p$, the following must hold.

$$\text{HE.dec}(\text{sk}, \text{HE.enc}(m_1) + \dots + \text{HE.enc}(m_q)) = m_1 + \dots + m_q.$$

Note that given an encryption ct of message $m \in \mathbb{F}_p$, and a plaintext $a \in \mathbb{F}_p$, one can use HE.add to compute an encryption of $m \cdot a$ efficiently. Let $a \cdot \text{ct}$ represent this operation.

Security The security game is the usual IND-CPA security game between a challenger and a PPT adversary Att .

1. The challenger chooses $(\text{pk}, \text{sk}) \leftarrow \text{HE.setup}(1^\lambda)$ and sends pk to Att .
2. Att sends messages $m_0, m_1 \in \mathbb{F}_p$ to the challenger.
3. The challenger chooses $b \leftarrow \{0, 1\}$, computes $\text{ct}_b \leftarrow \text{HE.enc}(\text{pk}, m_b)$ and sends ct_b to Att .
4. Att finally outputs a guess b' .

Att wins if $b = b'$. Let $\text{Adv}_{\text{Att}}^{\mathcal{HE}} = \Pr[\text{Att wins}] - 1/2$.

Definition 2.6. An additively homomorphic encryption scheme $\mathcal{HE}$ is secure if for all PPT adversaries Att , $\text{Adv}_{\text{Att}}^{\mathcal{HE}}$ is negligible in λ .

2.5 Obfuscation

We recall the definition of indistinguishability obfuscation from [GGH⁺13, SW14].

Definition 2.7. (Indistinguishability Obfuscation) Let $\mathcal{C} = \{\mathcal{C}_\lambda\}_{\lambda \in \mathbb{N}}$ be a family of polynomial-size circuits. Let $i\mathcal{O}$ be a uniform PPT algorithm that takes as input the security parameter λ , a circuit $C \in \mathcal{C}_\lambda$ and outputs a circuit C' . $i\mathcal{O}$ is called an indistinguishability obfuscator for a circuit class $\{\mathcal{C}_\lambda\}$ if it satisfies the following conditions:

- (Preserving Functionality) For all security parameters $\lambda \in \mathbb{N}$, for all $C \in \mathcal{C}_\lambda$, for all inputs x , we have that $C'(x) = C(x)$ where $C' \leftarrow i\mathcal{O}(1^\lambda, C)$.
- (Indistinguishability of Obfuscation) For any (not necessarily uniform) PPT distinguisher $\mathcal{B} = (\text{Samp}, \mathcal{D})$, there exists a negligible function $\text{negl}(\cdot)$ such that the following holds: if for all security parameters $\lambda \in \mathbb{N}$, $\Pr[\forall x, C_0(x) = C_1(x) : (C_0; C_1; \sigma) \leftarrow \text{Samp}(1^\lambda)] > 1 - \text{negl}(\lambda)$, then

$$\begin{aligned} & |\Pr[\mathcal{D}(\sigma, i\mathcal{O}(1^\lambda, C_0)) = 1 : (C_0; C_1; \sigma) \leftarrow \text{Samp}(1^\lambda)] - \\ & \Pr[\mathcal{D}(\sigma, i\mathcal{O}(1^\lambda, C_1)) = 1 : (C_0; C_1; \sigma) \leftarrow \text{Samp}(1^\lambda)]| \leq \text{negl}(\lambda). \end{aligned}$$

In a recent work, [GGH⁺13] showed how indistinguishability obfuscators can be constructed for the circuit class $P/poly$. We remark that $(Samp, \mathcal{D})$ are two algorithms that pass state, which can be viewed equivalently as a single stateful algorithm $\mathcal{B}$. In our proofs we employ the latter approach, although here we state the definition as it appears in prior work.

A stronger notion of obfuscation, *virtual black box obfuscation* was proposed by Barak et al. [BGI⁺12].

Definition 2.8 (Virtual Black-Box Obfuscator). Let $\mathcal{C} = \{\mathcal{C}_\lambda\}_{\lambda \in \mathbb{N}}$ be a family of polynomial-size circuits. Let $\mathcal{O}$ be a PPT algorithm that takes as input the security parameter λ , a circuit $C \in \mathcal{C}_\lambda$ and outputs a circuit C' . $\mathcal{O}$ is called a virtual black-box obfuscator for a circuit class $\{\mathcal{C}_\lambda\}_{\lambda \in \mathbb{N}}$ if it satisfies the following conditions:

- (Preserving Functionality) For all security parameters $\lambda \in \mathbb{N}$, for all $C \in \mathcal{C}_\lambda$, for all inputs x , we have that $C'(x) = C(x)$ where $C' \leftarrow \mathcal{O}(1^\lambda, C)$.
- (Virtual Black-Box) For every (non-uniform) PPT algorithm $\mathcal{A}$, there exists a PPT simulator S such that, for all $C \in \mathcal{C}_\lambda$,

$$\Pr[\mathcal{A}(\mathcal{O}(1^\lambda, C)) = 1] - \Pr[S^C(1^\lambda, 1^{|C|}) = 1] \leq \text{negl}(\lambda)$$

For simplicity of notation, we will drop the dependence of $i\mathcal{O}$ and $\mathcal{O}$ on 1^λ .

2.6 Puncturable Pseudorandom Functions

The notion of constrained PRFs was introduced in the concurrent works of [BW13, BGI13, KPTZ13]. Punctured PRFs, first termed by [SW14] are a special class of constrained PRFs.

A PRF $F : \mathcal{K} \times \mathcal{X} \rightarrow \mathcal{Y}$ is a puncturable pseudorandom function if there is an additional key space $\mathcal{K}_p$ and three polynomial time algorithms $F.\text{setup}$, $F.\text{eval}$ and $F.\text{puncture}$ as follows:

- $F.\text{setup}(1^\lambda)$ is a randomized algorithm that takes the security parameter λ as input and outputs a description of the key space $\mathcal{K}$, the punctured key space $\mathcal{K}_p$ and the PRF F .
- $F.\text{puncture}(K, x)$ is a randomized algorithm that takes as input a PRF key $K \in \mathcal{K}$ and $x \in \mathcal{X}$, and outputs a key $K_x \in \mathcal{K}_p$.
- $F.\text{eval}(K_x, x')$ is a deterministic algorithm that takes as input a punctured key $K_x \in \mathcal{K}_p$ and $x' \in \mathcal{X}$. Let $K \in \mathcal{K}$, $x \in \mathcal{X}$ and $K_x \leftarrow F.\text{puncture}(K, x)$. For correctness, we need the following property:

$$F.\text{eval}(K_x, x') = \begin{cases} F(K, x') & \text{if } x \neq x' \\ \perp & \text{otherwise} \end{cases}$$

In this work, we will only need selectively secure puncturable PRFs. The selective security game between the challenger and the adversary A consists of the following phases.

Challenge Phase A sends a challenge $x^* \in \mathcal{X}$. The challenger chooses uniformly at random a PRF key $K \leftarrow \mathcal{K}$ and a bit $b \leftarrow \{0, 1\}$. It computes $K\{x^*\} \leftarrow F.\text{puncture}(K, x^*)$. If $b = 0$, the challenger sets $y = F(K, x^*)$, else $y \leftarrow \mathcal{Y}$. It sends $K\{x^*\}, y$ to A .

Guess A outputs a guess b' of b .

A wins if $b = b'$. The advantage of A is defined to be $\text{Adv}_A^F(\lambda) = \Pr[A \text{ wins}]$.

Definition 2.9. The PRF F is a selectively secure puncturable PRF if for all probabilistic polynomial time adversaries A , $\text{Adv}_A^F(\lambda)$ is negligible in λ .

2.7 Universal Parameters

In a recent work, Hofheinz et al. [HJK⁺14] introduced the notion of universal parameters. A universal parameters scheme $\mathcal{U}$, parameterized by polynomials ℓ_{ckt} , ℓ_{inp} and ℓ_{out} , consists of algorithms **UniversalGen** and **InduceGen** defined below.

- **UniversalGen**(1^λ) takes as input the security parameter λ and outputs the universal parameters U .
- **InduceGen**(U, d) takes as input the universal parameters U and a circuit d of size at most ℓ_{ckt} bits. The circuit d takes as input ℓ_{inp} bits and outputs ℓ_{out} bits.

In this work, we will be using a universal parameter scheme that is adaptively secure in the random oracle model. In order to define adaptive security for universal parameters, let us first define the notion of an admissible adversary $\mathcal{A}$.

An admissible adversary $\mathcal{A}$ is defined to be an efficient interactive Turing Machine that outputs one bit, with the following input/output behavior:

- $\mathcal{A}$ takes as input security parameter λ and a universal parameter U .
- $\mathcal{A}$ can send a random oracle query (RO, x) , and receives the output of the random oracle on input x .
- $\mathcal{A}$ can send a message of the form (params, d) where $d \in \mathcal{C}[\ell_{\text{ckt}}, \ell_{\text{inp}}, \ell_{\text{out}}]$. Upon sending this message, $\mathcal{A}$ is required to honestly compute $p_d = \text{InduceGen}(U, d)$, making use of any additional random oracle queries, and $\mathcal{A}$ appends (d, p_d) to an auxiliary tape.

Let **SimUGen** and **SimRO** be PPT algorithms. Consider the following two experiments:

$\text{Real}^{\mathcal{A}}(1^\lambda)$:

1. The random oracle **RO** is implemented by assigning random outputs to each unique query made to **RO**.
2. $U \leftarrow \text{UniversalGen}^{\text{RO}}(1^\lambda)$.
3. $\mathcal{A}(1^\lambda, U)$ is executed, where every message of the form (RO, x) receives the response $\text{RO}(x)$.
4. Upon termination of $\mathcal{A}$, the output of the experiment is the final output of the execution of $\mathcal{A}$.

$\text{Ideal}_{\text{SimUGen}, \text{SimRO}}^{\mathcal{A}}(1^\lambda)$:

1. A truly random function F that maps ℓ_{ckt} bits to ℓ_{out} bits is implemented by assigning random ℓ_{out} -bit outputs to each unique query made to F . Throughout this experiment, a Parameters Oracle O is implemented as follows: On input d , where $d \in \mathcal{C}[\ell_{\text{ckt}}, \ell_{\text{inp}}, \ell_{\text{out}}]$, O outputs $d(F(d))$.
2. $(U, \tau) \leftarrow \text{SimUGen}(1^\lambda)$. Here, **SimUGen** can make arbitrary queries to the Parameters Oracle O .
3. $\mathcal{A}(1^\lambda, U)$ and **SimRO**(τ) begin simultaneous execution.
 - Whenever $\mathcal{A}$ sends a message of the form (RO, x) , this is forwarded to **SimRO**, which produces a response to be sent back to $\mathcal{A}$.
 - **SimRO** can make any number of queries to the Parameter Oracle O .
 - Finally, after $\mathcal{A}$ sends any message of the form (params, d) , the auxiliary tape of $\mathcal{A}$ is examined until an entry of the form (d, p_d) is added to it. At this point, if p_d is not equal to $d(F(d))$, then experiment aborts, resulting in an *Honest Parameter Violation*.
4. Upon termination of $\mathcal{A}$, the output of the experiment is the final output of the execution of $\mathcal{A}$.

Definition 2.10. A universal parameters scheme $\mathcal{U} = (\text{UniversalGen}, \text{InduceGen})$, parameterized by polynomials ℓ_{ckt} , ℓ_{inp} and ℓ_{out} , is said to be adaptively secure in the random oracle model if there exist PPT algorithms **SimUGen** and **SimRO** such that for all PPT adversaries $\mathcal{A}$, the following hold:

$$\Pr[\text{Ideal}_{\text{SimUGen}, \text{SimRO}}^{\mathcal{A}}(1^\lambda) \text{ aborts}] = 0^{10}$$

and

$$|\Pr[\text{Real}^{\mathcal{A}}(1^\lambda) = 1] - \Pr[\text{Ideal}_{\text{SimUGen}, \text{SimRO}}^{\mathcal{A}}(1^\lambda) = 1]| \leq \text{negl}(\lambda)$$

¹⁰The definition in [HJK⁺14] only requires this probability to be negligible in λ . However, the construction actually achieves zero probability of Honest Parameter Violation. Hence, for the simplicity of our proof, we will use this definition.

Hofheinz et al. [HJK⁺14] construct a universal parameters scheme that is adaptively secure in the random oracle model, assuming a secure indistinguishability obfuscator, a selectively secure puncturable PRF and an injective one way function.

2.8 RSA Assumption

Assumption 1 (RSA [RSA78]). Let λ be the security parameter. Let $N = pq$ be the RSA modulus, where p, q are randomly chosen, distinct, λ -bit primes. Let e be a randomly chosen positive integer less than and relatively prime to $\phi(N) = (p-1)(q-1)$ and $y \leftarrow \mathbb{Z}_N$. For any PPT algorithm $\mathcal{A}$, $\Pr[x \leftarrow \mathcal{A}(N, e, y) \text{ and } x^e = y] \leq \text{negl}(\lambda)$.

3 Universal Signature Aggregators

In this section, we define the notion of universal signature aggregators. Let $\ell_{\text{ver}}, \ell_{\text{vk}}, \ell_{\text{msg}}, \ell_{\text{sig}}$ be polynomials. Given any security parameter λ , $\ell_{\text{ver}}(\lambda)$ represents a bound on the size of verification circuits, $\ell_{\text{vk}}(\lambda)$ represents a bound on the size of verification key, $\ell_{\text{msg}}(\lambda)$ is a bound on the length of messages signed and $\ell_{\text{sig}}(\lambda)$ is a bound on the size of signatures. For simplicity of notation, we will drop the dependence on λ when the context is clear.

A universal signature aggregator $(\ell_{\text{ver}}, \ell_{\text{vk}}, \ell_{\text{msg}}, \ell_{\text{sig}})$ -UniversalSigAgg consists of three algorithms UniversalSetup, UniversalAgg and UniversalVerify defined as follows.

- **UniversalSetup**(1^λ) is a randomized algorithm that takes as input security parameter λ and outputs public parameters PP.
- **UniversalAgg**(PP, $\{(\text{Verify}_i, \text{VK}_i, m_i, \sigma_i)\}_{i=1}^t$) is a deterministic algorithm that takes as input security parameter λ , public parameters PP and t tuples $(\text{Verify}_i, \text{VK}_i, m_i, \sigma_i)$ (for some arbitrary t) where each tuple is $(\ell_{\text{ver}}, \ell_{\text{vk}}, \ell_{\text{msg}}, \ell_{\text{sig}})$ -length qualified. It outputs an aggregate signature σ_{agg} whose length is polynomial in λ , but independent of t .
- **UniversalVerify**(PP, $\{(\text{Verify}_i, \text{VK}_i, m_i)\}_{i=1}^t, \sigma_{\text{agg}}$) is a deterministic algorithm that takes as input security parameter λ , public parameters PP, t tuples $(\text{Verify}_i, \text{VK}_i, m_i)$ that are $(\ell_{\text{ver}}, \ell_{\text{vk}}, \ell_{\text{msg}})$ -length qualified, and an aggregated signature σ_{agg} . It outputs either 0 or 1.

For our constructions, we will assume that all verification circuits have ℓ_{ver} bit representation, all verification keys have length ℓ_{vk} , all messages signed have length ℓ_{msg} and the corresponding signatures have length ℓ_{sig} .

Correctness Let $\{(\text{Verify}_i, \text{VK}_i, m_i, \sigma_i)\}_{i=1}^t$ be any t distinct tuples that are $(\ell_{\text{ver}}, \ell_{\text{vk}}, \ell_{\text{msg}}, \ell_{\text{sig}})$ -length qualified and for all $i \leq t$, $\text{Verify}_i(\text{VK}_i, m_i, \sigma_i) = 1$. For all $\lambda \in \mathbb{N}$, $\text{PP} \leftarrow \text{UniversalSetup}(1^\lambda)$ and $\sigma_{\text{agg}} \leftarrow \text{UniversalAgg}(1^\lambda, \text{PP}, \{(\text{Verify}_i, \text{VK}_i, m_i, \sigma_i)\}_i)$, we require that $\text{UniversalVerify}(\text{PP}, \{(\text{Verify}_i, \text{VK}_i, m_i)\}_i, \sigma_{\text{agg}}) = 1$.

3.1 Security of Universal Signature Aggregators

We now proceed to the formal security definition for universal signature aggregators.

Let $\mathcal{S} = (\mathcal{S}.\text{Gen}, \mathcal{S}.\text{Sign}, \mathcal{S}.\text{Verify})$ be a secure $(\ell_{\text{ver}}, \ell_{\text{vk}}, \ell_{\text{msg}}, \ell_{\text{sig}})$ -length qualified signature scheme. Consider the following security game between an adversary $\mathcal{A}$ and the challenger.

$\text{Exp}_{\mathcal{A}, \mathcal{S}}(\lambda)$:

- **Setup Phase** Challenger chooses $(\text{SK}, \text{VK}) \leftarrow \mathcal{S}.\text{Gen}(1^\lambda)$, computes $\text{PP} \leftarrow \text{UniversalSetup}(1^\lambda)$ and sends PP, VK to $\mathcal{A}$.

- **Signing Phase** $\mathcal{A}$ sends signing query x_i , and receives $\sigma_i \leftarrow \mathcal{S}.\text{Sign}(\text{SK}, x_i)$.
- **Forgery** $\mathcal{A}$ finally outputs t tuples $(\text{Verify}_i, \text{VK}_i, m_i)$ and an aggregated forgery σ_{agg} .

$\mathcal{A}$ wins if there exists $i^* \in [t]$ such that $\text{Verify}_{i^*} = \mathcal{S}.\text{Verify}$, $\text{VK}_{i^*} = \text{VK}$, message m_{i^*} was not queried during the signing phase and $\text{UniversalVerify}(\text{PP}, \{(\text{Verify}_i, \text{VK}_i, m_i)\}, \sigma_{\text{agg}}) = 1$. Let $\text{Adv}_{\mathcal{A}, \mathcal{S}}(\lambda) = \Pr[\mathcal{A} \text{ wins } \text{Exp}_{\mathcal{A}, \mathcal{S}}(\lambda)]$.

Definition 3.1. Let $\mathcal{S}$ be a $(\ell_{\text{ver}}, \ell_{\text{vk}}, \ell_{\text{msg}}, \ell_{\text{sig}})$ -length qualified secure signature scheme. A universal signature aggregator scheme $(\ell_{\text{ver}}, \ell_{\text{vk}}, \ell_{\text{msg}}, \ell_{\text{sig}})$ -**UniversalSigAgg** is secure with respect to scheme $\mathcal{S}$ if for all PPT adversaries $\mathcal{A}$, $\text{Adv}_{\mathcal{A}, \mathcal{S}}(\lambda)$ is negligible in λ .

We can also define a weaker *selective* notion where the adversary $\mathcal{A}$ chooses the message m corresponding to $(\mathcal{S}.\text{Verify}, \text{VK})$ before receiving the public parameters PP . More formally, the selective experiment $\text{Exp}_{\mathcal{A}, \mathcal{S}}^{\text{sel}}(\lambda)$ is defined as follows.

$\text{Exp}_{\mathcal{A}, \mathcal{S}}^{\text{sel}}(\lambda)$:

- $\mathcal{A}$ sends a message m to the challenger.
- **Setup Phase** Challenger computes $(\text{SK}, \text{VK}) \leftarrow \mathcal{S}.\text{Gen}(1^\lambda)$, $\text{PP} \leftarrow \text{UniversalSetup}(1^\lambda)$ and sends PP, VK to $\mathcal{A}$.
- **Signing Phase** $\mathcal{A}$ sends signing query $x_i \neq m$, and receives $\sigma_i \leftarrow \mathcal{S}.\text{Sign}(\text{SK}, x_i)$.
- **Forgery** $\mathcal{A}$ finally outputs t tuples $(\text{Verify}_i, \text{VK}_i, m_i)$ and an aggregated forgery σ_{agg} .

$\mathcal{A}$ wins if there exists an $i^* \in [t]$ such that $\text{Verify}_{i^*} = \mathcal{S}.\text{Verify}$, $\text{VK}_{i^*} = \text{VK}$, $m_{i^*} = m$ and $\text{UniversalVerify}(\text{PP}, \{(\text{Verify}_i, \text{VK}_i, m_i)\}, \sigma_{\text{agg}}) = 1$. Let $\text{Adv}_{\mathcal{A}, \mathcal{S}}^{\text{sel}}(\lambda) = \Pr[\mathcal{A} \text{ wins } \text{Exp}_{\mathcal{A}, \mathcal{S}}^{\text{sel}}(\lambda)]$.

Definition 3.2. Let $\mathcal{S}$ be a $(\ell_{\text{ver}}, \ell_{\text{vk}}, \ell_{\text{msg}}, \ell_{\text{sig}})$ -length qualified secure signature scheme. A universal signature aggregator scheme $(\ell_{\text{ver}}, \ell_{\text{vk}}, \ell_{\text{msg}}, \ell_{\text{sig}})$ -**UniversalSigAgg** is *selectively* secure with respect to scheme $\mathcal{S}$ if for all PPT adversaries $\mathcal{A}$, $\text{Adv}_{\mathcal{A}, \mathcal{S}}^{\text{sel}}(\lambda)$ is negligible in λ .

In certain situations, it may be possible that the number of signatures to be aggregated is known in advance. In such a scenario, we can use bounded universal signature aggregators (defined below).

Definition 3.3. An n -bounded universal signature aggregator scheme $(\ell_{\text{ver}}, \ell_{\text{vk}}, \ell_{\text{msg}}, \ell_{\text{sig}})$ -**UniversalSigAgg** = $(\text{UniversalSetup}, \text{UniversalAgg}, \text{UniversalVerify})$ is a universal signature aggregator in which **UniversalSetup** takes an additional input 1^n . The public parameters output by **UniversalSetup** have size bounded by some polynomial in λ and n . However, the aggregated signature has size bounded by a polynomial in λ , but is independent of n .

4 Universally Aggregating Unique Signatures

We will now describe our universal signature aggregator $(\ell_{\text{ver}}, \ell_{\text{vk}}, \ell_{\text{msg}}, \ell_{\text{sig}})$ -**UniversalSigAgg**. Let $i\mathcal{O}$ be a secure indistinguishability obfuscation scheme, F a puncturable PRF with key space $\mathcal{K}$, punctured key space $\mathcal{K}_p$, domain $\mathcal{X} = \{0, 1\}^{\ell_{\text{ver}}} \times \{0, 1\}^{\ell_{\text{vk}}} \times \{0, 1\}^{\ell_{\text{msg}}}$ and range $\mathcal{Y} = \mathbb{Z}_N^*$ for some randomly chosen RSA modulus N , and algorithms $F.\text{setup}$, $F.\text{puncture}$, $F.\text{eval}$. Our scheme consists of the three algorithms **UniversalSetup**, **UniversalAgg** and **UniversalVerify**.

UniversalSetup(1^λ) **UniversalSetup** first chooses an RSA modulus N and $e \leftarrow \mathbb{Z}_{\phi(N)}^*$. Next, it chooses a PRF key $K \leftarrow F.\text{setup}(1^\lambda)$ and computes obfuscations of the programs $\text{Transform}_{N,K}$ ¹¹ and $\text{Transform-Image}_{N,K,e}$ ¹² defined below. It sets the public parameters to be $\text{PP} = (i\mathcal{O}(\text{Transform}_{N,K}), i\mathcal{O}(\text{Transform-Image}_{N,K,e}), N, e)$.

Transform $_{N,K}$:

Inputs: $\text{Verify}' \in \{0, 1\}^{\ell_{\text{ver}}}$, $\text{VK}' \in \{0, 1\}^{\ell_{\text{vk}}}$, $m' \in \{0, 1\}^{\ell_{\text{msg}}}$, $\sigma' \in \{0, 1\}^{\ell_{\text{sig}}}$.
 Constants : RSA modulus $N \in \mathbb{N}$, $K \in \mathcal{K}$.

if $\text{Verify}'(\text{VK}', m', \sigma') = 0$ **then**
 Output $\perp$.
else
 Output $F(K, \text{Verify}' || \text{VK}' || m')$.
end if

Transform-Image $_{N,K,e}$:

Inputs: $\text{Verify}' \in \{0, 1\}^{\ell_{\text{ver}}}$, $\text{VK}' \in \{0, 1\}^{\ell_{\text{vk}}}$, $m' \in \{0, 1\}^{\ell_{\text{msg}}}$.
 Constants : RSA modulus $N \in \mathbb{N}$, $K \in \mathcal{K}$, $e \in \mathbb{Z}_{\phi(N)}$.

Let $w = F(K, \text{Verify}' || \text{VK}' || m')$. Output $w^e \pmod{N}$.

UniversalAgg($\text{PP}, \{(\text{Verify}_i, \text{VK}_i, m_i, \sigma_i)\}_{i=1}^n$): Let $\text{PP} = (P_1, P_2, N, e)$. **UniversalAgg** first checks if the n tuples are distinct. If not, it outputs $\perp$. Else, it computes $t_i = P_1(\text{Verify}_i, \text{VK}_i, m_i, \sigma_i)$ for each $i \leq n$. If $t_i = \perp$ for some i , then **UniversalAgg** outputs $\perp$, else it outputs $\sigma_{\text{agg}} = \prod_i t_i \pmod{N}$.

UniversalVerify($\text{PP}, \{(\text{Verify}_i, \text{VK}_i, m_i)\}_{i=1}^n, \sigma_{\text{agg}}$): Let $\text{PP} = (P_1, P_2, N, e)$. **UniversalVerify** first checks if all n tuples are distinct. If not, it outputs 0. Else, it computes, for all $i \leq n$, $s_i = \text{Transform-Image}(\text{Verify}_i, \text{VK}_i, m_i)$. If $(\prod_i s_i) = \sigma_{\text{agg}}^e \pmod{N}$, it outputs 1, else it outputs 0.

Correctness: Let $\{(\text{Verify}_i, \text{VK}_i, m_i, \sigma_i)\}_{i=1}^n$ be n tuples such that they are all distinct and $\text{Verify}_i(\text{VK}_i, m_i, \sigma_i) = 1$ for all $i \leq n$. Fix any $\lambda \in \mathbb{N}$, $\text{PP} \leftarrow \text{UniversalSetup}(1^\lambda)$, $(\sigma_{\text{agg}}) \leftarrow \text{UniversalAgg}(\text{PP}, \{(\text{Verify}_i, \text{VK}_i, m_i, \sigma_i)\})$. Then,

$$\begin{aligned} \sigma_{\text{agg}}^e &= \left(\prod \text{Transform}(\text{Verify}_i, \text{VK}_i, m_i, \sigma_i) \right)^e \pmod{N} \\ &= \left(\prod F(K, \text{Verify}_i || \text{VK}_i || m_i) \right)^e \pmod{N} \\ &= \left(\prod F(K, \text{Verify}_i || \text{VK}_i || m_i)^e \right) \pmod{N} \\ &= \left(\prod \text{Transform-Image}_{N,K,e}(\text{Verify}_i, \text{VK}_i, m_i) \right) \pmod{N} \end{aligned}$$

Also, note that the size of the aggregated signature ($\sigma_{\text{agg}} \in \mathbb{Z}_N^*$) depends only on the security parameter λ , but not on the number of signatures aggregated.

4.1 Proof of Security

In this subsection, we will show that our construction from Section 4 is selectively secure with respect to secure unique signature schemes.

¹¹Padded to be of the same size as **TransformAlt** and **Transform-Reject**.

¹²Padded to be of the same size as **Transform-Image-1**.

Theorem 4.1. Assuming $i\mathcal{O}$ is a secure indistinguishability obfuscator, F is a selectively secure puncturable PRF and RSA is secure, for all $(\ell_{\text{ver}}, \ell_{\text{vk}}, \ell_{\text{msg}}, \ell_{\text{sig}})$ -length qualified secure unique signature schemes $\mathcal{S}$, the universal signature aggregator $(\ell_{\text{ver}}, \ell_{\text{vk}}, \ell_{\text{msg}}, \ell_{\text{sig}})$ -UniversalSigAgg is selectively secure with respect to $\mathcal{S}$.

Let $\mathcal{S} = (\mathcal{S}.\text{Gen}, \mathcal{S}.\text{Sign}, \mathcal{S}.\text{Verify})$ be a secure $(\ell_{\text{ver}}, \ell_{\text{vk}}, \ell_{\text{msg}}, \ell_{\text{sig}})$ -length qualified unique signature scheme, and Att a PPT adversary. In order to prove this theorem, we will define a sequence of experiments Game 0, ..., Game 3, where Game 0 = $\text{Exp}_{\text{Att}, \mathcal{S}}^{\text{sel}}$.

4.1.1 Sequence of Games

Game 0: This game corresponds to $\text{Exp}_{\text{Att}, \mathcal{S}}^{\text{sel}}$. The adversary Att first selectively sends message m , and then receives the verification key and public parameters for the aggregator. Next, the adversary makes signing queries, and finally submits the forgery.

1. Att sends message m .
2. Compute $(\text{SK}, \text{VK}) \leftarrow \mathcal{S}.\text{Gen}(1^\lambda)$. Choose an RSA modulus N , $e \leftarrow \mathbb{Z}_{\phi(N)}^*$, $K \leftarrow F.\text{setup}(1^\lambda)$ and set $\text{PP} = (i\mathcal{O}(\text{Transform}_{N,K}), i\mathcal{O}(\text{Transform-Image}_{N,K,e}), N, e)$. Send PP, VK to Att.
3. For each signing query $x_i \neq m$, compute $\sigma_i \leftarrow \mathcal{S}.\text{Sign}(\text{SK}, x_i)$ and send σ_i to Att.
4. Att sends forgery σ_{agg} and n tuples $\{(\text{Verify}_i, \text{VK}_i, m_i)\}$. Att wins if $\exists i^* \in [n]$ such that $\text{Verify}_{i^*} = \mathcal{S}.\text{Verify}$, $\text{VK}_{i^*} = \text{VK}$ and $m_{i^*} = m$ and $\text{UniversalVerify}(\text{PP}, \{(\text{Verify}_i, \text{VK}_i, m_i)\}, \sigma_{\text{agg}}) = 1$.

Game 1: This game is exactly similar to the previous one, except that the program Transform is replaced by Transform-Reject¹³ which outputs $\perp$ if the input tuples is $(\mathcal{S}.\text{Verify}, \text{VK}, m, \sigma)$ even if $\mathcal{S}.\text{Verify}(\text{VK}, m, \sigma) = 1$. Also, it uses a PRF key punctured at $y = \mathcal{S}.\text{Verify}||\text{VK}||m$.

1. Att sends message m .
2. Compute $(\text{SK}, \text{VK}) \leftarrow \mathcal{S}.\text{Gen}(1^\lambda)$. Choose an RSA modulus N , $e \leftarrow \mathbb{Z}_{\phi(N)}^*$, $K \leftarrow F.\text{setup}(1^\lambda)$. Set $y = \mathcal{S}.\text{Verify}||\text{VK}||m$, compute $K\{y\} \leftarrow F.\text{puncture}(K, y)$ and $\text{PP} = (i\mathcal{O}(\text{Transform-Reject}_{y,N,K\{y\}}), i\mathcal{O}(\text{Transform-Image}_{N,K,e}), N, e)$. Send PP, VK to Att.
3. For each signing query $x_i \neq m$, compute $\sigma_i \leftarrow \mathcal{S}.\text{Sign}(\text{SK}, x_i)$ and send σ_i to Att.
4. Att sends forgery σ_{agg} and n tuples $\{(\text{Verify}_i, \text{VK}_i, m_i)\}$. Att wins if $\exists i^* \in [n]$ such that $\text{Verify}_{i^*} = \mathcal{S}.\text{Verify}$, $\text{VK}_{i^*} = \text{VK}$ and $m_{i^*} = m$ and $\text{UniversalVerify}(\text{PP}, \{(\text{Verify}_i, \text{VK}_i, m_i)\}, \sigma_{\text{agg}}) = 1$.

Transform-Reject_{y,N,K{y}} :

Inputs: $\text{Verify}' \in \{0, 1\}^{\ell_{\text{ver}}}$, $\text{VK}' \in \{0, 1\}^{\ell_{\text{vk}}}$, $m' \in \{0, 1\}^{\ell_{\text{msg}}}$, $\sigma' \in \{0, 1\}^{\ell_{\text{sig}}}$.

Constants : $y \in \{0, 1\}^{\ell_{\text{ver}}} \times \{0, 1\}^{\ell_{\text{vk}}} \times \{0, 1\}^{\ell_{\text{msg}}}$, RSA modulus $N \in \mathbb{N}$, $K\{y\} \in \mathcal{K}_p$.

```

if  $\text{Verify}'(\text{VK}', m', \sigma') = 0$  then
  Output  $\perp$ .
else if  $\text{Verify}'||\text{VK}'||m' = y$  then
  Output  $\perp$ .
else
  Output  $F.\text{eval}(K\{y\}, \text{Verify}'||\text{VK}'||m')$ .
end if

```

Game 2: This game is similar to the previous one, except that the program Transform-Image is replaced by Transform-Image-1¹⁴. It uses a PRF key punctured at $y = \mathcal{S}.\text{Verify}||\text{VK}||m$. For input y , it outputs a hardcoded constant z . In this game, z is set to be $F(K, y)^e$.

¹³Padded appropriately to be of the same size as Transform and TransformAlt.

¹⁴Padded appropriately to be of the same size as Transform-Image.

1. Att sends message m .
2. Compute $(SK, VK) \leftarrow \mathcal{S}.\text{Gen}(1^\lambda)$. Choose an RSA modulus N , $e \leftarrow \mathbb{Z}_{\phi(N)}^*$ and $K \leftarrow F.\text{setup}(1^\lambda)$.
Set $y = \mathcal{S}.\text{Verify}||VK||m$. Compute $K\{y\} \leftarrow F.\text{puncture}(K, y)$, $w = F(K, y)$ and $z = w^e \pmod{N}$.
Set $PP = (i\mathcal{O}(\text{Transform-Reject}_{y,N,K\{y\}}), i\mathcal{O}(\text{Transform-Image-1}_{y,N,K\{y\},z,e}), N, e)$ and send PP , VK to Att.
3. For each signing query $x_i \neq m$, compute $\sigma_i \leftarrow \mathcal{S}.\text{Sign}(SK, x_i)$ and send σ_i to Att.
4. Att sends forgery σ_{agg} and n tuples $\{(\text{Verify}_i, VK_i, m_i)\}$. Att wins if $\exists i^* \in [n]$ such that $\text{Verify}_{i^*} = \mathcal{S}.\text{Verify}$, $VK_{i^*} = VK$ and $m_{i^*} = m$ and $\text{UniversalVerify}(PP, \{(\text{Verify}_i, VK_i, m_i)\}, \sigma_{\text{agg}}) = 1$.

Transform-Image-1_{y,N,K{y},z,e} :

Inputs: $\text{Verify}' \in \{0, 1\}^{\ell_{\text{ver}}}$, $VK' \in \{0, 1\}^{\ell_{\text{vk}}}$, $m' \in \{0, 1\}^{\ell_{\text{msg}}}$.

Constants: $y \in \{0, 1\}^{\ell_{\text{ver}}} \times \{0, 1\}^{\ell_{\text{vk}}} \times \{0, 1\}^{\ell_{\text{msg}}}$, RSA modulus $N \in \mathbb{N}$,
 $K\{y\} \in \mathcal{K}_p$, $z \in \mathbb{Z}_N^*$, $e \in \mathbb{Z}_{\phi(N)}^*$.

if $\text{Verify}_i||VK_i||m_i = y$ **then**

Output z .

else

Let $w = F.\text{eval}(K\{y\}, \text{Verify}'||VK'||m')$.

Output w^e .

end if

Game 3: In this game, the challenger chooses z at random.

1. Att sends message m .
2. Compute $(SK, VK) \leftarrow \mathcal{S}.\text{Gen}(1^\lambda)$. Choose an RSA modulus N , $e \leftarrow \mathbb{Z}_{\phi(N)}^*$ and $K \leftarrow F.\text{setup}(1^\lambda)$.
Set $y = \mathcal{S}.\text{Verify}||VK||m$. Compute $K\{y\} \leftarrow F.\text{puncture}(K, y)$ and $z \leftarrow \mathbb{Z}_N^*$.
Set $PP = (i\mathcal{O}(\text{Transform-Reject}_{y,N,K\{y\}}), i\mathcal{O}(\text{Transform-Image-1}_{y,N,K\{y\},z,e}), N, e)$ and send PP , VK to Att.
3. For each signing query $x_i \neq m$, compute $\sigma_i \leftarrow \mathcal{S}.\text{Sign}(SK, x_i)$ and send σ_i to Att.
4. Att sends forgery σ_{agg} and n tuples $\{(\text{Verify}_i, VK_i, m_i)\}$. Att wins if all the n tuples are distinct and $\exists i^* \in [n]$ such that $\text{Verify}_{i^*} = \mathcal{S}.\text{Verify}$, $VK_{i^*} = VK$ and $m_{i^*} = m$ and $\text{UniversalVerify}(PP, \{(\text{Verify}_i, VK_i, m_i)\}, \sigma_{\text{agg}}) = 1$.

4.1.2 Analysis

Let $\text{Adv}_{\text{Att}}^j$ denote the advantage of adversary Att in Game j .

Lemma 4.1. Assuming $i\mathcal{O}$ is a secure indistinguishability obfuscator and $\mathcal{S}$ is a secure $(\ell_{\text{ver}}, \ell_{\text{vk}}, \ell_{\text{msg}}, \ell_{\text{sig}})$ -length qualified unique signature scheme, for any PPT adversary Att,

$$\text{Adv}_{\text{Att}}^0 - \text{Adv}_{\text{Att}}^1 \leq \text{negl}(\lambda).$$

Proof. Suppose, on the contrary, there exists a PPT adversary Att such that $\text{Adv}_{\text{Att}}^0 - \text{Adv}_{\text{Att}}^1 = \epsilon$, where ϵ is non-negligible in λ . Assuming $\mathcal{O}$ is a secure indistinguishability obfuscator, we will use Att to construct a PPT algorithm $\mathcal{B}$ that breaks the security of $\mathcal{S}$. Here, we will crucially use the fact that $\mathcal{S}$ is a unique signature scheme; that is, there is a unique accepting signature $\sigma \in \{0, 1\}^{\ell_{\text{sig}}}$ corresponding to verification key VK and message m .

First, let us consider the following altered circuit $\text{TransformAlt}_{j,b,y,N,K}$ ¹⁵ which takes as input a tuple $(\text{Verify}', VK', m', \sigma')$ and outputs $\perp$ if $\text{Verify}' = \mathcal{S}.\text{Verify}$, $VK' = VK$ and the j^{th} bit of σ' is b .

¹⁵Padded appropriately to be of the same size as Transform and Transform-Reject.

TransformAlt _{j,b,y,N,K} :

Inputs: $\text{Verify}' \in \{0,1\}^{\ell_{\text{ver}}}$, $\text{VK}' \in \{0,1\}^{\ell_{\text{vk}}}$, $m' \in \{0,1\}^{\ell_{\text{msg}}}$, $\sigma' \in \{0,1\}^{\ell_{\text{sig}}}$.

Constants : $j \in [\ell_{\text{sig}}]$, $b \in \{0,1\}$, $y \in \{0,1\}^{\ell_{\text{ver}}} \times \{0,1\}^{\ell_{\text{vk}}} \times \{0,1\}^{\ell_{\text{msg}}}$, RSA modulus $N \in \mathbb{N}$, $K \in \mathcal{K}$.

```

if  $\text{Verify}'(\text{VK}', m', \sigma') = 0$  then
  Output  $\perp$ .
else if  $\text{Verify}'||\text{VK}'||m' = y$  and  $\sigma'[j] = b$  then
  Output  $\perp$ .
else
  Output  $F(K, \text{Verify}'||\text{VK}'||m')$ .
end if

```

We will now state two observations which will be useful for proving our claim. Fix a message $m \in \{0,1\}^{\ell_{\text{msg}}}$. Let $(\text{SK}, \text{VK}) \leftarrow \mathcal{S}.\text{Setup}(1^\lambda)$, $\sigma \leftarrow \mathcal{S}.\text{Sign}(\text{SK}, m)$ and $y = \mathcal{S}.\text{Verify}||\text{VK}||m$. Let $\sigma[j]$ denote the j^{th} bit of σ .

Observation 4.1. For all $j \in [\ell_{\text{sig}}]$, the circuits $\text{Transform}_{N,K}$ and $\text{TransformAlt}_{j,1-\sigma[j],y,N,K}$ are functionally identical.

Observation 4.2. For all $j \in [\ell_{\text{sig}}]$, the circuits $\text{Transform-Reject}_{y,N,K\{y\}}$ and $\text{TransformAlt}_{j,\sigma[j],y,N,K}$ are functionally identical.

Both these observations follow from the fact that $\mathcal{S}$ is a unique signature scheme and the correctness of punctured key on non-punctured inputs.

Next, we define **Game-Alt** j,b , which is exactly similar to **Game 0** and **Game 1**, except that the challenger outputs $\mathcal{O}(1^\lambda, \text{TransformAlt}_{j,b,y,N,K})$ (instead of $\mathcal{O}(1^\lambda, \text{Transform}_{N,K})$ or $\mathcal{O}(1^\lambda, \text{Transform-Reject}_{y,N,K\{y\}})$) as part of the public parameters PP. Let $\mathcal{E}_j^\sigma$ be the event that $\sigma[j] = 1$, where σ is the unique signature corresponding to challenge message m output by Att.

From these observations, it follows that $\mathcal{O}(1^\lambda, \text{Transform}_{N,K})$ and $\mathcal{O}(1^\lambda, \text{TransformAlt}_{j,1-\sigma[j],y,N,K})$ are computationally indistinguishable (by the security of $\mathcal{O}$) and similarly, $\mathcal{O}(1^\lambda, \text{TransformAlt}_{j,\sigma[j],y,N,K})$ and $\mathcal{O}(1^\lambda, \text{Transform-Reject}_{y,N,K\{y\}})$ are computationally indistinguishable. Hence, for all $j \leq \ell_{\text{sig}}$, we get the following equations:

$$|\Pr[(\text{Att wins in Game 0}) | \mathcal{E}_j^\sigma] - \Pr[(\text{Att wins in Game-Alt } j, 0) | \mathcal{E}_j^\sigma]| \leq \text{negl}(\lambda), \quad (1)$$

$$|\Pr[(\text{Att wins in Game 0}) | \neg \mathcal{E}_j^\sigma] - \Pr[(\text{Att wins in Game-Alt } j, 1) | \neg \mathcal{E}_j^\sigma]| \leq \text{negl}(\lambda), \quad (2)$$

$$|\Pr[(\text{Att wins in Game 1}) | \mathcal{E}_j^\sigma] - \Pr[(\text{Att wins in Game-Alt } j, 1) | \mathcal{E}_j^\sigma]| \leq \text{negl}(\lambda) \quad (3)$$

$$|\Pr[(\text{Att wins in Game 1}) | \neg \mathcal{E}_j^\sigma] - \Pr[(\text{Att wins in Game-Alt } j, 0) | \neg \mathcal{E}_j^\sigma]| \leq \text{negl}(\lambda) \quad (4)$$

Continuing with our proof, let us assume $\text{Att} = (\text{Att}_1, \text{Att}_2)$. Att_1 is a randomized algorithm that on input 1^λ , outputs message $m \in \{0,1\}^{\ell_{\text{msg}}}$ which it sends to the challenger, and state st which is sent to Att_2 . Att_2 is a randomized algorithm that receives state m, st from Att_1 and inputs PP, VK from challenger. It makes signature queries before outputting the forgery. We will now describe algorithm $\mathcal{B}$ that interacts with a unique signature $\mathcal{S}$ challenger. Let $\tau = \frac{32\lambda}{\epsilon^2}$.

1. $\mathcal{B}$ first runs $\text{Att}_1(1^\lambda)$ and receives message m and state st . It sends m to $\mathcal{S}$ challenger, and receives VK.
2. For $j = 1$ to ℓ_{sig} , do

- (a) Set $\text{count}_{j,0} = 0$. For $i = 1$ to τ ,
 - i. Choose RSA modulus N , $e \leftarrow \mathbb{Z}_{\phi(N)}^*$ and $K \leftarrow F.\text{setup}(1^\lambda)$. Set $y = \mathcal{S}.\text{Verify}[\text{VK}][m]$, $\text{PP} = (\mathcal{O}(1^\lambda, \text{TransformAlt}_{j,0,y,N,K}), \mathcal{O}(1^\lambda, \text{Transform-Image}_{N,K,e}), N, e)$ and send $\text{PP}, \text{VK}, m, \text{st}$ to Att_2 as input. Att_2 uses fresh randomness for each run.
 - ii. For each signing query x_r , $\mathcal{B}$ forwards x_r to the challenger, receives σ_r , which it sends to Att_2 .
 - iii. Finally, Att_2 outputs σ_{agg} and n tuples. If Att wins, $\mathcal{B}$ increments $\text{count}_{j,0}$.
- (b) Set $\text{count}_{j,1} = 0$. For $i = 1$ to τ
 - i. Choose RSA modulus N , $e \leftarrow \mathbb{Z}_{\phi(N)}^*$ and $K \leftarrow F.\text{setup}(1^\lambda)$. Set $y = \mathcal{S}.\text{Verify}[\text{VK}][m]$, $\text{PP} = (\mathcal{O}(1^\lambda, \text{TransformAlt}_{j,1,y,N,K}), \mathcal{O}(1^\lambda, \text{Transform-Image}_{N,K,e}), N, e)$ and send $\text{PP}, \text{VK}, m, \text{st}$ to Att_2 as input. Att_2 uses fresh randomness for each run.
 - ii. For each signing query x_r , $\mathcal{B}$ forwards x_r to the challenger, receives σ_r , which it sends to Att_2 .
 - iii. Finally, Att_2 outputs σ_{agg} and n tuples. If Att wins, $\mathcal{B}$ increments $\text{count}_{j,1}$.
- (c) If $\text{count}_{j,0} > \text{count}_{j,1}$, $\mathcal{B}$ sets $\alpha_j = 1$, else it sets $\alpha_j = 0$.

3. Finally, $\mathcal{B}$ outputs $\sigma' = \alpha_1 \dots \alpha_{\ell_{\text{sig}}}$ as forgery to challenger.

We will now analyze the winning probability of $\mathcal{B}$. In order to do this, we will first define a subset of verification keys which are ‘good’ (i.e. verification keys for which the difference between the advantages of Att in **Game 0** and **Game 1** is ‘large’) and then show that a non-negligible fraction of the verification keys are ‘good’. This technique is similar to the *heavy row lemma* used in [OO98].

For any $(m, \text{st}) \leftarrow \text{Att}_1(1^\lambda)$, let $\text{Good}_{m,\text{st}} \subset \mathcal{VK}$ be the set of verification keys VK such that the following holds:

$$\Pr[\text{Att}_2(\text{PP}, \text{VK}, m, \text{st}) \text{ wins in Game 0}] - \Pr[\text{Att}_2(\text{PP}, \text{VK}, m, \text{st}) \text{ wins in Game 1}] \geq \epsilon/2,$$

where the probability is taken over the random coins used by Att_2 and the random coins used by the challenger to compute PP and the signatures.

Let $\mathcal{E}$ denote the event $(m, \text{st}) \leftarrow \text{Att}_1(1^\lambda)$ **and** $(\text{SK}, \text{VK}) \leftarrow \mathcal{S}.\text{Gen}(1^\lambda)$ **and** $\text{VK} \in \text{Good}_{m,\text{st}}$. We can also view $\mathcal{E}$ as the set of all tuples $(m, \text{st}, \text{VK})$ such that $(m, \text{st}) \leftarrow \text{Att}_1(1^\lambda)$ and $(\text{SK}, \text{VK}) \leftarrow \mathcal{S}.\text{Gen}(1^\lambda)$ and $\text{VK} \in \text{Good}_{m,\text{st}}$.

Claim 4.1. $\Pr[\mathcal{E}] \geq \epsilon/2$, where the probability is over the random coins of Att_1 and $\mathcal{S}.\text{Gen}$.

Proof.

$$\begin{aligned}
 \epsilon &= \Pr[\text{Att wins in Game 0}] - \Pr[\text{Att wins in Game 1}] \\
 &= \Pr[\text{Att wins in Game 0} | \mathcal{E}] \Pr[\mathcal{E}] + \Pr[\text{Att wins in Game 0} | \neg \mathcal{E}] \Pr[\neg \mathcal{E}] \\
 &\quad - \Pr[\text{Att wins in Game 1} | \mathcal{E}] \Pr[\mathcal{E}] - \Pr[\text{Att wins in Game 1} | \neg \mathcal{E}] \Pr[\neg \mathcal{E}] \\
 &= \Pr[\mathcal{E}] (\Pr[\text{Att wins in Game 0} | \mathcal{E}] - \Pr[\text{Att wins in Game 1} | \mathcal{E}]) \\
 &\quad + \Pr[\neg \mathcal{E}] (\Pr[\text{Att wins in Game 0} | \neg \mathcal{E}] - \Pr[\text{Att wins in Game 1} | \neg \mathcal{E}]) \\
 &\leq \Pr[\mathcal{E}] + \epsilon/2
 \end{aligned}$$

This implies $\Pr[\mathcal{E}] \geq \epsilon/2$. ■

Let us assume event $\mathcal{E}$. We will now compute the probability that $\mathcal{B}$ fails to recover forgery, given $\mathcal{E}$. Let p_j denote the probability that the j^{th} bit of forgery σ' is incorrect, given $\mathcal{E}$.

Let $v = (m, \text{st}, \text{VK}) \in \mathcal{E}$. Define $\theta_{j,b}^v = \Pr[\text{Att}_2(\text{PP}, \text{VK}, m, \text{st}) \text{ wins in Game-Alt } j, b | v]$. Then, the expected value of $\text{count}_{j,b}$ given v , $E[\text{count}_{j,b} | v] = \theta_{j,b}^v \cdot \tau$. Note that in each of the runs, the random coins used by Att_2 and the random coins used by the challenger to compute PP and the signatures are chosen afresh. By Chernoff-Hoeffding bounds,

$$\Pr \left[|\text{count}_{j,0} - \theta_{j,0}^v \cdot \tau| > \left(\frac{\epsilon}{4}\right) \cdot \tau \mid v \right] \leq \exp \left(- \left(\frac{\epsilon^2}{32}\right) \cdot \tau \right) \quad (5)$$

$$\Pr \left[|\text{count}_{j,1} - \theta_{j,1}^v \cdot \tau| > \left(\frac{\epsilon}{4}\right) \cdot \tau \mid v \right] \leq \exp \left(- \left(\frac{\epsilon^2}{32}\right) \cdot \tau \right) \quad (6)$$

Setting $\tau = \frac{32\lambda}{\epsilon^2}$, we get that the above probabilities are bounded by a negligible function in λ .

We will now compute p_j .

$p_j = \Pr [\alpha_j \neq \sigma[j] \mid \mathcal{E}] = \sum_{v \in \mathcal{E}} \Pr [\alpha_j \neq \sigma[j] \mid v] \cdot \Pr [v \mid \mathcal{E}]$. Let us focus on one such term $\Pr [\alpha_j \neq \sigma[j] \mid v]$ for some $v \in \mathcal{E}$. $\Pr [\alpha_j \neq \sigma[j] \mid v] = \Pr [\alpha_j = 0 \text{ and } \sigma[j] = 1 \mid v] + \Pr [\alpha_j = 1 \text{ and } \sigma[j] = 0 \mid v]$.

Since $\mathcal{S}$ is a unique signature scheme, given v , $\sigma[j]$ is fixed. If $\sigma[j] = 1$, then,

$$\begin{aligned} & \Pr [\alpha_j \neq \sigma[j] \mid v] \\ &= \Pr [\alpha_j = 0 \text{ and } \sigma[j] = 1 \mid v] \\ &= \Pr [\alpha_j = 0 \mid v] \\ &= \Pr [\text{count}_{j,0} \leq \text{count}_{j,1} \mid v] \\ &\leq \Pr [\text{count}_{j,0} \leq (\theta_{j,0}^v + \theta_{j,1}^v)\tau/2 \mid v] + \Pr [\text{count}_{j,1} \geq (\theta_{j,0}^v + \theta_{j,1}^v)\tau/2 \mid v] \\ &= \Pr [\text{count}_{j,0} \leq \theta_{j,0}^v\tau/2 - (\theta_{j,0}^v - \theta_{j,1}^v)\tau/2 \mid v] + \Pr [\text{count}_{j,1} \geq \theta_{j,1}^v + (\theta_{j,0}^v - \theta_{j,1}^v)\tau/2 \mid v] \end{aligned} \quad (7)$$

Now, note that if $v \in \mathcal{E}$ and $\sigma[j] = 1$, then

$$\theta_{j,0}^v = \Pr[\text{Att}_2(\text{PP}, \text{VK}, m, \text{st}) \text{ wins in Game-Alt } j, 0 \mid v] \quad (8)$$

$$\geq \Pr[\text{Att}_2(\text{PP}, \text{VK}, m, \text{st}) \text{ wins in Game } 0 \mid v] - \text{negl}(\lambda) \quad (9)$$

$$\geq \Pr[\text{Att}_2(\text{PP}, \text{VK}, m, \text{st}) \text{ wins in Game } 1 \mid v] + \epsilon/2 - \text{negl}(\lambda) \quad (10)$$

$$\geq \Pr[\text{Att}_2(\text{PP}, \text{VK}, m, \text{st}) \text{ wins in Game-Alt } j, 1 \mid v] + \epsilon/2 - \text{negl}(\lambda) \quad (11)$$

$$= \theta_{j,1}^v + \epsilon/2 - \text{negl}(\lambda) \quad (12)$$

The transitions from Equation 8 and 9, and from Equation 10 to 11 follow from Equations 1 and 3 respectively, while the transition from Equation 9 to 10 uses the fact that $v \in \mathcal{E}$. Hence, getting back to Equation 7,

$$\begin{aligned} & \Pr [\text{count}_{j,0} \leq \theta_{j,0}^v\tau - (\theta_{j,0}^v - \theta_{j,1}^v)\tau/2 \mid v] + \Pr [\text{count}_{j,1} \geq \theta_{j,1}^v + (\theta_{j,0}^v - \theta_{j,1}^v)\tau/2 \mid v] \\ & \leq \Pr [\text{count}_{j,0} \leq \theta_{j,0}^v\tau - \epsilon \cdot \tau/4 \mid v] + \Pr [\text{count}_{j,1} \geq \theta_{j,1}^v + \epsilon \cdot \tau/4 \mid v] \end{aligned}$$

Now, we can use Equations 5 and 6 to conclude that $\Pr [\alpha_j \neq \sigma[j] \mid v] \leq \text{negl}(\lambda)$. A similar argument follows if v is such that $\sigma[j] = 0$. Therefore, $\Pr [\alpha_j \neq \sigma[j] \mid \mathcal{E}] \leq \text{negl}(\lambda)$. Hence, $\Pr[\mathcal{B} \text{ fails} \mid \mathcal{E}] \leq \text{negl}(\lambda)$. This implies $\Pr[\mathcal{B} \text{ wins}] \geq \Pr[\mathcal{B} \text{ wins} \mid \mathcal{E}] \Pr[\mathcal{E}] \geq \epsilon/2 - \text{negl}(\lambda)$. Since this violates the unforgeability of the signature scheme, we have our contradiction. ■

Claim 4.2. Assuming $i\mathcal{O}$ is a secure indistinguishability obfuscator, for any PPT adversary Att ,

$$\text{Adv}_{\text{Att}}^1 - \text{Adv}_{\text{Att}}^2 \leq \text{negl}(\lambda).$$

Proof. Suppose there exists a PPT adversary Att such that $\text{Adv}_{\text{Att}}^1 - \text{Adv}_{\text{Att}}^2 = \epsilon$. We will construct a PPT algorithm $\mathcal{B}$ that constructs two circuits C_0 and C_1 with identical functionality, and uses Att to distinguish between $i\mathcal{O}(C_0)$ and $i\mathcal{O}(C_1)$, thereby breaking the security of $i\mathcal{O}$.

$\mathcal{B}$ receives m from Att , chooses $(\text{SK}, \text{VK}) \leftarrow \mathcal{S}.\text{Gen}(1^\lambda)$, RSA modulus N , $e \leftarrow \mathbb{Z}_{\phi(N)}^*$ and $K \leftarrow F.\text{setup}(1^\lambda)$. It sets $y = \mathcal{S}.\text{Verify}(\text{VK} \parallel m)$ and computes $K\{y\} \leftarrow F.\text{puncture}(K, y)$. It sets $C_0 = \text{Transform-Image}_{N,K,e}$

and $C_1 = \text{Transform-Image-1}_{y,N,K\{y\},z,e}$, and sends C_0, C_1 to the $i\mathcal{O}$ challenger. It receives $C = i\mathcal{O}(C_b)$. $\mathcal{B}$ sets $\text{PP} = (i\mathcal{O}(\text{Transform-Reject}_{y,N,K\{y\}}), C, N, e)$ and sends PP, VK to Att .

Note that $\mathcal{B}$ can respond to the signing queries perfectly, since it has SK . Finally, if Att wins, then $\mathcal{B}$ outputs 0, else it outputs 1. Clearly, if $C = i\mathcal{O}(C_0)$, then it corresponds to **Game 1**, else it corresponds to **Game 2**.

To conclude, we need to argue that C_0 and C_1 have identical functionality. This follows from the correctness property of puncturable PRFs. $\blacksquare$

Claim 4.3. Assuming F is a selectively secure puncturable PRF, for any PPT adversary Att ,

$$\text{Adv}_{\text{Att}}^2 - \text{Adv}_{\text{Att}}^3 \leq \text{negl}(\lambda).$$

Proof. Suppose there exists a PPT adversary Att such that $\text{Adv}_{\text{Att}}^2 - \text{Adv}_{\text{Att}}^3 = \epsilon$. We will construct a PPT algorithm $\mathcal{B}$ that uses Att to break the security of puncturable PRF F with advantage ϵ .

First, $\mathcal{B}$ receives the message m from Att . As in **Game 2** and **Game 3**, it computes $(\text{SK}, \text{VK}) \leftarrow \mathcal{S}.\text{Gen}(1^\lambda)$, chooses an RSA modulus N and $e \leftarrow \mathbb{Z}_{\phi(N)}^*$. Next, it sends $y = \mathcal{S}.\text{Verify}[\|\text{VK}\|]m$ as the challenge to the PRF challenger. $\mathcal{B}$ receives a punctured key $K\{y\}$ and $z \in \mathbb{Z}_N^*$, where $z = F(K, y)$ or $z \leftarrow \mathbb{Z}_N^*$. $\mathcal{B}$ sets the public parameters $\text{PP} = (i\mathcal{O}(\text{Transform-Reject}_{y,N,K\{y\}}), i\mathcal{O}(\text{Transform-Image-1}_{y,N,K\{y\},z,e}), N, e)$. It sends PP, VK to Att .

The signing phase and forgery phase are exactly similar in **Game 2** and **Game 3**. For each signing query x_i , $\mathcal{B}$ sends $\mathcal{S}.\text{Sign}(\text{SK}, x_i)$ to Att . Finally, Att outputs the forgery σ_{agg} and n tuples $\{(\text{Verify}_i, \text{VK}_i, m_i)\}$.

Note that if $z = F(K, y)$, then $\mathcal{B}$ simulates **Game 2** perfectly. If $z \leftarrow \mathbb{Z}_N^*$, $\mathcal{B}$ simulates **Game 3** perfectly. This concludes our proof. $\blacksquare$

Claim 4.4. Assuming RSA is secure, for any PPT adversary Att , $\text{Adv}_{\text{Att}}^3 \leq \text{negl}(\lambda)$.

Proof. Suppose there exists a PPT adversary Att such that $\text{Adv}_{\text{Att}}^3 = \epsilon$. We will construct a PPT algorithm $\mathcal{B}$ that breaks the RSA assumption with advantage ϵ .

$\mathcal{B}$ receives message m from Att . It receives the RSA tuple (N, e, z) from the RSA challenger. $\mathcal{B}$ chooses $(\text{SK}, \text{VK}) \leftarrow \mathcal{S}.\text{Gen}(1^\lambda)$, $K \leftarrow F.\text{setup}(1^\lambda)$. Next, it sets $y = \text{Verify}[\|\text{VK}\|]m$ and computes $K\{y\} \leftarrow F.\text{puncture}(K, y)$. It sets $\text{PP} = (i\mathcal{O}(1^\lambda(\text{Transform-Reject}_{y,N,K\{y\}}), i\mathcal{O}(\text{Transform-Image-1}_{y,N,K\{y\},z,e}), N, e)$ and sends PP, VK to Att .

Att sends signature queries, which $\mathcal{B}$ can compute by itself, since it has the signing key SK . Finally, Att outputs forgery σ_{agg} along with n tuples $\{\text{Verify}_i, \text{VK}_i, m_i\}$. If Att wins, then all n tuples are distinct, and there exists $i^* \in [n]$ such that $\text{Verify}_{i^*} = \mathcal{S}.\text{Verify}$, $\text{VK}_{i^*} = \text{VK}$, $m_{i^*} = m$ and $\sigma_{\text{agg}}^e = (\prod_{i \neq i^*} \text{Transform-Image-1}_{y,N,K\{y\},z,e}(\text{Verify}_i, \text{VK}_i, m_i))z \pmod{N}$. For all $i \neq i^*$, $\text{Transform-Image-1}_{y,N,K\{y\},z,e}$ outputs $F(K, \text{Verify}_i[\|\text{VK}_i\|]m_i)^e$ on input $(\text{Verify}_i, \text{VK}_i, m_i)$. Therefore, $\left(\frac{\sigma_{\text{agg}}}{\prod_{i \neq i^*} F(K, \text{Verify}_i[\|\text{VK}_i\|]m_i)}\right)^e = z \pmod{N}$. $\blacksquare$

Using the above claims, it follows that any PPT adversary has negligible advantage in **Game 0**, assuming $i\mathcal{O}$ is a secure indistinguishability obfuscator, F is a selectively secure puncturable PRF and the RSA assumption holds. Therefore, the construction in Section 4 is selectively secure with respect to all secure unique signature schemes.

5 Universal Aggregation of Arbitrary Signatures Using VBB Obfuscation

In this section, we will describe our construction based on *virtual black box* obfuscation. The construction is similar to the one in Section 4, the only difference being in program **Transform-VBB**, which now takes

some additional inputs and has additional constants hardwired. The additional inputs/constants are used for “oracle assimilation” (see Section 1 for a discussion on this technical issue).

We will assume that all signing algorithms (corresponding to schemes whose signatures need to be aggregated) use at most ℓ_{rnd} random bits to compute signatures, for some polynomial ℓ_{rnd} . We use a pseudorandom generator $\text{PRG} : \{0, 1\}^\ell \leftarrow \{0, 1\}^{2\ell}$ (where ℓ is some polynomial in λ), a (standard) PRF $\tilde{F}$ with key space $\tilde{\mathcal{K}}$, domain $\tilde{\mathcal{X}}$ and range $\tilde{\mathcal{Y}} = \{0, 1\}^{\ell_{\text{rnd}}}$ and a puncturable PRF F as in Section 4.

Our universal signature aggregator consists of the three algorithms **UniversalSetup**, **UniversalAgg** and **UniversalVerify** described below.

UniversalSetup(1^λ) **UniversalSetup** first chooses random primes $p, q \in \Theta(2^\lambda)$, sets the RSA modulus $N = pq$. It chooses $e \leftarrow \mathbb{Z}_{\phi(N)}^*$, PRF key $K \leftarrow F.\text{setup}(1^\lambda)$ as in Section 4. It computes obfuscations of the programs **Transform-VBB** $_{N,K}$ ¹⁶ and **Transform-Image** $_{N,K,e}$ ¹⁷, where **Transform-VBB** $_{N,K}$ is defined below, while **Transform-Image** $_{N,K,e}$ is the same as in Section 4. It sets the public parameters to be $\text{PP} = (\mathcal{O}(\text{Transform-VBB}_{N,K}), \mathcal{O}(\text{Transform-Image}_{N,K,e}), N, e)$.

Transform-VBB $_{N,K}$:

Inputs: $b \in \{0, 1\}$, $a \in \{0, 1\}^\ell$, $\text{Verify}' \in \{0, 1\}^{\ell_{\text{ver}}}$, $\text{VK}' \in \{0, 1\}^{\ell_{\text{vk}}}$, $m' \in \{0, 1\}^{\ell_{\text{msg}}}$, $\sigma' \in \{0, 1\}^{\ell_{\text{sig}}}$.

Constants : RSA modulus $N \in \mathbb{N}$, $K \in \mathcal{K}$.

```

if  $b = 0$  then
    Output  $\perp$ .
else if  $\text{Verify}'(\text{VK}', m', \sigma') = 0$  then
    Output  $\perp$ .
else
    Output  $F(K, \text{Verify}' || \text{VK}' || m')$ .
end if

```

UniversalAgg($\text{PP}, \{(\text{Verify}_i, \text{VK}_i, m_i, \sigma_i)\}_{i=1}^n$): Let $\text{PP} = (P_1, P_2, N, e)$. **UniversalAgg** first checks that all the n tuples are distinct. If not, it outputs $\perp$. Else, it computes $t_i = P_1(\text{Verify}_i, \text{VK}_i, m_i, \sigma_i)$ for each $i \leq n$. If $t_i = \perp$ for some i , then **UniversalAgg** outputs $\perp$, else it outputs $\sigma_{\text{agg}} = \prod_i t_i \pmod{N}$.

UniversalVerify($\text{PP}, \{(\text{Verify}_i, \text{VK}_i, m_i)\}_{i=1}^n, \sigma_{\text{agg}}$): Let $\text{PP} = (P_1, P_2, N, e)$. **UniversalVerify** checks that the n tuples are distinct. If not, it outputs 0. Else, it computes, for all $i \leq n$, $s_i = \text{Transform-Image}(\text{Verify}_i, \text{VK}_i, m_i)$. If $(\prod_i s_i) = \sigma_{\text{agg}}^e \pmod{N}$, it outputs 1, else it outputs 0.

5.1 Proof of Security

We will now prove that the construction in Section 5 is selectively secure with respect to all secure signature schemes.

Theorem 5.1. Assuming $\mathcal{O}$ is a secure virtual black-box obfuscator for a class of circuits $\mathcal{C}$ (as defined in Section 5.1.3), F is a selectively secure puncturable PRF, $\tilde{F}$ is a secure PRF, PRG is a secure pseudorandom generator and RSA is secure, for all $(\ell_{\text{ver}}, \ell_{\text{vk}}, \ell_{\text{msg}}, \ell_{\text{sig}})$ -length qualified signature schemes $\mathcal{S}$, the universal signature aggregator $(\ell_{\text{ver}}, \ell_{\text{vk}}, \ell_{\text{msg}}, \ell_{\text{sig}})$ -**UniversalSigAgg** is selectively secure with respect to $\mathcal{S}$.

We will now describe the intermediate hybrid experiments.

¹⁶Padded appropriately to be of the same size as **Transform-VBB-1**, **Transform-VBB-2**, **Transform-VBB-3** defined later in this section.

¹⁷Padded appropriately to be of the same size as **Transform-Image-1** as in Section 4.

5.1.1 Sequence of Games

Game 0: This game corresponds to $\text{Exp}_{\text{Att}, \mathcal{S}}^{\text{sel}}$.

1. Att sends message m .
2. Compute $(\text{SK}, \text{VK}) \leftarrow \mathcal{S}.\text{Gen}(1^\lambda)$. Choose an RSA modulus N , $e \leftarrow \mathbb{Z}_{\phi(N)}^*$, $K \leftarrow F.\text{setup}(1^\lambda)$ and set $\text{PP} = (\mathcal{O}(\text{Transform-VBB}_{N,K}), \mathcal{O}(\text{Transform-Image}_{N,K,e}), N, e)$. Send PP, VK to Att.
3. For each signing query $x_i \neq m$, compute $\sigma_i \leftarrow \mathcal{S}.\text{Sign}(\text{SK}, x_i)$ and send σ_i to Att.
4. Att sends forgery σ_{agg} and n tuples $\{(\text{Verify}_i, \text{VK}_i, m_i)\}$. Att wins if $\exists i^* \in [n]$ such that $\text{Verify}_{i^*} = \mathcal{S}.\text{Verify}, \text{VK}_{i^*} = \text{VK}$ and $m_{i^*} = m$ and $\text{UniversalVerify}(\text{PP}, \{(\text{Verify}_i, \text{VK}_i, m_i)\}, \sigma_{\text{agg}}) = 1$.

Game 1: In this game, the challenger uses pseudorandomly generated strings as randomness for the signature queries.

1. Att sends message m .
2. Compute $(\text{SK}, \text{VK}) \leftarrow \mathcal{S}.\text{Gen}(1^\lambda)$. Choose an RSA modulus N , $e \leftarrow \mathbb{Z}_{\phi(N)}^*$, $K \leftarrow F.\text{setup}(1^\lambda)$.
Choose standard PRF key $\tilde{K} \leftarrow \tilde{F}.\text{setup}(1^\lambda)$.
Set $\text{PP} = (\mathcal{O}(\text{Transform-VBB}_{N,K}), \mathcal{O}(\text{Transform-Image}_{N,K,e}), N, e)$. Send PP, VK to Att.
3. For each signing query $x_i \neq m$, choose $\rho_i \leftarrow \{0, 1\}^{\ell_{\text{sig}}}$, compute $r_i = \tilde{F}(\tilde{K}, \rho_i)$, $\sigma_i \leftarrow \mathcal{S}.\text{Sign}(\text{SK}, x_i; r_i)$ and send σ_i to Att.
4. Att sends forgery σ_{agg} and n tuples $\{(\text{Verify}_i, \text{VK}_i, m_i)\}$. Att wins if $\exists i^* \in [n]$ such that $\text{Verify}_{i^*} = \mathcal{S}.\text{Verify}, \text{VK}_{i^*} = \text{VK}$ and $m_{i^*} = m$ and $\text{UniversalVerify}(\text{PP}, \{(\text{Verify}_i, \text{VK}_i, m_i)\}, \sigma_{\text{agg}}) = 1$.

Game 2: In this game, the challenger uses the program Transform-VBB-1 instead of Transform-VBB. Unlike Transform-VBB, Transform-VBB-1 uses the input a to check if $\text{PRG}(a)$ is equal to the hardwired α . If the ‘mode’ bit is 0 and $\text{PRG}(a) = \alpha$, then the program outputs the verification key VK and a signature on the desired message.

1. Att sends message m .
2. Compute $(\text{SK}, \text{VK}) \leftarrow \mathcal{S}.\text{Gen}(1^\lambda)$. Choose an RSA modulus N , $e \leftarrow \mathbb{Z}_{\phi(N)}^*$, $K \leftarrow F.\text{setup}(1^\lambda)$.
Choose PRF key $\tilde{K} \leftarrow \tilde{F}.\text{setup}(1^\lambda)$, $\alpha \leftarrow \{0, 1\}^{2\ell}$.
Let Transform-VBB-1¹⁸ be the circuit defined below.
Set $\text{PP} = (\mathcal{O}(\text{Transform-VBB-1}_{N,K,\alpha,\text{SK},\tilde{K}}), \mathcal{O}(\text{Transform-Image}_{N,K,e}), N, e)$. Send PP, VK to Att.
3. For each signing query $x_i \neq m$, choose $\rho_i \leftarrow \{0, 1\}^{\ell_{\text{sig}}}$, compute $r_i = \tilde{F}(\tilde{K}, \rho_i)$, $\sigma_i \leftarrow \mathcal{S}.\text{Sign}(\text{SK}, x_i; r_i)$ and send σ_i to Att.
4. Att sends forgery σ_{agg} and n tuples $\{(\text{Verify}_i, \text{VK}_i, m_i)\}$. Att wins if $\exists i^* \in [n]$ such that $\text{Verify}_{i^*} = \mathcal{S}.\text{Verify}, \text{VK}_{i^*} = \text{VK}$ and $m_{i^*} = m$ and $\text{UniversalVerify}(\text{PP}, \{(\text{Verify}_i, \text{VK}_i, m_i)\}, \sigma_{\text{agg}}) = 1$.

¹⁸Padded appropriately to be of the same size as Transform-VBB, Transform-VBB-2 and Transform-VBB-3.

Transform-VBB-1 _{$N, K, \alpha, SK, \tilde{K}$} :

Inputs: $b \in \{0, 1\}$, $a \in \{0, 1\}^\ell$, $\text{Verify}' \in \{0, 1\}^{\ell_{\text{ver}}}$, $\text{VK}' \in \{0, 1\}^{\ell_{\text{vk}}}$, $m' \in \{0, 1\}^{\ell_{\text{msg}}}$, $\sigma' \in \{0, 1\}^{\ell_{\text{sig}}}$.

Constants : RSA modulus $N \in \mathbb{N}$, $K \in \mathcal{K}$, $\alpha \in \{0, 1\}^{2\ell}$, $SK \in \mathcal{SK}$, $\tilde{K} \in \tilde{\mathcal{K}}$.

```

if  $b = 0$  then
  if  $\text{PRG}(a) \neq \alpha$  then
    Output  $\perp$ .
  else
    Output  $(\text{VK}, \mathcal{S}.\text{Sign}(SK, m'; \tilde{F}(\tilde{K}, \sigma')))$ .
  end if
else if  $\text{Verify}'(\text{VK}', m', \sigma') = 0$  then
  Output  $\perp$ .
else
  Output  $F(K, \text{Verify}' || \text{VK}' || m')$ .
end if

```

Game 3: In this experiment, α is a pseudorandom string; i.e. $\alpha = \text{PRG}(a)$, where $a \leftarrow \{0, 1\}^\ell$.

1. Att sends message m .
2. Compute $(SK, VK) \leftarrow \mathcal{S}.\text{Gen}(1^\lambda)$. Choose an RSA modulus N , $e \leftarrow \mathbb{Z}_{\phi(N)}^*$, $K \leftarrow F.\text{setup}(1^\lambda)$. Choose $a \leftarrow \{0, 1\}^\ell$ and set $\alpha = \text{PRG}(a)$. Choose PRF key $\tilde{K} \leftarrow \tilde{F}.\text{setup}(1^\lambda)$. Set $\text{PP} = (\mathcal{O}(\text{Transform-VBB-1}_{N, K, \alpha, SK, \tilde{K}}), \mathcal{O}(\text{Transform-Image}_{N, K, e}), N, e)$. Send PP, VK to Att.
3. For each signing query $x_i \neq m$, choose $\rho_i \leftarrow \{0, 1\}^{\ell_{\text{sig}}}$, compute $r_i = \tilde{F}(\tilde{K}, \rho_i)$, $\sigma_i \leftarrow \mathcal{S}.\text{Sign}(SK, x_i; r_i)$ and send σ_i to Att.
4. Att sends forgery σ_{agg} and n tuples $\{(\text{Verify}_i, \text{VK}_i, m_i)\}$. Att wins if $\exists i^* \in [n]$ such that $\text{Verify}_{i^*} = \mathcal{S}.\text{Verify}$, $\text{VK}_{i^*} = \text{VK}$ and $m_{i^*} = m$ and $\text{UniversalVerify}(\text{PP}, \{(\text{Verify}_i, \text{VK}_i, m_i)\}, \sigma_{\text{agg}}) = 1$.

Game 4: This experiment is similar to the previous one, except that the challenger uses Transform-VBB-2 instead of Transform-VBB-1.

1. Att sends message m .
2. Compute $(SK, VK) \leftarrow \mathcal{S}.\text{Gen}(1^\lambda)$. Choose an RSA modulus N , $e \leftarrow \mathbb{Z}_{\phi(N)}^*$, $K \leftarrow F.\text{setup}(1^\lambda)$. Choose $a \leftarrow \{0, 1\}^\ell$ and set $\alpha = \text{PRG}(a)$. Choose $\tilde{K} \leftarrow \tilde{F}.\text{setup}(1^\lambda)$. Let Transform-VBB-2¹⁹ be the circuit defined below. Set $y = \mathcal{S}.\text{Verify} || \text{VK} || m$, $\text{PP} = (\mathcal{O}(\text{Transform-VBB-2}_{y, N, K, \alpha, SK, \tilde{K}}), \mathcal{O}(\text{Transform-Image}_{N, K, e}), N, e)$. Send PP, VK to Att.
3. For each signing query $x_i \neq m$, choose $\rho_i \leftarrow \{0, 1\}^{\ell_{\text{sig}}}$, compute $r_i = \tilde{F}(\tilde{K}, \rho_i)$ and send $\sigma_i = \mathcal{S}.\text{Sign}(SK, x_i; r_i)$ to Att.
4. Att sends forgery σ_{agg} and n tuples $\{(\text{Verify}_i, \text{VK}_i, m_i)\}$. Att wins if $\exists i^* \in [n]$ such that $\text{Verify}_{i^*} = \mathcal{S}.\text{Verify}$, $\text{VK}_{i^*} = \text{VK}$ and $m_{i^*} = m$ and $\text{UniversalVerify}(\text{PP}, \{(\text{Verify}_i, \text{VK}_i, m_i)\}, \sigma_{\text{agg}}) = 1$.

¹⁹Padded appropriately to be of the same size as Transform-VBB, Transform-VBB-1 and Transform-VBB-3.

Transform-VBB-2 _{$y, N, K, \alpha, SK, \tilde{K}$} :

Inputs: $b \in \{0, 1\}$, $a \in \{0, 1\}^\ell$, $\text{Verify}' \in \{0, 1\}^{\ell_{\text{ver}}}$, $\text{VK}' \in \{0, 1\}^{\ell_{\text{vk}}}$, $m' \in \{0, 1\}^{\ell_{\text{msg}}}$, $\sigma' \in \{0, 1\}^{\ell_{\text{sig}}}$.

Constants : $y \in \{0, 1\}^{\ell_{\text{ver}}} \times \{0, 1\}^{\ell_{\text{vk}}} \times \{0, 1\}^{\ell_{\text{msg}}}$, RSA modulus $N \in \mathbb{N}$, $K \in \mathcal{K}$, $\alpha \in \{0, 1\}^{2\ell}$, $SK \in \mathcal{SK}$, $\tilde{K} \in \tilde{\mathcal{K}}$.

```

if  $b = 0$  then
  if  $\text{PRG}(a) \neq \alpha$  then
    Output  $\perp$ .
  else
    Output  $(\text{VK}, \mathcal{S}.\text{Sign}(SK, m'; \tilde{F}(\tilde{K}, \sigma')))$ .
  end if
else if  $\text{Verify}'(\text{VK}', m', \sigma') = 0$  then
  Output  $\perp$ .
else if  $\text{Verify}' || \text{VK}' || m' = y$  then
  Output  $\perp$ .
else
  Output  $F(K, \text{Verify}' || \text{VK}' || m')$ .
end if

```

Game 5: In this experiment, the challenger uses a key punctured at y instead of the master PRF key.

1. Att sends message m .
2. Compute $(SK, VK) \leftarrow \mathcal{S}.\text{Gen}(1^\lambda)$. Choose an RSA modulus N , $e \leftarrow \mathbb{Z}_{\phi(N)}^*$, $K \leftarrow F.\text{setup}(1^\lambda)$. Choose $a \leftarrow \{0, 1\}^\ell$ and set $\alpha = \text{PRG}(a)$. Choose $\tilde{K} \leftarrow \tilde{F}.\text{setup}(1^\lambda)$. Set $y = \mathcal{S}.\text{Verify} || VK || m$, compute $K\{y\} \leftarrow F.\text{puncture}(K, y)$ and $z = F(K, y)^e$. Let Transform-VBB-3²⁰ be the circuit defined below, while Transform-Image-1²¹, e is the same as in Section 4.1 Set $PP = (\mathcal{O}(\text{Transform-VBB-3}_{y, N, K\{y\}, \alpha, SK, \tilde{K}}), \mathcal{O}(\text{Transform-Image-1}_{y, N, K\{y\}, z, e}))$. Send PP , VK to Att.
3. For each signing query $x_i \neq m$, choose $\rho_i \leftarrow \{0, 1\}^{\ell_{\text{sig}}}$, compute $r_i = \tilde{F}(\tilde{K}, \rho_i)$ and send $\sigma_i = \mathcal{S}.\text{Sign}(SK, x_i; r_i)$ to Att.
4. Att sends forgery σ_{agg} and n tuples $\{(\text{Verify}_i, \text{VK}_i, m_i)\}$. Att wins if $\exists i^* \in [n]$ such that $\text{Verify}_{i^*} = \mathcal{S}.\text{Verify}$, $\text{VK}_{i^*} = \text{VK}$ and $m_{i^*} = m$ and $\text{UniversalVerify}(PP, \{(\text{Verify}_i, \text{VK}_i, m_i)\}, \sigma_{\text{agg}}) = 1$.

²⁰Padded appropriately to be of the same size as Transform-VBB, Transform-VBB-1 and Transform-VBB-2.

²¹Padded appropriately to be of the same size as Transform-Image-1.

Transform-VBB-3 _{$y, N, K\{y\}, \alpha, SK, \tilde{K}$} :

Inputs: $b \in \{0, 1\}, a \in \{0, 1\}^\ell, \text{Verify}' \in \{0, 1\}^{\ell_{\text{ver}}}, \text{VK}' \in \{0, 1\}^{\ell_{\text{vk}}}, m' \in \{0, 1\}^{\ell_{\text{msg}}}, \sigma' \in \{0, 1\}^{\ell_{\text{sig}}}$.

Constants : $y \in \{0, 1\}^{\ell_{\text{ver}}} \times \{0, 1\}^{\ell_{\text{vk}}} \times \{0, 1\}^{\ell_{\text{msg}}}$, RSA modulus $N \in \mathbb{N}$, $K\{y\} \in \mathcal{K}_p$, $\alpha \in \{0, 1\}^{2\ell}$, $SK \in \mathcal{SK}$, $\tilde{K} \in \tilde{\mathcal{K}}$.

```

if  $b = 0$  then
  if  $\text{PRG}(a) \neq \alpha$  then
    Output  $\perp$ .
  else
    Output  $(\text{VK}, \mathcal{S}.\text{Sign}(\text{SK}, m'; \tilde{F}(\tilde{K}, \sigma')))$ .
  end if
else if  $\text{Verify}'(\text{VK}', m', \sigma') = 0$  then
  Output  $\perp$ .
else if  $\text{Verify}' || \text{VK}' || m' = y$  then
  Output  $\perp$ .
else
  Output  $F.\text{eval}(K\{y\}, \text{Verify}' || \text{VK}' || m')$ .
end if

```

Game 6: Here the challenger chooses a uniformly random $z \leftarrow \mathbb{Z}_N^*$.

1. Att sends message m .
2. Compute $(\text{SK}, \text{VK}) \leftarrow \mathcal{S}.\text{Gen}(1^\lambda)$. Choose an RSA modulus N , $e \leftarrow \mathbb{Z}_{\phi(N)}^*$, $K \leftarrow F.\text{setup}(1^\lambda)$. Choose $a \leftarrow \{0, 1\}^\ell$ and set $\alpha = \text{PRG}(a)$. Choose $\tilde{K} \leftarrow \tilde{F}.\text{setup}(1^\lambda)$. Set $y = \mathcal{S}.\text{Verify} || \text{VK} || m$, compute $K\{y\} \leftarrow F.\text{puncture}(K, y)$ and $z \leftarrow \mathbb{Z}_N^*$. Set $\text{PP} = (\mathcal{O}(\text{Transform-VBB-3}_{y, N, K\{y\}, \alpha, \text{SK}, \tilde{K}}), \mathcal{O}(\text{Transform-Image-1}_{y, N, K\{y\}, z, e}), e)$. Send PP, VK to Att.
3. For each signing query $x_i \neq m$, compute $\sigma_i \leftarrow \mathcal{S}.\text{Sign}(\text{SK}, x_i)$ and send σ_i to Att.
4. Att sends forgery σ_{agg} and n tuples $\{(\text{Verify}_i, \text{VK}_i, m_i)\}$. Att wins if $\exists i^* \in [n]$ such that $\text{Verify}_{i^*} = \mathcal{S}.\text{Verify}$, $\text{VK}_{i^*} = \text{VK}$ and $m_{i^*} = m$ and $\text{UniversalVerify}(\text{PP}, \{(\text{Verify}_i, \text{VK}_i, m_i)\}, \sigma_{\text{agg}}) = 1$.

5.1.2 Analysis

We will now show that if a PPT adversary has non negligible advantage in **Game** i , then it has non-negligible advantage in the next game. Some of the proofs are exactly similar to the corresponding ones in Section 4.1, and hence we skip them in this section. Except the part involving oracle assimilation, the remaining proofs are relatively easier. The step involving oracle assimilation is discussed in a separate subsection (Section 5.1.3).

Let $\text{Adv}_{\text{Att}}^j$ denote the advantage of adversary Att in **Game** j .

Claim 5.1. Assuming $\tilde{F}$ is a secure PRF, for any PPT adversary Att,

$$\text{Adv}_{\text{Att}}^0 - \text{Adv}_{\text{Att}}^1 \leq \text{negl}(\lambda).$$

Proof. Suppose there exists an adversary Att such that $\text{Adv}_{\text{Att}}^0 - \text{Adv}_{\text{Att}}^1 = \epsilon$. We will construct a PPT algorithm $\mathcal{B}$ that uses Att and breaks the security of $\tilde{F}$ with advantage ϵ .

$\mathcal{B}$ receives message m from Att. It chooses $(\text{SK}, \text{VK}) \leftarrow \mathcal{S}.\text{Gen}(1^\lambda)$, RSA modulus N , $e \leftarrow \mathbb{Z}_{\phi(N)}^*$ and $K \leftarrow F.\text{setup}(1^\lambda)$. It sets $\text{PP} = (\mathcal{O}(\text{Transform-VBB}_{N, K}), \mathcal{O}(\text{Transform-Image}_{N, K, e}), e)$ and sends PP, VK to Att.

For each signing query x_i , $\mathcal{B}$ first chooses $\rho_i \leftarrow \{0, 1\}^{\ell_{\text{sig}}}$ and sends ρ_i to the PRF challenger. In response, it receives r_i . $\mathcal{B}$ sends $\sigma_i = \mathcal{S}.\text{Sign}(\text{SK}, x_i; r_i)$ to Att.

Finally, Att outputs a forgery. If Att wins, then $\mathcal{B}$ outputs 1, indicating that the PRF challenger's responses were truly random. Else it outputs 0.

If the PRF challenger's responses were truly random, then for each query ρ_i , r_i is a truly random string. Therefore, this corresponds to Game 0. If the PRF challenger's responses were pseudorandom, then there exists a PRF key $\tilde{K}$ such that for each query ρ_i , $r_i = \tilde{F}(\tilde{K}, \rho_i)$. This corresponds to Game 1. Therefore, $\text{Adv}_{\mathcal{B}}^{\tilde{F}} = \epsilon$. $\blacksquare$

Claim 5.2. Assuming $\mathcal{O}$ is a secure indistinguishability obfuscator, for any PPT adversary Att,

$$\text{Adv}_{\text{Att}}^1 - \text{Adv}_{\text{Att}}^2 \leq \text{negl}(\lambda).$$

Proof. Suppose there exists a PPT adversary Att such that $\text{Adv}_{\text{Att}}^1 - \text{Adv}_{\text{Att}}^2 = \epsilon$. We will construct a PPT algorithm $\mathcal{B}$ that breaks the security of $\mathcal{O}$ with advantage ϵ .

$\mathcal{B}$ receives message m from Att. It chooses $(\text{SK}, \text{VK}) \leftarrow \mathcal{S}.\text{Gen}(1^\lambda)$, RSA modulus N , $e \leftarrow \mathbb{Z}_{\phi(N)}^*$, $K \leftarrow F.\text{setup}(1^\lambda)$, $\tilde{K} \leftarrow \tilde{F}.\text{setup}(1^\lambda)$ and $\alpha \leftarrow \{0, 1\}^{2\ell}$. It sets $C_0 = \text{Transform-VBB}_{N,K}$, $C_1 = \text{Transform-VBB-1}_{N,K,\alpha,\text{SK},\tilde{K}}$ and sends C_0, C_1 to the $\mathcal{O}$ challenger. It receives an obfuscated circuit $C' = \mathcal{O}(C_b)$ in response, and sets $\text{PP} = (C', \mathcal{O}(\text{Transform-Image}_{N,K,e}), e)$ and sends PP, VK to Att.

For each signing query x_i , $\mathcal{B}$ first chooses $\rho_i \leftarrow \{0, 1\}^{\ell_{\text{sig}}}$ and computes $r_i = \tilde{F}(\tilde{K}, \rho_i)$. $\mathcal{B}$ sends $\sigma_i = \mathcal{S}.\text{Sign}(\text{SK}, x_i; r_i)$ to Att.

Finally, Att outputs a forgery. If Att wins, then $\mathcal{B}$ outputs 0, else it outputs 1. Clearly, if $b = 0$, then this corresponds to Game 1, else it corresponds to Game 2. Therefore, in order to show that $\text{Adv}_{\mathcal{B}}^{\mathcal{O}} = \epsilon$, we need to show that C_0 and C_1 have identical functionality.

This follows from the observation that with overwhelming probability, there exists no $a \in \{0, 1\}^\ell$ such that $\alpha = \text{PRG}(a)$, since α is chosen uniformly at random. As a result, on input $(0, a, \text{Verify}', \text{VK}', m', \sigma')$, both circuits output $\perp$ for all $a, \text{Verify}', \text{VK}', m', \sigma'$. This concludes our proof. $\blacksquare$

Claim 5.3. Assuming PRG is a secure pseudorandom generator, for any PPT adversary Att,

$$\text{Adv}_{\text{Att}}^2 - \text{Adv}_{\text{Att}}^3 \leq \text{negl}(\lambda).$$

Proof. Suppose there exists a PPT adversary Att such that $\text{Adv}_{\text{Att}}^2 - \text{Adv}_{\text{Att}}^3 = \epsilon$. We will construct a PPT algorithm $\mathcal{B}$ that breaks the security of PRG with advantage ϵ .

$\mathcal{B}$ receives α from the PRG challenger, where $\alpha \leftarrow \{0, 1\}^\ell$ or $\alpha = \text{PRG}(a)$ for some $a \leftarrow \{0, 1\}^\ell$. Note that $\mathcal{B}$ can simulate either Game 1 or Game 2 perfectly using α . It chooses $(\text{SK}, \text{VK}) \leftarrow \mathcal{S}.\text{Gen}(1^\lambda)$, RSA modulus N , $e \leftarrow \mathbb{Z}_{\phi(N)}^*$, $K \leftarrow F.\text{setup}(1^\lambda)$, $\tilde{K} \leftarrow \tilde{F}.\text{setup}(1^\lambda)$. $\mathcal{B}$ sets $\text{PP} = (\mathcal{O}(\text{Transform-VBB-1}_{N,K,\alpha,\text{SK},\tilde{K}}), \mathcal{O}(\text{Transform-Image}_{N,K,e}), e)$ and sends PP, VK to Att.

For the signature queries, $\mathcal{B}$ uses SK. Finally, if Att wins, $\mathcal{B}$ outputs 1 (indicating that $\alpha \leftarrow \{0, 1\}^{2\ell}$). Else it outputs 0. Clearly, $\text{Adv}_{\mathcal{B}} = \epsilon$. This concludes our proof. $\blacksquare$

Lemma 5.1. Assuming $\mathcal{O}$ is a secure virtual black box obfuscator for a class of circuits $\mathcal{C}$ (defined in Section 5.1.3), $\tilde{F}$ is a secure pseudorandom function, PRG is a secure pseudorandom generator and $\mathcal{S}$ is a $(\ell_{\text{ver}}, \ell_{\text{vk}}, \ell_{\text{msg}}, \ell_{\text{sig}})$ -length qualified secure signature scheme,

$$\text{Adv}_{\text{Att}}^3 - \text{Adv}_{\text{Att}}^4 \leq \text{negl}(\lambda).$$

The proof of this lemma consists of multiple intermediate hybrids, and is contained in Section 5.1.3.

Claim 5.4. Assuming $\mathcal{O}$ is a secure indistinguishability obfuscator, for any PPT adversary Att,

$$\text{Adv}_{\text{Att}}^4 - \text{Adv}_{\text{Att}}^5 \leq \text{negl}(\lambda).$$

Proof. Similar to proof of Claim 4.2. ■

Claim 5.5. Assuming F is a selectively secure puncturable PRF, for any PPT adversary Att ,

$$\text{Adv}_{\text{Att}}^5 - \text{Adv}_{\text{Att}}^6 \leq \text{negl}(\lambda).$$

Proof. Similar to proof of Claim 4.3. ■

Claim 5.6. Assuming RSA is secure, for any PPT adversary Att ,

$$\text{Adv}_{\text{Att}}^6 \leq \text{negl}(\lambda).$$

Proof. Similar to proof of Claim 4.4. ■

Using the above claims, we can conclude that any PPT adversary has at most negligible advantage in Game 0, assuming $\mathcal{O}$ is a secure virtual black-box obfuscator for circuit family $\mathcal{C}$, F is a selectively secure puncturable PRF, $\tilde{F}$ is a secure (standard) PRF, PRG is a secure pseudorandom generator, and RSA is secure. Therefore, the construction described in Section 5 is selectively secure with respect to all secure length-qualified signature schemes.

5.1.3 Proof of Lemma 5.1

Proof. Let Att be a PPT adversary such that $\text{Adv}_{\text{Att}}^3 - \text{Adv}_{\text{Att}}^4 = \epsilon$. As in proof of Lemma 4.1, we will assume that $\text{Att} = (\text{Att}_1, \text{Att}_2)$ where Att_1 takes as input the security parameter λ and outputs (m, st) , where st denotes some state information. Att_2 takes as input $m, \text{st}, \text{PP}, \text{VK}$, issues signature queries and finally outputs a forgery.

Let us assume $\text{rnd}_{\text{RSA}} = \text{rnd}_{\text{RSA}}(\lambda)$ bits are used to choose the RSA modulus N , $\text{rnd}_F = \text{rnd}_F(\lambda)$ bits are used by $F.\text{setup}(1^\lambda)$ to choose a PRF key $K \in \mathcal{K}$ and $\text{rnd}_{\text{Att}} = \text{rnd}_{\text{Att}}(\lambda)$ bits are used by Att_1 to compute (m, st) . Let $\mathcal{V}_\lambda = \{(a, r_N, r_K, r_{\text{Att}}) \mid a \in \{0, 1\}^\ell, r_N \in \{0, 1\}^{\text{rnd}_{\text{RSA}}}, r_K \in \{0, 1\}^{\text{rnd}_F}, r_{\text{Att}} \in \{0, 1\}^{\text{rnd}_{\text{Att}}}\}$. For any $v = (a, r_N, r_K, r_{\text{Att}}) \in \mathcal{V}_\lambda$, let N_v denote the RSA modulus generated by r_N , $K_v = F.\text{setup}(1^\lambda; r_K)$ and $(m_v, \text{st}_v) = \text{Att}_1(1^\lambda; r_{\text{Att}})$. Let $\mathcal{C}_{\lambda, v}^0$ denote the family of circuits corresponding to Transform-VBB-1; that is

$$\mathcal{C}_{\lambda, v}^0 = \{\text{Transform-VBB-1}_{N_v, K_v, \alpha, \text{SK}, \tilde{K}} : \alpha = \text{PRG}(a), \text{SK} \in \mathcal{SK}, \text{VK} \in \mathcal{VK}, \tilde{K} \in \tilde{\mathcal{K}}\}.$$

Similarly, $\mathcal{C}_{\lambda, v}^1$ denotes the circuits corresponding to Transform-VBB-2; that is

$$\mathcal{C}_{\lambda, v}^1 = \{\text{Transform-VBB-2}_{y, N_v, K_v, \alpha, \text{SK}, \tilde{K}} : y = \mathcal{S}.\text{Verify}(\text{VK} || m_v, \alpha = \text{PRG}(a), \text{SK} \in \mathcal{SK}, \text{VK} \in \mathcal{VK}, \tilde{K} \in \tilde{\mathcal{K}}\}.$$

When the context is clear, we will drop the dependence of N_v , K_v , m_v and st_v on v . We will now define a PPT algorithm Alg_v that takes as input a circuit $C' \in \mathcal{C}_{\lambda, v}^0 \cup \mathcal{C}_{\lambda, v}^1$, has v hardwired, interacts with Att_2 and outputs a bit b' .

Alg_v:

Inputs: Circuit $C' \in \mathcal{C}_{\lambda,v}^0 \cup \mathcal{C}_{\lambda,v}^1$

Constants: $v = (a, r_N, r_K, r_{\text{Att}}) \in \{0, 1\}^\ell \times \{0, 1\}^{\text{rnd}_{\text{RSA}}} \times \{0, 1\}^{\text{rnd}_F} \times \{0, 1\}^{\text{rnd}_{\text{Att}}}$

1. Compute p, q using r_N , set $N = pq$ and choose $e \leftarrow \mathbb{Z}_{\phi(N)}^*$. Compute $K \leftarrow F.\text{setup}(1^\lambda; r_K)$.
2. Choose $\text{Verify}' \leftarrow \{0, 1\}^{\ell_{\text{ver}}}$, $\text{VK}' \leftarrow \{0, 1\}^{\ell_{\text{vk}}}$, $m' \leftarrow \{0, 1\}^{\ell_{\text{msg}}}$, $\sigma' \leftarrow \{0, 1\}^{\ell_{\text{sig}}}$ and compute $(\text{VK}, \rho) = C'(0, a, \text{Verify}', \text{VK}', m', \sigma')$.
3. Compute $P_2 \leftarrow \mathcal{O}(\text{Transform-Image}_{N,K,e})$. Set $\text{PP} = (C', P_2, e)$ and send $\text{PP}, \text{VK}, m, \text{st}$ to Att_2 .
4. For each signing query x_i , $\mathcal{A}_v$ chooses $\sigma' \leftarrow \{0, 1\}^{\ell_{\text{sig}}}$ computes $C'(0, a, \mathcal{S}.\text{Verify}, \text{VK}, x_i, \sigma') = \sigma_i$ and sends σ_i to Att .
5. Att_2 sends forgery σ_{agg} and n tuples $\{(\text{Verify}_i, \text{VK}_i, m_i)\}$. If $\exists i^* \in [n]$ such that $\text{Verify}_{i^*} = \mathcal{S}.\text{Verify}$, $\text{VK}_{i^*} = \text{VK}$ and $m_{i^*} = m$ and $\text{UniversalVerify}(\text{PP}, \{(\text{Verify}_i, \text{VK}_i, m_i)\}, \sigma_{\text{agg}}) = 1$, then $\mathcal{A}_v$ outputs 1. Else it outputs 0.

Consider the following experiment Exp_v^b : Compute RSA modulus N using r_N , $K = F.\text{setup}(1^\lambda; r_K)$, $\alpha = \text{PRG}(a)$ and $(m, \text{st}) = \text{Att}_1(1^\lambda; r_{\text{Att}})$. Choose $(\text{SK}, \text{VK}) \leftarrow \mathcal{S}.\text{Gen}(1^\lambda)$ and $\tilde{K} \leftarrow \tilde{F}.\text{setup}(1^\lambda)$. If $b = 0$, set $C' \leftarrow \mathcal{O}(\text{Transform-VBB-1}_{N,K,\alpha,\text{SK},\tilde{K}})$, else set $C' \leftarrow \mathcal{O}(\text{Transform-VBB-2}_{y,N,K,\alpha,\text{SK},\tilde{K}})$, where $y = \mathcal{S}.\text{Verify}||\text{VK}||m$. Output $\text{Alg}_v(C')$.

From the definition of Exp_v^0 and Alg_v , it follows that $\Pr[1 \leftarrow \text{Exp}_v^0] = \Pr[\text{Att wins in Game 3}|v]$. Similarly, $\Pr[1 \leftarrow \text{Exp}_v^1] = \Pr[\text{Att wins in Game 4}]$. Hence, $E[\Pr[1 \leftarrow \text{Exp}_v^0] - \Pr[1 \leftarrow \text{Exp}_v^1]] = \epsilon$, where the expectation is over the choice of $v \leftarrow \mathcal{V}_\lambda$. Let $v^* = v^*(\lambda) = \arg \max_{v \in \mathcal{V}} \{\Pr[1 \leftarrow \text{Exp}_v^0] - \Pr[1 \leftarrow \text{Exp}_v^1]\}$. Then, it follows that

$$\Pr[1 \leftarrow \text{Exp}_{v^*}^0] - \Pr[1 \leftarrow \text{Exp}_{v^*}^1] \geq \epsilon. \quad (13)$$

Using Alg_v , we can now define our non-uniform algorithm $\mathcal{A}$. For each security parameter λ , $\mathcal{A}(1^\lambda) = \text{Alg}_{v^*(\lambda)}$.

Now, consider the class of circuits $\mathcal{C}_\lambda = \mathcal{C}_{\lambda,v^*}^0 \cup \mathcal{C}_{\lambda,v^*}^1$. We will require our obfuscator $\mathcal{O}$ to be a virtual black box obfuscator for circuit class $\mathcal{C} = \{\mathcal{C}_\lambda\}_{\lambda \in \mathbb{N}}$.

From the security property of VBB obfuscator, it follows that there exists a PPT simulator S corresponding to $\mathcal{A}$ such that

$$\Pr[\mathcal{A}(\mathcal{O}(C)) = 1] - \Pr[S^C(1^{|C|}) = 1] \leq \text{negl}(\lambda) \quad (14)$$

for all circuits $C \in \mathcal{C}_\lambda$ and the probabilities are over the random coins of $\mathcal{A}$ and S respectively.

Therefore, from Equations 13 and 14, we get the following observation.

Observation 5.1. Let $(\text{SK}, \text{VK}) \leftarrow \mathcal{S}.\text{Gen}(1^\lambda)$ and $\tilde{K} \leftarrow \tilde{F}.\text{setup}(1^\lambda)$. Let $v^* = (a, r_N, r_K, r_{\text{Att}})$, $\alpha = \text{PRG}(a)$. N_{v^*} , K_{v^*} and $(m_{v^*}, \text{st}_{v^*})$ computed using r_N , r_K and r_{Att} respectively, and $y = \mathcal{S}.\text{Verify}||\text{VK}||m_{v^*}$. Let $C_0 = \text{Transform-VBB-1}_{N_{v^*}, K_{v^*}, \alpha, \text{SK}, \tilde{K}}$ and $C_1 = \text{Transform-VBB-2}_{y, N_{v^*}, K_{v^*}, \alpha, \text{SK}, \tilde{K}}$. Then

$$\left| \Pr[S^{C_0}(1^{|C_0|}) = 1] - \Pr[S^{C_1}(1^{|C_1|}) = 1] \right| \geq \epsilon - \text{negl}(\lambda)$$

where the probabilities are over the choice of (SK, VK) , $\tilde{K}$ and the random coins of S .

We will show that this leads to a contradiction. Consider the algorithm $\text{Transform-VBB}'\text{-1}$ which is exactly similar to the circuit Transform-VBB-1 , except that the signature is computed using true randomness instead of using $\tilde{F}$.

Transform-VBB'-1_{N,K,α,SK} :

Inputs: $b \in \{0, 1\}$, $a \in \{0, 1\}^\ell$, $\text{Verify}' \in \{0, 1\}^{\ell_{\text{ver}}}$, $\text{VK}' \in \{0, 1\}^{\ell_{\text{vk}}}$, $m' \in \{0, 1\}^{\ell_{\text{msg}}}$, $\sigma' \in \{0, 1\}^{\ell_{\text{sig}}}$.

Constants : RSA modulus $N \in \mathbb{N}$, $K \in \mathcal{K}$, $\alpha \in \{0, 1\}^{2\ell}$, $\text{SK} \in \mathcal{SK}$.

```

if  $b = 0$  then
  if  $\text{PRG}(a) \neq \alpha$  then
    Output  $\perp$ .
  else
    Choose  $r \in \{0, 1\}^{\ell_{\text{rnd}}}$  and output  $(\text{VK}, \mathcal{S}.\text{Sign}(\text{SK}, m'; r))$ .
  end if
else if  $\text{Verify}'(\text{VK}', m', \sigma') = 0$  then
  Output  $\perp$ .
else
  Output  $F(K, \text{Verify}' || \text{VK}' || m')$ .
end if

```

From the security of $\tilde{F}$, we get the following claim:

Claim 5.7. Let $(\text{SK}, \text{VK}) \leftarrow \mathcal{S}.\text{Sign}(1^\lambda)$ and $\tilde{K} \leftarrow \tilde{F}.\text{setup}(1^\lambda)$. Let $v^* = (a, r_N, r_K, r_{\text{Att}})$, $\alpha = \text{PRG}(a)$. N_{v^*} , K_{v^*} , $(m_{v^*}, \text{st}_{v^*})$ are computed using r_N, r_K, r_{Att} respectively. Let $C_0 = \text{Transform-VBB}'\text{-}1_{N,K,\alpha,\text{SK},\tilde{K}}$ and $C'_0 = \text{Transform-VBB}'\text{-}1_{N,K,\alpha,\text{SK}}$. Assuming $\tilde{F}$ is a secure PRF, for any PPT algorithm S ,

$$\Pr \left[S^{C_0} \left(1^{|C_0|} \right) = 1 \right] - \Pr \left[S^{C'_0} \left(1^{|C'_0|} \right) = 1 \right] \leq \text{negl}(\lambda).$$

Similarly, we define an algorithm Transform-VBB'-2 which is exactly similar to Transform-VBB-2, except that the signature is computed using true randomness.

Claim 5.8. Let $(\text{SK}, \text{VK}) \leftarrow \mathcal{S}.\text{Sign}(1^\lambda)$ and $\tilde{K} \leftarrow \tilde{F}.\text{setup}(1^\lambda)$. Let $v^* = (a, r_N, r_K, r_{\text{Att}})$, $\alpha = \text{PRG}(a)$. N_{v^*} , K_{v^*} , $(m_{v^*}, \text{st}_{v^*})$ are computed using r_N, r_K, r_{Att} respectively and $y = \mathcal{S}.\text{Verify} || \text{VK} || m_{v^*}$. Let $C_1 = \text{Transform-VBB}'\text{-}2_{y,N,K,\alpha,\text{SK},\tilde{K}}$ and $C'_1 = \text{Transform-VBB}'\text{-}2_{y,N,K,\alpha,\text{SK}}$. Assuming $\tilde{F}$ is a secure PRF, for any PPT algorithm S ,

$$\left| \Pr \left[S^{C_1} \left(1^{|C_1|} \right) = 1 \right] - \Pr \left[S^{C'_1} \left(1^{|C'_1|} \right) = 1 \right] \right| \leq \text{negl}(\lambda).$$

Therefore, if we can show that no PPT algorithm can distinguish between C'_0 and C'_1 given only oracle access, then together with Equation 14 and Claims 5.7, 5.8, this leads to a contradiction. Note that if any algorithm S has only oracle access to C'_0 and C'_1 , then in order to distinguish between the two, S must send a query $(1, a', \mathcal{S}.\text{Verify}, \text{VK}, m, \sigma)$ such that $\mathcal{S}.\text{Verify}(\text{VK}, m, \sigma) = 1$. This breaks the security of signature scheme $\mathcal{S}$.

Claim 5.9. Let $(\text{SK}, \text{VK}) \leftarrow \mathcal{S}.\text{Sign}(1^\lambda)$ and $\tilde{K} \leftarrow \tilde{F}.\text{setup}(1^\lambda)$. Let $v^* = (a, r_N, r_K, r_{\text{Att}})$, $\alpha = \text{PRG}(a)$. N_{v^*} , K_{v^*} , $(m_{v^*}, \text{st}_{v^*})$ are computed using r_N, r_K, r_{Att} respectively and $y = \mathcal{S}.\text{Verify} || \text{VK} || m_{v^*}$. Let $C'_0 = \text{Transform-VBB}'\text{-}1_{N,K,\alpha,\text{SK}}$ and $C'_1 = \text{Transform-VBB}'\text{-}2_{y,N,K,\alpha,\text{SK}}$. Assuming $\mathcal{S}$ is a secure signature scheme, for any PPT algorithm S ,

$$\left| \Pr \left[S^{C'_0} \left(1^{|C'_0|} \right) = 1 \right] - \Pr \left[S^{C'_1} \left(1^{|C'_1|} \right) = 1 \right] \right| \leq \text{negl}(\lambda).$$

■

6 Universal Aggregation of Arbitrary Signatures from $i\mathcal{O}$ in the Random Oracle Model

In this section, we describe our n -bounded universal signature aggregator $(\ell_{\text{ver}}, \ell_{\text{vk}}, \ell_{\text{msg}}, \ell_{\text{sig}})$ -UniversalSigAgg. By n -bounded, we mean that at most n signatures can be aggregated.

We will use a secure $(\ell_{\text{ckt}}, \ell_{\text{inp}}, \ell_{\text{out}})$ universal parameters scheme $\mathcal{U} = (\text{UniversalGen}, \text{InduceGen})$ (where the parameters $\ell_{\text{ckt}}, \ell_{\text{inp}}$ and ℓ_{out} will be specified later), an additively homomorphic encryption scheme $(\text{HE.setup}, \text{HE.enc}, \text{HE.dec}, \text{HE.add})$ with message space $\mathbb{F}_p$ for some prime $p > 2^{\ell_{\text{sig}}}$ and ciphertext space $\mathcal{C}_{\text{HE}}$. We will assume each $\text{ct} \in \mathcal{C}_{\text{HE}}$ can be represented using ℓ_{ct} bits. Finally, we will also use a one-way function $f : \{0, 1\}^\ell \rightarrow \{0, 1\}^{2\ell}$ and a secure indistinguishability obfuscator $i\mathcal{O}$.

Our construction consists of three algorithms UniversalSetup, UniversalAgg and UniversalVerify described as follows.

UniversalSetup $(1^\lambda, 1^n)$ Let $(\text{pk}, \text{sk}) \leftarrow \text{HE.setup}(1^\lambda)$. It computes n ciphertexts $\text{ct}_i \leftarrow \text{HE.enc}(\text{pk}, 0)$ and $U \leftarrow \text{UniversalGen}(1^\lambda)$. It sets the public parameters to be $\text{PP} = (\text{pk}, \text{ct}_1, \dots, \text{ct}_n, U)$. Let us assume PP can be represented using ℓ_{pp} bits.

UniversalAgg $(\text{PP} = (\text{pk}, \text{ct}_1, \dots, \text{ct}_n, U), \{\text{Verify}_i, \text{VK}_i, m_i, \sigma_i\}_{i=1}^n)$ We will view each signature σ_i as an integer in $[0, 2^{\ell_{\text{sig}}} - 1]$.

The universal aggregator first checks if all n tuples are distinct. If not, it outputs $\perp$. Else, it computes $t = \sigma_1 \cdot \text{ct}_1 + \dots + \sigma_n \cdot \text{ct}_n$.

Let **AggSetup** be the (randomized) algorithm (defined below) that takes as input security parameter λ , and outputs a program C_{agg} and $\tilde{s} \in \{0, 1\}^{2\ell}$. It uses ℓ_{inp} bits of randomness, and its output has length ℓ_{out} . Let $\mathcal{C}\text{-AggSetup}_{t, \text{PP}, \{\text{Verify}_i, \text{VK}_i, m_i\}_i} \in \{0, 1\}^{\ell_{\text{ckt}}}$ be a string corresponding to canonical description of **AggSetup** $_{t, \text{PP}, \{\text{Verify}_i, \text{VK}_i, m_i\}_i}$. We will assume that given $\mathcal{C}\text{-AggSetup}_{t, \text{PP}, \{\text{Verify}_i, \text{VK}_i, m_i\}_i}$, one can efficiently extract the hardwired constants t , PP and the n tuples $\{\text{Verify}_i, \text{VK}_i, m_i\}_i$.

The aggregator algorithm first computes $(C_{\text{agg}}, \tilde{s}) = \text{InduceGen}(\mathcal{C}\text{-AggSetup}_{t, \text{PP}, \{\text{Verify}_i, \text{VK}_i, m_i\}_i})$. Next, it computes $s = C_{\text{agg}}(\sigma_1, \dots, \sigma_n)$ and outputs $\sigma_{\text{agg}} = (t, s)$.

AggSetup _{$t, \text{PP}, \{\text{Verify}_i, \text{VK}_i, m_i\}_i$} :

Inputs: Security parameter 1^λ , $r \in \{0, 1\}^{\ell_{\text{inp}}}$.

Constants: $t \in \mathcal{C}_{\text{HE}}$, $\text{PP} = (\text{pk}, \text{ct}_1, \dots, \text{ct}_n, U) \in \{0, 1\}^{\ell_{\text{PP}}}$, $\{\text{Verify}_i, \text{VK}_i, m_i\}_i \in (\{0, 1\}^{\ell_{\text{ver}}} \times \{0, 1\}^{\ell_{\text{vk}}} \times \{0, 1\}^{\ell_{\text{msg}}})^n$.

1. Choose $s \leftarrow \{0, 1\}^\ell$ using r .
2. Compute $C_{\text{agg}} \leftarrow i\mathcal{O}(\text{AggSign}_{s, t, \text{PP}, \{\text{Verify}_i, \text{VK}_i, m_i\}_i})$, where **AggSign** is the circuit described below.

AggSign _{$s, t, \text{PP}, \{\text{Verify}_i, \text{VK}_i, m_i\}_i$} :

Inputs: $\sigma_1, \dots, \sigma_n$, where $\sigma_i \in \{0, 1\}^{\ell_{\text{sig}}}$.

Constants: $s \in \{0, 1\}^\ell$, $t \in \mathcal{C}_{\text{HE}}$, $\text{PP} = (\text{pk}, \text{ct}_1, \dots, \text{ct}_n, U)$, $\{\text{Verify}_i, \text{VK}_i, m_i\}_i$.

```

if  $\exists i$  such that  $\text{Verify}_i(\text{VK}_i, m_i, \sigma_i) = 0$  then
  Output  $\perp$ .
end if
if  $t \neq \sigma_1 \cdot \text{ct}_1 + \dots + \sigma_n \cdot \text{ct}_n$  then
  Output  $\perp$ .
end if
Output  $s$ .

```

3. Compute $\tilde{s} = f(s)$.
4. Output $(C_{\text{agg}}, \tilde{s})$.

UniversalVerify($\text{PP} = (\text{pk}, \text{ct}_1, \dots, \text{ct}_n, U), \{\text{Verify}_i, \text{VK}_i, m_i\}_{i=1}^n, \sigma_{\text{agg}} = (t, s')$) The verification algorithm first checks if all n tuples are distinct. If not, it outputs 0. Else, let $\mathcal{C}\text{-AggSetup}$ be the canonical description of **AggSetup** as defined above. It computes $(C_{\text{agg}}, \tilde{s}) = \text{InduceGen}(U, \mathcal{C}\text{-AggSetup}_{t, \text{PP}, \{\text{Verify}_i, \text{VK}_i, m_i\}_i})$. If $\tilde{s} = f(s')$, output 1, else output 0.

Correctness follows directly from the observation that **InduceGen** is a deterministic algorithm.

6.1 Proof of Security

Theorem 6.1. Assuming $i\mathcal{O}$ is a secure indistinguishability obfuscator, (**UniversalGen**, **InduceGen**) is a secure universal parameters scheme in the random oracle model, $\mathcal{HE}$ is a secure additively homomorphic encryption scheme and f is a secure one-way function, for all $(\ell_{\text{ver}}, \ell_{\text{vk}}, \ell_{\text{msg}}, \ell_{\text{sig}})$ -length qualified secure signature schemes $\mathcal{S}$, the bounded universal signature aggregator described in Section 6 is adaptively secure in the random oracle model with respect to $\mathcal{S}$.

We will first describe a sequence of intermediate experiments **Game 0**, $\dots$, **Game 5**, where **Game 0** is the adaptive security game in random oracle model. From **Game 3** onwards, the challenger starts simulating the universal parameters and the responses to random oracle queries. In order to do so, the challenger implements a parameter oracle $\mathcal{O}$, and the simulation algorithms are allowed to make random oracle queries to $\mathcal{O}$. Let us assume the simulator algorithms **SimUGen** and **SimRO** makes at most q_{par} calls to the Parameters Oracle.

6.1.1 Sequence of Games

Game 0: In this game, the challenger first sends PP, VK to the adversary Att. Att then makes polynomially many signature and random oracle queries. Finally, Att outputs forgery σ_{agg} and n tuples $\{\text{Verify}_i, \text{VK}_i, m_i\}_i$.

1. Choose $(\text{SK}, \text{VK}) \leftarrow \mathcal{S}.\text{Gen}(1^\lambda)$, $(\text{pk}, \text{sk}) \leftarrow \text{HE}.\text{setup}(1^\lambda)$ and $U \leftarrow \text{UniversalGen}(1^\lambda)$.
Compute $\text{ct}_i \leftarrow \text{HE}.\text{enc}(\text{pk}, 0)$ for all $i \in [n]$ and set $\text{PP} = (\text{pk}, \text{ct}_1, \dots, \text{ct}_n, U)$.
Send PP, VK to Att.
2. For each signature query x_i , compute $\sigma_i = \mathcal{S}.\text{Sign}(\text{SK}, x_i)$ and send σ_i to Att.
3. For each random oracle query y_i , check if y_i has already been queried.
If yes, let (y_i, α_i) be the tuple corresponding to y_i . Send α_i to Att.
If not, choose $\alpha_i \leftarrow \{0, 1\}^{\ell_{\text{RO}}}$, send α_i to Att and add (y_i, α_i) to table.
4. Finally, Att sends a forgery $\sigma_{\text{agg}} = (t^*, s^*)$ and n tuples $\{\text{Verify}_i, \text{VK}_i, m_i\}_i$. Att wins if
 - (a) $\exists i^*$ such that $\text{Verify}_{i^*} = \mathcal{S}.\text{Verify}$ and $\text{VK}_{i^*} = \text{VK}$,
 - (b) m_{i^*} was not queried during the signing phase,
 - (c) $f(s^*) = \tilde{s}$ and $\text{InduceGen}(U, \mathcal{C}\text{-AggSetup}_{t^*, \text{PP}, \{\text{Verify}_i, \text{VK}_i, m_i\}_i}) = (C, \tilde{s})$.

Game 1: This game is exactly similar to the previous one, except that the challenger guesses a position $i^* \in [n]$, and the attacker wins only if the forgery verifies, and the i^* th tuple corresponds to $\mathcal{S}.\text{Verify}$, VK.

1. Choose $i^* \leftarrow [n]$.
Choose $(\text{SK}, \text{VK}) \leftarrow \mathcal{S}.\text{Gen}(1^\lambda)$, $(\text{pk}, \text{sk}) \leftarrow \text{HE}.\text{setup}(1^\lambda)$ and $U \leftarrow \text{UniversalGen}(1^\lambda)$.
Compute $\text{ct}_i \leftarrow \text{HE}.\text{enc}(\text{pk}, 0)$ and set $\text{PP} = (\text{pk}, \text{ct}_1, \dots, \text{ct}_n, U)$.
Send PP, VK to Att.
2. For each signature query x_i , compute $\sigma_i = \mathcal{S}.\text{Sign}(\text{SK}, x_i)$ and send σ_i to Att.
3. For each random oracle query y_i , check if y_i has already been queried.
If yes, let (y_i, α_i) be the tuple corresponding to y_i . Send α_i to Att.
If not, choose $\alpha_i \leftarrow \{0, 1\}^{\ell_{\text{RO}}}$, send α_i to Att and add (y_i, α_i) to table.
4. Finally, Att sends a forgery $\sigma_{\text{agg}} = (t^*, s^*)$ and n tuples $\{\text{Verify}_i, \text{VK}_i, m_i\}_i$. Att wins if
 - (a) $\text{Verify}_{i^*} = \mathcal{S}.\text{Verify}$ and $\text{VK}_{i^*} = \text{VK}$,
 - (b) m_{i^*} was not queried during the signing phase,
 - (c) $f(s^*) = \tilde{s}$ and $\text{InduceGen}(U, \mathcal{C}\text{-AggSetup}_{t^*, \text{PP}, \{\text{Verify}_i, \text{VK}_i, m_i\}_i}) = (C, \tilde{s})$.

Game 2: In this game, the challenger modifies the public parameters PP. Instead of outputting n encryptions of 0, the challenger outputs an encryption of 1 at position i^* .

1. Choose $i^* \leftarrow [n]$.
Choose $(\text{SK}, \text{VK}) \leftarrow \mathcal{S}.\text{Gen}(1^\lambda)$, $(\text{pk}, \text{sk}) \leftarrow \text{HE}.\text{setup}(1^\lambda)$ and $U \leftarrow \text{UniversalGen}(1^\lambda)$.
Compute $\text{ct}_i \leftarrow \text{HE}.\text{enc}(\text{pk}, 0)$ for all $i \in [n], i \neq i^*$. Let $\text{ct}_{i^*} \leftarrow \text{HE}.\text{enc}(\text{pk}, 1)$.
Set $\text{PP} = (\text{pk}, \text{ct}_1, \dots, \text{ct}_n, U)$.
Send PP, VK to Att.
2. For each signature query x_i , compute $\sigma_i = \mathcal{S}.\text{Sign}(\text{SK}, x_i)$ and send σ_i to Att.
3. For each random oracle query y_i , check if y_i has already been queried.
If yes, let (y_i, α_i) be the tuple corresponding to y_i . Send α_i to Att.
If not, choose $\alpha_i \leftarrow \{0, 1\}^{\ell_{\text{RO}}}$, send α_i to Att and add (y_i, α_i) to table.
4. Finally, Att sends a forgery $\sigma_{\text{agg}} = (t^*, s^*)$ and n tuples $\{\text{Verify}_i, \text{VK}_i, m_i\}_i$. Att wins if
 - (a) $\text{Verify}_{i^*} = \mathcal{S}.\text{Verify}$ and $\text{VK}_{i^*} = \text{VK}$,
 - (b) m_{i^*} was not queried during the signing phase,
 - (c) $f(s^*) = \tilde{s}$ and $\text{InduceGen}(U, \mathcal{C}\text{-AggSetup}_{t^*, \text{PP}, \{\text{Verify}_i, \text{VK}_i, m_i\}_i}) = (C, \tilde{s})$.

Game 3 In this game, the challenger ‘simulates’ both the universal parameters U and the responses to random oracle queries. Let SimUGen and SimRO be the simulation algorithms corresponding to the universal parameters scheme ($\text{UniversalGen}, \text{InduceGen}$). The challenger also implements the Parameters Oracle O . O takes as input a circuit $d \in \mathcal{C}[\ell_{\text{ckt}}, \ell_{\text{inp}}, \ell_{\text{out}}]$. If d has already been queried, O returns the same response. Else, it chooses $r \leftarrow \{0, 1\}^{\ell_{\text{inp}}}$, outputs $d(r)$, and adds $(d, d(r))$ to its table T . Though the parameters oracle O is described in the Setup Phase, it is used in all the later phases as well.

1. Choose $i^* \leftarrow [n]$.
 Choose $(\text{SK}, \text{VK}) \leftarrow \mathcal{S}.\text{Gen}(1^\lambda)$, $(\text{pk}, \text{sk}) \leftarrow \text{HE}.\text{setup}(1^\lambda)$.
 Compute $U \leftarrow \text{SimUGen}(1^\lambda)$.
 Compute $\text{ct}_i \leftarrow \text{HE}.\text{enc}(\text{pk}, 0)$ for all $i \in [n], i \neq i^*$. Let $\text{ct}_{i^*} \leftarrow \text{HE}.\text{enc}(\text{pk}, 1)$.
 Set $\text{PP} = (\text{pk}, \text{ct}_1, \dots, \text{ct}_n, U)$.
Implement the Parameters Oracle O as follows.
 - Maintain a table T . Initially, T is empty.
 - For the i^{th} query $d \in \mathcal{C}[\ell_{\text{ckt}}, \ell_{\text{inp}}, \ell_{\text{out}}]$, check if T contains an entry corresponding to d .
 - If T contains an entry of the form (d, δ) , output δ .
 - Else choose $r \leftarrow \{0, 1\}^{\ell_{\text{inp}}}$ and output $d(r)$. Add $(d, d(r))$ to T .
- Send PP, VK to Att .
2. For each signature query x_i , compute $\sigma_i = \mathcal{S}.\text{Sign}(\text{SK}, x_i)$ and send σ_i to Att .
 3. For each random oracle query y_i , output $\text{SimRO}(y_i)$ ²².
 4. Finally, Att sends a forgery σ_{agg} and n tuples $\{\text{Verify}_i, \text{VK}_i, m_i\}_i$.
Let O-Queries_i denote the set of first i queries to O . Att wins if
 - (a) $\text{Verify}_{i^*} = \mathcal{S}.\text{Verify}$ and $\text{VK}_{i^*} = \text{VK}$,
 - (b) m_{i^*} was not queried during the signing phase,
 - (c) $f(s^*) = \tilde{s}$ and $\text{O}(\mathcal{C}\text{-AggSetup}_{t^*, \text{PP}, \{\text{Verify}_i, \text{VK}_i, m_i\}_i}) = (C, \tilde{s})$.

Recall from Section 6 that $\mathcal{C}\text{-AggSetup}_{t, \text{PP}, \{\text{Verify}_i, \text{VK}_i, m_i\}} \in \{0, 1\}^{\ell_{\text{ckt}}}$ allows efficient extraction of t , PP and $(\text{Verify}_i, \text{VK}_i, m_i)$ for all $i \leq n$. Without loss of generality, we can assume that if Att outputs $\sigma_{\text{agg}} = (t^*, s^*)$ as forgery, along with n tuples $\{\text{Verify}_i, \text{VK}_i, m_i\}_i$, then the circuit $\mathcal{C}\text{-AggSetup}_{t^*, \text{PP}, \{\text{Verify}_i, \text{VK}_i, m_i\}_i}$ was sent as query to the Parameters Oracle O . We will now define games **Game 4-j-a** and **Game 4-j-b** for $j \leq q_{\text{par}}$. Let us first define some notations. Given a canonical circuit $\mathcal{C}\text{-AggSetup}_{t, \text{PP}, \{\text{Verify}_i, \text{VK}_i, m_i\}_i}$, call it (i^*, sk) -rejecting if $\text{Verify}_{i^*}(\text{VK}_{i^*}, m_{i^*}, \text{HE}.\text{dec}(\text{sk}, t)) = 0$. Let Reject-ckt be a circuit of size same as AggSign that outputs $\perp$ for all inputs.

Game 4-j-a

1. Choose $i^* \leftarrow [n]$.
 Choose $(\text{SK}, \text{VK}) \leftarrow \mathcal{S}.\text{Gen}(1^\lambda)$, $(\text{pk}, \text{sk}) \leftarrow \text{HE}.\text{setup}(1^\lambda)$.
 Compute $U \leftarrow \text{SimUGen}(1^\lambda)$.
 Compute $\text{ct}_i \leftarrow \text{HE}.\text{enc}(\text{pk}, 0)$ for all $i \in [n], i \neq i^*$. Let $\text{ct}_{i^*} \leftarrow \text{HE}.\text{enc}(\text{pk}, 1)$.
 Set $\text{PP} = (\text{pk}, \text{ct}_1, \dots, \text{ct}_n, U)$.
 Implement the Parameters Oracle O as follows.
 - Maintain a table T . Initially, T is empty.
 - For the i^{th} query $d \in \mathcal{C}[\ell_{\text{ckt}}, \ell_{\text{inp}}, \ell_{\text{out}}]$, check if T contains an entry corresponding to d .
 - If T contains an entry of the form (d, δ) , output δ .
 - Else if $i \leq j$ and $d = \mathcal{C}\text{-AggSetup}_{t, \text{PP}, \{\text{Verify}_i, \text{VK}_i, m_i\}}$ is (i^*, sk) -rejecting,
output $i\mathcal{O}(\text{Reject-ckt})$ and $f(s)$ for $s \leftarrow \{0, 1\}^\ell$.
 - Else, choose $r \leftarrow \{0, 1\}^{\ell_{\text{inp}}}$ and output $d(r)$. Add $(d, d(r))$ to T .

²²Note that SimRO can make polynomially many queries to O .

Send PP, VK to Att.

2. For each signature query x_i , compute $\sigma_i = \mathcal{S}.\text{Sign}(\text{SK}, x_i)$ and send σ_i to Att.
3. For each random oracle query y_i , output $\text{SimRO}(y_i)$.
4. Finally, Att sends a forgery σ_{agg} and n tuples $\{\text{Verify}_i, \text{VK}_i, m_i\}_i$. Let O-Queries_i denote the set of first i queries to O . Att wins if
 - (a) $\text{Verify}_{i^*} = \mathcal{S}.\text{Verify}$ and $\text{VK}_{i^*} = \text{VK}$,
 - (b) m_{i^*} was not queried during the signing phase,
 - (c) $(\mathcal{C}\text{-AggSetup}_{t^*, \text{PP}, \{\text{Verify}_i, \text{VK}_i, m_i\}})$ is not (i^*, sk) -rejecting) **or** $(\mathcal{C}\text{-AggSetup}_{t^*, \text{PP}, \{\text{Verify}_i, \text{VK}_i, m_i\}} \notin \text{O-Queries}_{j-1})$,
 - (d) $f(s^*) = \tilde{s}$ and $O(\mathcal{C}\text{-AggSetup}_{t^*, \text{PP}, \{\text{Verify}_i, \text{VK}_i, m_i\}_i}) = (C, \tilde{s})$.

Game 4- j - b

1. Choose $i^* \leftarrow [n]$.
 Choose $(\text{SK}, \text{VK}) \leftarrow \mathcal{S}.\text{Gen}(1^\lambda)$, $(\text{pk}, \text{sk}) \leftarrow \text{HE}.\text{setup}(1^\lambda)$.
 Compute $U \leftarrow \text{SimUGen}(1^\lambda)$.
 Compute $\text{ct}_i \leftarrow \text{HE}.\text{enc}(\text{pk}, 0)$ for all $i \in [n], i \neq i^*$. Let $\text{ct}_{i^*} \leftarrow \text{HE}.\text{enc}(\text{pk}, 1)$.
 Set $\text{PP} = (\text{pk}, \text{ct}_1, \dots, \text{ct}_n, U)$.
 Implement the Parameters Oracle O as follows.
 - Maintain a table T . Initially, T is empty.
 - For the i^{th} query $d \in \mathcal{C}[\ell_{\text{ckt}}, \ell_{\text{inp}}, \ell_{\text{out}}]$, check if T contains an entry corresponding to d .
 - If T contains an entry of the form (d, δ) , output δ .
 - Else if $i \leq j$ and $d = \mathcal{C}\text{-AggSetup}_{t, \text{PP}, \{\text{Verify}_i, \text{VK}_i, m_i\}}$ is (i^*, sk) -rejecting, output $i\mathcal{O}(\text{Reject-ckt})$ and $f(s)$ for $s \leftarrow \{0, 1\}^\ell$.
 - Else, choose $r \leftarrow \{0, 1\}^{\ell_{\text{inp}}}$ and output $d(r)$. Add $(d, d(r))$ to T .

Send PP, VK to Att.

2. For each signature query x_i , compute $\sigma_i = \mathcal{S}.\text{Sign}(\text{SK}, x_i)$ and send σ_i to Att.
3. For each random oracle query y_i , output $\text{SimRO}(y_i)$.
4. Finally, Att sends a forgery $\sigma_{\text{agg}} = (t^*, s^*)$ and n tuples $\{\text{Verify}_i, \text{VK}_i, m_i\}_i$. Att wins if
 - (a) $\text{Verify}_{i^*} = \mathcal{S}.\text{Verify}$ and $\text{VK}_{i^*} = \text{VK}$,
 - (b) m_{i^*} was not queried during the signing phase,
 - (c) $(\mathcal{C}\text{-AggSetup}_{t^*, \text{PP}, \{\text{Verify}_i, \text{VK}_i, m_i\}})$ is not (i^*, sk) -rejecting) **or** $(\mathcal{C}\text{-AggSetup}_{t^*, \text{PP}, \{\text{Verify}_i, \text{VK}_i, m_i\}} \notin \text{O-Queries}_j)$,
 - (d) $f(s^*) = \tilde{s}$ and $O(\mathcal{C}\text{-AggSetup}_{t^*, \text{PP}, \{\text{Verify}_i, \text{VK}_i, m_i\}_i}) = (C, \tilde{s})$.

Game 5 This game is exactly similar to Game 4- q_{par} - b .

1. Choose $i^* \leftarrow [n]$.
 Choose $(\text{SK}, \text{VK}) \leftarrow \mathcal{S}.\text{Gen}(1^\lambda)$, $(\text{pk}, \text{sk}) \leftarrow \text{HE}.\text{setup}(1^\lambda)$.
 Compute $U \leftarrow \text{SimUGen}(1^\lambda)$.
 Compute $\text{ct}_i \leftarrow \text{HE}.\text{enc}(\text{pk}, 0)$ for all $i \in [n], i \neq i^*$. Let $\text{ct}_{i^*} \leftarrow \text{HE}.\text{enc}(\text{pk}, 1)$.
 Set $\text{PP} = (\text{pk}, \text{ct}_1, \dots, \text{ct}_n, U)$.
 Implement the Parameters Oracle O as follows.
 - Maintain a table T . Initially, T is empty.
 - For the i^{th} query $d \in \mathcal{C}[\ell_{\text{ckt}}, \ell_{\text{inp}}, \ell_{\text{out}}]$, check if T contains an entry corresponding to d .
 - If T contains an entry of the form (d, δ) , output δ .
 - Else if $d = \mathcal{C}\text{-AggSetup}_{t, \text{PP}, \{\text{Verify}_i, \text{VK}_i, m_i\}}$ is (i^*, sk) -rejecting, output $i\mathcal{O}(\text{Reject-ckt})$ and $f(s)$ for $s \leftarrow \{0, 1\}^\ell$.
 - Else, choose $r \leftarrow \{0, 1\}^{\ell_{\text{inp}}}$ and output $d(r)$. Add $(d, d(r))$ to T .

Send PP, VK to Att.

2. For each signature query x_i , compute $\sigma_i = \mathcal{S}.\text{Sign}(\text{SK}, x_i)$ and send σ_i to Att.

3. For each random oracle query y_i , output $\text{SimRO}(y_i)$.
4. Finally, Att sends a forgery $\sigma_{\text{agg}} = (t^*, s^*)$ and n tuples $\{\text{Verify}_i, \text{VK}_i, m_i\}_i$. Att wins if
 - (a) $\text{Verify}_{i^*} = \mathcal{S}.\text{Verify}$ and $\text{VK}_{i^*} = \text{VK}$,
 - (b) m_{i^*} was not queried during the signing phase,
 - (c) $\mathcal{S}.\text{Verify}(\text{VK}, m_{i^*}, \text{HE}.\text{dec}(\text{sk}, t^*)) = 1$,
 - (d) $f(s^*) = \tilde{s}$ and $O(\mathcal{C}\text{-AggSetup}_{t^*, \text{PP}, \{\text{Verify}_i, \text{VK}_i, m_i\}_i}) = (C, \tilde{s})$.

6.1.2 Analysis

Let $\text{Adv}_{\text{Att}}^j$ denote the advantage of Att in Game j .

Claim 6.1. For any adversary Att,

$$\text{Adv}_{\text{Att}}^1 = \text{Adv}_{\text{Att}}^0/n.$$

Proof. This follows from the definitions of Game 0 and Game 1. The only difference between the two experiments is the change in winning condition, which now includes the guess i^* . This guess is correct with probability $1/n$. $\blacksquare$

Claim 6.2. Assuming $(\text{HE}.\text{setup}, \text{HE}.\text{enc}, \text{HE}.\text{dec})$ is a secure additively homomorphic encryption scheme, for any PPT adversary Att,

$$\text{Adv}_{\text{Att}}^1 - \text{Adv}_{\text{Att}}^2 \leq \text{negl}(\lambda).$$

Proof. Suppose there exists an adversary Att such that $\text{Adv}_{\text{Att}}^1 - \text{Adv}_{\text{Att}}^2 = \epsilon$. We will construct a PPT algorithm $\mathcal{B}$ that breaks the semantic security of HE scheme using Att.

$\mathcal{B}$ receives the public key pk . It sends 0, 1 as challenge messages to the HE challenger, and receives ct in response. It chooses $i^* \leftarrow [n]$, (SK, VK) , computes $n - 1$ encryptions of 0, that is, $\text{ct}_i \leftarrow \text{HE}.\text{enc}(\text{pk}, 0)$ for $i \neq i^*$. It sets $\text{ct}_{i^*} = \text{ct}$. It computes $U \leftarrow \text{UniversalGen}(1^\lambda)$ and sends $\text{PP} = (\text{pk}, \text{ct}_1, \dots, \text{ct}_n, U)$ and VK to Att.

Att then asks for signature/random oracle queries, which $\mathcal{B}$ can simulate perfectly. Finally, Att outputs a forgery σ_{agg} and n tuples $\{\text{Verify}_i, \text{VK}_i, m_i\}$. If Att wins as per the winning conditions (which are the same in both Game 1 and Game 2), output 0, else output 1.

Clearly, if ct is an encryption of 0, then this corresponds to Game 1, else it corresponds to Game 2. This completes our proof. $\blacksquare$

Claim 6.3. Assuming $\mathcal{U} = (\text{UniversalGen}, \text{InduceGen})$ is a secure $(\ell_{\text{ckt}}, \ell_{\text{inp}}, \ell_{\text{out}})$ universal parameters scheme, for any PPT adversary Att,

$$\text{Adv}_{\text{Att}}^2 - \text{Adv}_{\text{Att}}^3 \leq \text{negl}(\lambda).$$

Proof. Suppose there exists a PPT adversary Att such that $\text{Adv}_{\text{Att}}^2 - \text{Adv}_{\text{Att}}^3 = \epsilon$. We will construct a PPT algorithm $\mathcal{A}$ such that $\Pr[\text{Real}^{\mathcal{A}}(1^\lambda) = 1] - \Pr[\text{Ideal}_{\text{SimUGen}, \text{SimRO}}^{\mathcal{A}}(1^\lambda) = 1] = \epsilon$.

$\mathcal{A}$ interacts with Att and participates in either the Real or Ideal game. It receives the universal parameters U . It chooses $(\text{SK}, \text{VK}) \leftarrow \mathcal{S}.\text{Gen}(1^\lambda)$, $(\text{pk}, \text{sk}) \leftarrow \text{HE}.\text{setup}(1^\lambda)$, computes ciphertexts $\text{ct}_1, \dots, \text{ct}_n$ and sets $\text{PP} = (\text{pk}, \text{ct}_1, \dots, \text{ct}_n, U)$. It sends PP, VK to Att.

For the signature queries, $\mathcal{A}$ computes the signatures using SK . For any random oracle query x , it forwards x to the challenger in the Real/Ideal game, and receives either $\text{RO}(x)$ or $\text{SimRO}(x)$. Finally, it receives a forgery $\sigma_{\text{agg}} = (t^*, s^*)$ and n tuples $\{\text{Verify}_i, \text{VK}_i, m_i\}_i$. Note that since there is no Honest Parameter Violation, $\text{InduceGen}(U, \mathcal{C}\text{-AggSetup}_{t^*, \text{PP}, \{\text{Verify}_i, \text{VK}_i, m_i\}_i}) = O(\mathcal{C}\text{-AggSetup}_{t^*, \text{PP}, \{\text{Verify}_i, \text{VK}_i, m_i\}_i})$. Therefore, Game 2 corresponds to $\text{Real}^{\mathcal{A}}(1^\lambda)$ experiment, while Game 3 corresponds to $\text{Ideal}_{\text{SimUGen}, \text{SimRO}}^{\mathcal{A}}(1^\lambda)$. Hence, $\Pr[\text{Real}^{\mathcal{A}}(1^\lambda) = 1] - \Pr[\text{Ideal}_{\text{SimUGen}, \text{SimRO}}^{\mathcal{A}}(1^\lambda) = 1] = \text{Adv}_{\text{Att}}^2 - \text{Adv}_{\text{Att}}^3$. $\blacksquare$

Claim 6.4. Assuming $i\mathcal{O}$ is a secure indistinguishability obfuscator, for any $j \leq q_{\text{par}}$, for any PPT adversary Att,

$$\text{Adv}_{\text{Att}}^{4-(j-1)-b} - \text{Adv}_{\text{Att}}^{4-j-a} \leq \text{negl}(\lambda).$$

Proof. The only difference between Game 4-(j-1)-b and Game 4-j-a is with respect to the j^{th} query to the parameters oracle O . If the j^{th} query is not of the form $\mathcal{C}\text{-AggSetup}_{t, \text{PP}, \{\text{Verify}_i, \text{VK}_i, m_i\}}$, or if it is not (i^*, sk) -rejecting, then both games are identical. Therefore, let us consider the case where the j^{th} query to O is $\mathcal{C}\text{-AggSetup}_{t, \text{PP}, \{\text{Verify}_i, \text{VK}_i, m_i\}}$ for some $t, \{\text{Verify}_i, \text{VK}_i, m_i\}$, and it is (i^*, sk) -rejecting. In Game 4-(j-1)-b, O outputs $(i\mathcal{O}(\text{AggSign}_{t, s, \text{PP}, \{\text{Verify}_i, \text{VK}_i, m_i\}}), f(s))$ while in Game 4-j-a, it outputs $(i\mathcal{O}(\text{Reject-ckt}), f(s))$. Hence, if we can show that $\mathcal{C}\text{-AggSetup}_{t, \text{PP}, \{\text{Verify}_i, \text{VK}_i, m_i\}}$ and Reject-ckt are functionally identical for (i^*, sk) -rejecting circuit, then we can use the security of $i\mathcal{O}$ to prove our claim.

Consider any input $\sigma_1, \dots, \sigma_n$ to $\mathcal{C}\text{-AggSetup}_{t, \text{PP}, \{\text{Verify}_i, \text{VK}_i, m_i\}}$. If $\exists i$ such that $\text{Verify}_i(\text{VK}_i, m_i, \sigma_i) = 0$, then it outputs $\perp$. If $t \neq \sigma_1 \cdot \text{ct}_1 + \dots + \sigma_n \cdot \text{ct}_n$, then it output $\perp$. However, note that if $t = \sigma_1 \cdot \text{ct}_1 + \dots + \sigma_n \cdot \text{ct}_n$, then t is an encryption of σ_{i^*} . Since $\mathcal{C}\text{-AggSetup}_{t, \text{PP}, \{\text{Verify}_i, \text{VK}_i, m_i\}}$ is (i^*, sk) -rejecting, $\text{Verify}_{i^*}(\text{VK}_{i^*}, m_{i^*}, \text{HE.dec}(\text{sk}, t)) = 0$. Therefore, this circuit outputs $\perp$ on all inputs, and is functionally identical to Reject-ckt . $\blacksquare$

Claim 6.5. Assuming f is a secure one way function, for any $j \leq q_{\text{par}}$, for any PPT adversary Att,

$$\text{Adv}_{\text{Att}}^{4-j-a} - \text{Adv}_{\text{Att}}^{4-j-b} \leq \text{negl}(\lambda).$$

Proof. Suppose there exists a PPT adversary Att such that $\text{Adv}_{\text{Att}}^{4-j-a} - \text{Adv}_{\text{Att}}^{4-j-b} = \epsilon$. We will construct a PPT algorithm $\mathcal{B}$ that inverts the one way function f using Att.

Note that the only way an adversary can distinguish between Game 4-j-a and Game 4-j-b is by submitting a forgery $\sigma_{\text{agg}} = (t^*, s^*)$ and n tuples $\{\text{Verify}_i, \text{VK}_i, m_i\}$ such that $\mathcal{C}\text{-AggSetup}_{t, \text{PP}, \{\text{Verify}_i, \text{VK}_i, m_i\}}$ is (i^*, sk) -rejecting and $\mathcal{C}\text{-AggSetup}_{t^*, \text{PP}, \{\text{Verify}_i, \text{VK}_i, m_i\}}$ was sent as j^{th} query to O .

$\mathcal{B}$ receives as input $\tilde{s}$. It chooses $i^* \leftarrow [n]$, chooses $(\text{SK}, \text{VK}) \leftarrow \mathcal{S}.\text{Gen}(1^\lambda)$ and sets PP as in Game 4-j-a and Game 4-j-b. It sends PP, VK to Att. For each signature query x_i , it sends $\sigma_i \leftarrow \mathcal{S}.\text{Sign}(\text{SK}, x_i)$ to Att. For each random oracle query y_i , $\mathcal{B}$ uses SimRO. SimRO, in turn, makes a number of queries to the Parameters Oracle O . If the j^{th} query to O is $\mathcal{C}\text{-AggSetup}_{t, \text{PP}, \{\text{Verify}_i, \text{VK}_i, m_i\}}$ and is (i^*, sk) -rejecting, send $(i\mathcal{O}(\text{Reject-ckt}), \tilde{s})$ as response. All other oracle queries are computed as before. Finally, if Att wins, then $\mathcal{B}$ can use the forgery $\sigma_{\text{agg}} = (t^*, s^*)$ and send s^* as inverse of $\tilde{s}$. $\blacksquare$

Claim 6.6. Assuming $\mathcal{S}$ is a $(\ell_{\text{ver}}, \ell_{\text{vk}}, \ell_{\text{msg}}, \ell_{\text{sig}})$ -length qualified secure signature scheme, for any adversary Att,

$$\text{Adv}_{\text{Att}}^5 \leq \text{negl}(\lambda).$$

Proof. Suppose $\text{Adv}_{\text{Att}}^5 = \epsilon$. We will construct a PPT algorithm $\mathcal{B}$ that breaks the security of $\mathcal{S}$ with advantage ϵ .

$\mathcal{B}$ receives VK from the challenger. It chooses $i^* \leftarrow [n]$, PP as in Game 5 and sends PP.VK to Att. For each signature query x_i sent by Att, $\mathcal{B}$ sends it to the challenger, receives σ_i , which it forwards to Att. It simulates the oracle queries using SimRO, as in Game 5. Finally, Att outputs a forgery $\sigma_{\text{agg}} = (t^*, s^*)$ and n tuples $\{\text{Verify}_i, \text{VK}_i, m_i\}_i$. Att wins if $\text{Verify}_{i^*} = \mathcal{S}.\text{Verify}_{i^*}$, $\text{VK}_{i^*} = \text{VK}$, m_{i^*} was not queried during the signature phase and $\mathcal{S}.\text{Verify}(\text{VK}, m_{i^*}, \text{HE.dec}(\text{sk}, t^*)) = 1$. It sends $(m_{i^*}, \text{HE.dec}(\text{sk}, t^*))$ as forgery. Note that $\mathcal{B}$ wins the signature game if Att wins Game 5. This concludes our proof. $\blacksquare$

Using the above claims, it follows that any PPT adversary has negligible advantage in Game 0, assuming the universal parameters scheme is secure (in the random oracle model), $\mathcal{HE}$ is a secure additively homomorphic encryption scheme and f is a secure one-way function. Therefore, the universal signature aggregator described in Section 6 is adaptively secure with respect to all secure signature schemes in the random oracle model.

7 Universal Aggregation of Arbitrary Signatures from $i\mathcal{O}$ in the Standard Model

In this section, we will describe a construction for an n -bounded universal signature aggregator that can be proven selective secure with respect to all secure length-qualified signature schemes using complexity leveraging. We will use an additively HE scheme $\mathcal{HE}$ with message space $\mathbb{F}_p$ for some prime $p > 2^{\ell_{\text{sig}}}$ and ciphertext space $\mathcal{C}_{\text{HE}}$, where each ciphertext in $\mathcal{C}_{\text{HE}}$ can be represented using ℓ_{ct} bits. We will also use an indistinguishability obfuscator $i\mathcal{O}$, a puncturable pseudorandom function F with key space $\mathcal{K}$, input space $\{0, 1\}^{\ell_{\text{ver}} + \ell_{\text{vk}} + \ell_{\text{msg}} + \log n + \log p}$ and range $\{0, 1\}^\ell$ for $\ell > 2\ell_{\text{ct}}$ and an injective one-way function $f : \{0, 1\}^\ell \rightarrow \{0, 1\}^{2\ell}$. The universal signature aggregator consists of three algorithms **UniversalSetup**, **UniversalAgg** and **UniversalVerify** described below.

UniversalSetup($1^\lambda, 1^n$) The setup algorithm takes λ, n as input, and chooses $(\text{pk}, \text{sk}) \leftarrow \text{HE.setup}(1^\lambda)$. It then computes n encryptions of 0, that is, $\text{ct}_i \leftarrow \text{HE.enc}(\text{pk}, 0)$ for $i \in [n]$.

Let $\sigma_i \in \mathbb{F}_p$ for $i \in [n]$. Let $C_{\sigma_1, \dots, \sigma_n}$ be a circuit that takes as input n bits $x_1, \dots, x_n$ and outputs $\sum \sigma_i x_i \bmod p$. The setup algorithm computes $P_1 = i\mathcal{O}(\text{AggSign}_{K, \text{pk}, \text{ct}_1, \dots, \text{ct}_n})$ and $P_2 = i\mathcal{O}(\text{AggVerify}_K)$, where the programs **AggSign**²³ and **AggVerify**²⁴ are defined below. It outputs $\text{PP} = (P_1, P_2)$.

AggSign _{$K, \text{pk}, \text{ct}_1, \dots, \text{ct}_n$}

Inputs: $\{\text{Verify}_i, \text{VK}_i, m_i, \sigma_i\}_i$.

Constants: PRF Key $K \in \mathcal{K}$, $\text{pk}, (\text{ct}_1, \dots, \text{ct}_n) \in \mathcal{C}_{\text{HE}}^n$.

if $\exists i$ such that $\text{Verify}_i(\text{VK}_i, m_i, \sigma_i) = 0$ **then**

Output $\perp$.

end if

Compute $t = \sigma_1 \cdot \text{ct}_1 + \dots + \sigma_n \cdot \text{ct}_n$.

Let $s_i = F(K, \text{Verify}_i || \text{VK}_i || m_i || i || t)$.

Output $\sigma_{\text{agg}} = (t, \oplus_i s_i)$.

AggVerify _{K}

Inputs: $\{\text{Verify}_i, \text{VK}_i, m_i\}_i, (t^*, s^*) \in \mathcal{C}_{\text{HE}} \times \{0, 1\}^\ell$

Constants: PRF key K

Compute $s = \oplus_i F(K, \text{Verify}_i || \text{VK}_i || m_i || i || t^*)$.

Output 1 if $s = s^*$, else output 0.

UniversalAgg($\text{PP} = (P_1, P_2), \{\text{Verify}_i, \text{VK}_i, m_i, \sigma_i\}_i$) The aggregator algorithm receives as input the public parameters PP and n tuples $\{\text{Verify}_i, \text{VK}_i, m_i, \sigma_i\}_i$. Without loss of generality, we will assume the n tuples are lexicographically ordered. If the n tuples are not distinct, the algorithm outputs $\perp$. Else, it outputs $P_1(\{\text{Verify}_i, \text{VK}_i, m_i, \sigma_i\}_i)$.

UniversalVerify($\text{PP} = (P_1, P_2), \{\text{Verify}_i, \text{VK}_i, m_i\}_{i=1}^n, \sigma_{\text{agg}} = (t^*, s^*)$) Assume the n tuples are sorted in lexicographic order. The verification algorithm checks that the n tuples are distinct. If not, it outputs 0. Else, it outputs $P_2(\{\text{Verify}_i, \text{VK}_i, m_i\}_i, (t^*, s^*))$.

²³Padded to be of same size as **AggSign**-1.

²⁴Padded to be of same size as **AggVerify**-1 and **AggVerify**-2.

7.1 Proof of Security

Let $\mathcal{S}$ be a secure signature scheme. In order to prove the construction in Section 7 selectively secure with respect to $\mathcal{S}$, we will describe a sequence of intermediate hybrid experiments. Looking ahead, there will be an exponential number of intermediate hybrid experiments, and hence we will be using stronger security for the indistinguishability obfuscator $i\mathcal{O}$, the puncturable PRF F and the one way function f .

Theorem 7.1. Let Att be any PPT adversary, and $\mathcal{S}$ a $(\ell_{\text{ver}}, \ell_{\text{vk}}, \ell_{\text{msg}}, \ell_{\text{sig}})$ -length qualified secure signature scheme. Let $\text{Adv}_{\text{Att}, \mathcal{S}}^{\text{sel}}$ denote the advantage of Att in the universal signature aggregator selective security game with respect to $\mathcal{S}$. Let $\text{Adv}^{\mathcal{S}}, \text{Adv}^{\mathcal{HE}}, \text{Adv}^{i\mathcal{O}}, \text{Adv}^F$ and Adv^f denote the maximum advantage of a PPT adversary against signature scheme $\mathcal{S}$, HE scheme $\mathcal{HE}$, indistinguishability obfuscator $i\mathcal{O}$, selectively secure puncturable PRF F and one way function f respectively. Then,

$$\text{Adv}_{\text{Att}, \mathcal{S}}^{\text{sel}} \leq n(\text{Adv}^{\mathcal{HE}} + 2^{\ell_{\text{ct}}} (6\text{Adv}^{i\mathcal{O}} + 2\text{Adv}^F + \text{Adv}^f) + \text{Adv}^{\mathcal{S}})$$

where ℓ_{ct} is the length of ciphertexts in $\mathcal{C}_{\text{HE}}$.

7.1.1 Sequence of Games

Game 0: This corresponds to the selective security game. The challenger receives m^* from Att , chooses $(\text{SK}, \text{VK}) \leftarrow \mathcal{S}.\text{Gen}(1^\lambda)$, the public parameters PP and sends PP, VK to the adversary Att . Att then queries for signatures, which the challenger can compute using SK . Finally, Att outputs forgery σ_{agg} and n tuples $\{\text{Verify}_i, \text{VK}_i, m_i\}$.

1. Att sends message m^* .
2. Choose $(\text{SK}, \text{VK}) \leftarrow \mathcal{S}.\text{Gen}(1^\lambda)$, $(\text{pk}, \text{sk}) \leftarrow \text{HE}.\text{setup}(1^\lambda)$ and $K \leftarrow F.\text{setup}(1^\lambda)$.
Compute $\text{ct}_i \leftarrow \text{HE}.\text{enc}(\text{pk}, 0)$ for all $i \in [n]$ and $P_1 \leftarrow i\mathcal{O}(\text{AggSign}_{K, \text{pk}, \text{ct}_1, \dots, \text{ct}_n})$, $P_2 \leftarrow i\mathcal{O}(\text{AggVerify}_K)$.
Set $\text{PP} = (P_1, P_2)$. Send PP, VK to Att .
3. For each signature query $x_i \neq m^*$, compute $\sigma_i = \mathcal{S}.\text{Sign}(\text{SK}, x_i)$ and send σ_i to Att .
4. Finally, Att sends a forgery $\sigma_{\text{agg}} = (t^*, s^*)$ and n tuples $\{\text{Verify}_i, \text{VK}_i, m_i\}_i$. Att wins
 - (a) $\exists i^*$ such that $\text{Verify}_{i^*} = \mathcal{S}.\text{Verify}$, $\text{VK}_{i^*} = \text{VK}$, $m_{i^*} = m^*$,
 - (b) $\text{AggVerify}_K(\text{PP}, \{\text{Verify}_i, \text{VK}_i, m_i\}, \sigma_{\text{agg}}) = 1$.

Game 1: In this experiment, the challenger chooses $i^* \leftarrow [n]$, and the adversary wins if $\text{Verify}_{i^*} = \mathcal{S}.\text{Verify}$, $\text{VK}_{i^*} = \text{VK}$ and $m_{i^*} = m^*$.

1. Att sends m^* .
2. Choose $i^* \leftarrow [n]$.
Choose $(\text{SK}, \text{VK}) \leftarrow \mathcal{S}.\text{Gen}(1^\lambda)$, $(\text{pk}, \text{sk}) \leftarrow \text{HE}.\text{setup}(1^\lambda)$ and $K \leftarrow F.\text{setup}(1^\lambda)$.
Compute $\text{ct}_i \leftarrow \text{HE}.\text{enc}(\text{pk}, 0)$ for all $i \in [n]$, $P_1 \leftarrow i\mathcal{O}(\text{AggSign}_{K, \text{pk}, \text{ct}_1, \dots, \text{ct}_n})$, $P_2 \leftarrow i\mathcal{O}(\text{AggVerify}_K)$.
Set $\text{PP} = (P_1, P_2)$. Send PP, VK to Att .
3. For each signature query $x_i \neq m^*$, compute $\sigma_i = \mathcal{S}.\text{Sign}(\text{SK}, x_i)$ and send σ_i to Att .
4. Finally, Att sends a forgery $\sigma_{\text{agg}} = (t^*, s^*)$ and n tuples $\{\text{Verify}_i, \text{VK}_i, m_i\}_i$. Att wins
 - (a) $\text{Verify}_{i^*} = \mathcal{S}.\text{Verify}$, $\text{VK}_{i^*} = \text{VK}$, $m_{i^*} = m^*$,
 - (b) $\text{AggVerify}_K(\text{PP}, \{\text{Verify}_i, \text{VK}_i, m_i\}, \sigma_{\text{agg}}) = 1$.

Game 2: This game is similar to the previous one, except that ct_{i^*} is an encryption of 1, instead of 0.

1. Att sends m^* .
2. Choose $i^* \leftarrow [n]$.
Choose $(\text{SK}, \text{VK}) \leftarrow \mathcal{S}.\text{Gen}(1^\lambda)$, $(\text{pk}, \text{sk}) \leftarrow \text{HE}.\text{setup}(1^\lambda)$ and $K \leftarrow F.\text{setup}(1^\lambda)$.
Compute $\text{ct}_i \leftarrow \text{HE}.\text{enc}(\text{pk}, 0)$ for all $i \neq i^*$, $\text{ct}_{i^*} \leftarrow \text{HE}.\text{enc}(\text{pk}, 1)$, $P_1 \leftarrow i\mathcal{O}(\text{AggSign}_{K, \text{pk}, \text{ct}_1, \dots, \text{ct}_n})$, $P_2 \leftarrow i\mathcal{O}(\text{AggVerify}_K)$.
Set $\text{PP} = (P_1, P_2)$. Send PP, VK to Att .

3. For each signature query $x_i \neq m^*$, compute $\sigma_i = \mathcal{S}.\text{Sign}(\text{SK}, x_i)$ and send σ_i to Att.
4. Finally, Att sends a forgery $\sigma_{\text{agg}} = (t^*, s^*)$ and n tuples $\{\text{Verify}_i, \text{VK}_i, m_i\}_i$. Att wins
 - (a) $\text{Verify}_{i^*} = \mathcal{S}.\text{Verify}, \text{VK}_{i^*} = \text{VK}, m_{i^*} = m^*$,
 - (b) $\text{AggVerify}_K(\text{PP}, \{\text{Verify}_i, \text{VK}_i, m_i\}, \sigma_{\text{agg}}) = 1$.

We will now describe an exponential number of hybrid experiments **Game 3, j** for $j \leq 2^{\ell_{\text{ct}}}$. Before describing these intermediate hybrids, we will define some notations. Recall AggVerify_K takes as input tuples of the form $(\{\text{Verify}_i, \text{VK}_i, m_i\}, (t^*, s^*))$. Call such a tuple (i^*, sk) -rejecting if $\text{Verify}_{i^*}(\text{VK}_{i^*}, m_{i^*}, \text{HE}.\text{dec}(\text{sk}, t^*)) = 0$.

Game 3, j : In this game, the adversary does not win if the forgery input $(\{\text{Verify}_i, \text{VK}_i, m_i\}, (t^*, s^*))$ is (i^*, sk) -rejecting and $t^* \leq j$.

1. Att sends m^* .
2. Choose $i^* \leftarrow [n]$.
 Choose $(\text{SK}, \text{VK}) \leftarrow \mathcal{S}.\text{Gen}(1^\lambda)$, $(\text{pk}, \text{sk}) \leftarrow \text{HE}.\text{setup}(1^\lambda)$ and $K \leftarrow F.\text{setup}(1^\lambda)$.
 Compute $\text{ct}_i \leftarrow \text{HE}.\text{enc}(\text{pk}, 0)$ for all $i \neq i^*$, $\text{ct}_{i^*} \leftarrow \text{HE}.\text{enc}(\text{pk}, 1)$, $P_1 \leftarrow i\mathcal{O}(\text{AggSign}_{K, \text{pk}, \text{ct}_1, \dots, \text{ct}_n})$,
 $P_2 \leftarrow i\mathcal{O}(\text{AggVerify}_K)$.
 Set $\text{PP} = (P_1, P_2)$. Send PP, VK to Att.
3. For each signature query $x_i \neq m^*$, compute $\sigma_i = \mathcal{S}.\text{Sign}(\text{SK}, x_i)$ and send σ_i to Att.
4. Finally, Att sends a forgery $\sigma_{\text{agg}} = (t^*, s^*)$ and n tuples $\{\text{Verify}_i, \text{VK}_i, m_i\}_i$. Att wins
 - (a) $\text{Verify}_{i^*} = \mathcal{S}.\text{Verify}, \text{VK}_{i^*} = \text{VK}, m_{i^*} = m^*$,
 - (b) $(\{\text{Verify}_i, \text{VK}_i, m_i\}, (t^*, s^*))$ is not (i^*, sk) -rejecting or $t^* > j$,
 - (c) $\text{AggVerify}_K(\{\text{Verify}_i, \text{VK}_i, m_i\}, \sigma_{\text{agg}}) = 1$.

Game 4: This game is identical to **Game 3, $2^{\ell_{\text{ct}}}$** .

1. Att sends m^* .
2. Choose $i^* \leftarrow [n]$.
 Choose $(\text{SK}, \text{VK}) \leftarrow \mathcal{S}.\text{Gen}(1^\lambda)$, $(\text{pk}, \text{sk}) \leftarrow \text{HE}.\text{setup}(1^\lambda)$ and $K \leftarrow F.\text{setup}(1^\lambda)$.
 Compute $\text{ct}_i \leftarrow \text{HE}.\text{enc}(\text{pk}, 0)$ for all $i \neq i^*$, $\text{ct}_{i^*} \leftarrow \text{HE}.\text{enc}(\text{pk}, 1)$, $P_1 \leftarrow i\mathcal{O}(\text{AggSign}_{K, \text{pk}, \text{ct}_1, \dots, \text{ct}_n})$,
 $P_2 \leftarrow i\mathcal{O}(\text{AggVerify}_K)$.
 Set $\text{PP} = (P_1, P_2)$. Send PP, VK to Att.
3. For each signature query x_i , compute $\sigma_i = \mathcal{S}.\text{Sign}(\text{SK}, x_i)$ and send σ_i to Att.
4. Finally, Att sends a forgery $\sigma_{\text{agg}} = (t^*, s^*)$ and n tuples $\{\text{Verify}_i, \text{VK}_i, m_i\}_i$. Att wins
 - (a) $\text{Verify}_{i^*} = \mathcal{S}.\text{Verify}, \text{VK}_{i^*} = \text{VK}, m_{i^*} = m^*$,
 - (b) $(\{\text{Verify}_i, \text{VK}_i, m_i\}, (t^*, s^*))$ is not (i^*, sk) -rejecting,
 - (c) $\text{AggVerify}_K(\{\text{Verify}_i, \text{VK}_i, m_i\}, \sigma_{\text{agg}}) = 1$.

7.1.2 Analysis

Let $\text{Adv}_{\text{Att}}^j$ denote the advantage of Att in **Game j** .

Claim 7.1. For any adversary Att,

$$\text{Adv}_{\text{Att}}^1 = \text{Adv}_{\text{Att}}^0/n.$$

Proof. This follows from the definitions of **Game 0** and **Game 1**. The only difference between the two experiments is the change in winning condition, which now includes the guess i^* . This guess is correct with probability $1/n$. ■

Claim 7.2. For any PPT adversary Att,

$$\text{Adv}_{\text{Att}}^1 - \text{Adv}_{\text{Att}}^2 \leq \text{Adv}_{\mathcal{HE}}(\lambda).$$

Proof. Suppose there exists an adversary Att such that $\text{Adv}_{\text{Att}}^1 - \text{Adv}_{\text{Att}}^2 = \epsilon$. We will construct a PPT algorithm $\mathcal{B}$ that breaks the semantic security of HE scheme using Att.

$\mathcal{B}$ receives the public key pk . It sends 0, 1 as challenge messages to the HE challenger, and receives ct in response. It chooses $i^* \leftarrow [n]$, (SK, VK) , computes $n - 1$ encryptions of 0, that is, $\text{ct}_i \leftarrow \text{HE.enc}(\text{pk}, 0)$ for $i \neq i^*$. It sets $\text{ct}_{i^*} = \text{ct}$. It chooses $K \leftarrow F.\text{setup}(1^\lambda)$, computes $P_1 \leftarrow i\mathcal{O}(\text{AggSign}_{K, \text{pk}, \text{ct}_1, \dots, \text{ct}_n})$, $P_2 \leftarrow i\mathcal{O}(\text{AggVerify}_K)$ and sends $\text{PP} = (P_1, P_2)$ and VK to Att.

Att then asks for signature/random oracle queries, which $\mathcal{B}$ can simulate perfectly. Finally, Att outputs a forgery σ_{agg} and n tuples $\{\text{Verify}_i, \text{VK}_i, m_i\}$. If Att wins as per the winning conditions (which are the same in both Game 1 and Game 2), output 0, else output 1.

Clearly, if ct is an encryption of 0, then this corresponds to Game 1, else it corresponds to Game 2. This completes our proof. $\blacksquare$

Observation 7.1. For any PPT adversary Att,

$$\text{Adv}_{\text{Att}}^2 = \text{Adv}_{\text{Att}}^{3,0}.$$

Claim 7.3. For any $j < 2^{\ell_{\text{ct}}}$,

$$\text{Adv}_{\text{Att}}^{3,j} - \text{Adv}_{\text{Att}}^{3,j+1} \leq 6\text{Adv}^{i\mathcal{O}} + 2\text{Adv}^F + \text{Adv}^f.$$

Proof. The proof of this claim involves a sequence of intermediate hybrids described below. Note that the only difference between the two hybrids is Step 4b. Both games are identical if $j + 1$ is not (i^*, sk) -rejecting. Hence, we will consider the case where $\mathcal{S}.\text{Verify}(\text{VK}, m^*, \text{HE.dec}(\text{sk}, j + 1)) = 0$.

Game 3, j, a In this game, the challenger uses obfuscations of circuit AggVerify-1 instead of AggVerify . Instead of checking whether $s^* = \oplus_i s_i$, AggVerify-1 uses an injective one way function f to check if $f(s \oplus (\oplus_{i \neq i^*} s_i)) = f(s_{i^*})$.

1. Att sends m^* .
2. Choose $i^* \leftarrow [n]$.
Choose $(\text{SK}, \text{VK}) \leftarrow \mathcal{S}.\text{Gen}(1^\lambda)$, $(\text{pk}, \text{sk}) \leftarrow \text{HE}.\text{setup}(1^\lambda)$ and $K \leftarrow F.\text{setup}(1^\lambda)$.
Compute $\text{ct}_i \leftarrow \text{HE.enc}(\text{pk}, 0)$ for all $i \neq i^*$, $\text{ct}_{i^*} \leftarrow \text{HE.enc}(\text{pk}, 1)$, $P_1 \leftarrow i\mathcal{O}(\text{AggSign}_{K, \text{pk}, \text{ct}_1, \dots, \text{ct}_n})$, $P_2 \leftarrow i\mathcal{O}(\text{AggVerify-1}_K)$.
Set $\text{PP} = (P_1, P_2)$. Send PP , VK to Att.
3. For each signature query $x_i \neq m^*$, compute $\sigma_i = \mathcal{S}.\text{Sign}(\text{SK}, x_i)$ and send σ_i to Att.
4. Finally, Att sends a forgery $\sigma_{\text{agg}} = (t^*, s^*)$ and n tuples $\{\text{Verify}_i, \text{VK}_i, m_i\}_i$. Att wins
 - (a) $\text{Verify}_{i^*} = \mathcal{S}.\text{Verify}$, $\text{VK}_{i^*} = \text{VK}$, $m_{i^*} = m^*$,
 - (b) $(\{\text{Verify}_i, \text{VK}_i, m_i\}, (t^*, s^*))$ is not (i^*, sk) -rejecting or $t^* > j$,
 - (c) $\text{AggVerify-1}_K(\{\text{Verify}_i, \text{VK}_i, m_i\}, \sigma_{\text{agg}}) = 1$.

AggVerify-1_K

Inputs: $\{\text{Verify}_i, \text{VK}_i, m_i\}_i, (t^*, s^*) \in \mathcal{C}_{\text{HE}} \times \{0, 1\}^\ell$

Constants: PRF key K

Compute $\tilde{s} = (\oplus_{i \neq i^*} F(K, \text{Verify}_i || \text{VK}_i || m_i || i || t)) \oplus s^*$.

Output 1 if $f(F(K, \text{Verify}_{i^*} || \text{VK}_{i^*} || m_{i^*} || i^* || t^*)) = f(\tilde{s})$, else output 0.

Game 3, j, b : In this game, **AggSign** and **AggVerify-1** are replaced by **AggSign-1** and **AggVerify-2**. Both the replaced programs use a PRF key punctured at $y = \mathcal{S}.\text{Verify}||\text{VK}||m_{i^*}||i^*||j+1$.

1. Att sends m^* .
2. Choose $i^* \leftarrow [n]$.
 Choose $(\text{SK}, \text{VK}) \leftarrow \mathcal{S}.\text{Gen}(1^\lambda)$, $(\text{pk}, \text{sk}) \leftarrow \text{HE}.\text{setup}(1^\lambda)$ and $K \leftarrow F.\text{setup}(1^\lambda)$.
 Let $y = \mathcal{S}.\text{Verify}||\text{VK}||m^*||i^*||j+1$, $K\{y\} \leftarrow F.\text{puncture}(K, y)$ and $z = f(F(K, y))$.
 Compute $\text{ct}_i \leftarrow \text{HE}.\text{enc}(\text{pk}, 0)$ for all $i \neq i^*$, $\text{ct}_{i^*} \leftarrow \text{HE}.\text{enc}(\text{pk}, 1)$, $P_1 \leftarrow \text{iO}(\text{AggSign-1}_{K\{y\}, \text{pk}, \text{ct}_1, \dots, \text{ct}_n})$,
 $P_2 \leftarrow \text{iO}(\text{AggVerify-2}_{y, K\{y\}, z})$.
 Set $\text{PP} = (P_1, P_2)$. Send PP, VK to Att.
3. For each signature query $x_i \neq m^*$, compute $\sigma_i = \mathcal{S}.\text{Sign}(\text{SK}, x_i)$ and send σ_i to Att.
4. Finally, Att sends a forgery $\sigma_{\text{agg}} = (t^*, s^*)$ and n tuples $\{\text{Verify}_i, \text{VK}_i, m_i\}_i$. Att wins
 - (a) $\text{Verify}_{i^*} = \mathcal{S}.\text{Verify}$, $\text{VK}_{i^*} = \text{VK}$, $m_{i^*} = m^*$,
 - (b) $(\{\text{Verify}_i, \text{VK}_i, m_i\}, (t^*, s^*))$ is not (i^*, sk) -rejecting or $t^* > j$,
 - (c) $\text{AggVerify-2}_{y, K\{y\}, z}(\{\text{Verify}_i, \text{VK}_i, m_i\}, \sigma_{\text{agg}}) = 1$.

AggSign-1 $_{K\{y\}, \text{pk}, \text{ct}_1, \dots, \text{ct}_n}$

Inputs: $\{\text{Verify}_i, \text{VK}_i, m_i, \sigma_i\}_i$.

Constants: PRF Key $K\{y\}, \text{pk}, (\text{ct}_1, \dots, \text{ct}_n) \in \mathcal{C}_{\text{HE}}^n$.

if $\exists i$ such that $\text{Verify}_i(\text{VK}_i, m_i, \sigma_i) = 0$ **then**
 Output $\perp$.
end if
 Compute $t = \sigma_1 \cdot \text{ct}_1 + \dots + \sigma_n \text{ct}_n$.
 Let $s_i = F.\text{eval}(K\{y\}, \text{Verify}_i||\text{VK}_i||m_i||i||t)$.
 Output $\sigma_{\text{agg}} = (t, \oplus_i s_i)$.

AggVerify-2 $_{y, K\{y\}, z}$

Inputs: $\{\text{Verify}_i, \text{VK}_i, m_i\}_i, (t^*, s^*) \in \mathcal{C}_{\text{HE}} \times \{0, 1\}^\ell$

Constants: y , PRF key $K\{y\}$, $z \in \{0, 1\}^{2\ell}$.

Compute $\tilde{s} = (\oplus_{i \neq i^*} F(K, \text{Verify}_i||\text{VK}_i||m_i||i||t)) \oplus s^*$.
if $\text{Verify}_{i^*}||\text{VK}_{i^*}||m_{i^*}||i^*||t^* = y$ **then**
 Output 1 if $z = f(\tilde{s})$, else output 0.
else
 Output 1 if $f(F.\text{eval}(K, \text{Verify}_{i^*}||\text{VK}_{i^*}||m_{i^*}||i^*||t^*)) = f(\tilde{s})$, else output 0.
end if

Game 3, j, c : This game is similar to the previous one, except that z is a uniformly random string.

1. Att sends m^* .
2. Choose $i^* \leftarrow [n]$.
 Choose $(\text{SK}, \text{VK}) \leftarrow \mathcal{S}.\text{Gen}(1^\lambda)$, $(\text{pk}, \text{sk}) \leftarrow \text{HE}.\text{setup}(1^\lambda)$ and $K \leftarrow F.\text{setup}(1^\lambda)$.
 Let $y = \mathcal{S}.\text{Verify}||\text{VK}||m^*||i^*||j+1$, $K\{y\} \leftarrow F.\text{puncture}(K, y)$ and $z' \leftarrow \{0, 1\}^\ell$, $z = f(z')$.
 Compute $\text{ct}_i \leftarrow \text{HE}.\text{enc}(\text{pk}, 0)$ for all $i \neq i^*$, $\text{ct}_{i^*} \leftarrow \text{HE}.\text{enc}(\text{pk}, 1)$, $P_1 \leftarrow \text{iO}(\text{AggSign-1}_{K\{y\}, \text{pk}, \text{ct}_1, \dots, \text{ct}_n})$,
 $P_2 \leftarrow \text{iO}(\text{AggVerify-2}_{y, K\{y\}, z})$.
 Set $\text{PP} = (P_1, P_2)$. Send PP, VK to Att.

3. For each signature query $x_i \neq m^*$, compute $\sigma_i = \mathcal{S}.\text{Sign}(\text{SK}, x_i)$ and send σ_i to Att.
4. Finally, Att sends a forgery $\sigma_{\text{agg}} = (t^*, s^*)$ and n tuples $\{\text{Verify}_i, \text{VK}_i, m_i\}_i$. Att wins
 - (a) $\text{Verify}_{i^*} = \mathcal{S}.\text{Verify}, \text{VK}_{i^*} = \text{VK}, m_{i^*} = m^*$,
 - (b) $(\{\text{Verify}_i, \text{VK}_i, m_i\}, (t^*, s^*))$ is not (i^*, sk) -rejecting or $t^* > j$,
 - (c) $\text{AggVerify-2}_{y, K\{y\}, z}(\{\text{Verify}_i, \text{VK}_i, m_i\}, \sigma_{\text{agg}}) = 1$.

Game 3, j, d : In this game, the challenger modifies the winning condition in Step 4b.

1. Att sends m^* .
2. Choose $i^* \leftarrow [n]$.
 Choose $(\text{SK}, \text{VK}) \leftarrow \mathcal{S}.\text{Gen}(1^\lambda)$, $(\text{pk}, \text{sk}) \leftarrow \text{HE}.\text{setup}(1^\lambda)$ and $K \leftarrow F.\text{setup}(1^\lambda)$.
 Let $y = \mathcal{S}.\text{Verify}[\|\text{VK}\| m^* \| i^* \| j + 1, K\{y\} \leftarrow F.\text{puncture}(K, y)$, $z' \leftarrow \{0, 1\}^\ell$ and $z = f(z')$.
 Compute $\text{ct}_i \leftarrow \text{HE}.\text{enc}(\text{pk}, 0)$ for all $i \neq i^*$, $\text{ct}_{i^*} \leftarrow \text{HE}.\text{enc}(\text{pk}, 1)$, $P_1 \leftarrow i\mathcal{O}(\text{AggSign-1}_{K\{y\}, \text{pk}, \text{ct}_1, \dots, \text{ct}_n})$,
 $P_2 \leftarrow i\mathcal{O}(\text{AggVerify-2}_{y, K\{y\}, z})$.
 Set $\text{PP} = (P_1, P_2, \text{pk}, \text{ct}_1, \dots, \text{ct}_n)$. Send PP, VK to Att.
3. For each signature query $x_i \neq m^*$, compute $\sigma_i = \mathcal{S}.\text{Sign}(\text{SK}, x_i)$ and send σ_i to Att.
4. Finally, Att sends a forgery $\sigma_{\text{agg}} = (t^*, s^*)$ and n tuples $\{\text{Verify}_i, \text{VK}_i, m_i\}_i$. Att wins
 - (a) $\text{Verify}_{i^*} = \mathcal{S}.\text{Verify}, \text{VK}_{i^*} = \text{VK}, m_{i^*} = m^*$,
 - (b) $(\{\text{Verify}_i, \text{VK}_i, m_i\}, (t^*, s^*))$ is not (i^*, sk) -rejecting or $t^* > j + 1$,
 - (c) $\text{AggVerify-2}_{y, K\{y\}, z}(\{\text{Verify}_i, \text{VK}_i, m_i\}, \sigma_{\text{agg}}) = 1$.

Game 3, j, e : In this game, the challenger sets $z = f(F(K, y))$ as in Game 3, j, c .

1. Att sends m^* .
2. Choose $i^* \leftarrow [n]$.
 Choose $(\text{SK}, \text{VK}) \leftarrow \mathcal{S}.\text{Gen}(1^\lambda)$, $(\text{pk}, \text{sk}) \leftarrow \text{HE}.\text{setup}(1^\lambda)$ and $K \leftarrow F.\text{setup}(1^\lambda)$.
 Let $y = \mathcal{S}.\text{Verify}[\|\text{VK}\| m^* \| i^* \| j + 1, K\{y\} \leftarrow F.\text{puncture}(K, y)$, and $z = f(F(K, y))$.
 Compute $\text{ct}_i \leftarrow \text{HE}.\text{enc}(\text{pk}, 0)$ for all $i \neq i^*$, $\text{ct}_{i^*} \leftarrow \text{HE}.\text{enc}(\text{pk}, 1)$, $P_1 \leftarrow i\mathcal{O}(\text{AggSign-1}_{K\{y\}, \text{pk}, \text{ct}_1, \dots, \text{ct}_n})$,
 $P_2 \leftarrow i\mathcal{O}(\text{AggVerify-2}_{y, K\{y\}, z})$.
 Set $\text{PP} = (P_1, P_2)$. Send PP, VK to Att.
3. For each signature query $x_i \neq m^*$, compute $\sigma_i = \mathcal{S}.\text{Sign}(\text{SK}, x_i)$ and send σ_i to Att.
4. Finally, Att sends a forgery $\sigma_{\text{agg}} = (t^*, s^*)$ and n tuples $\{\text{Verify}_i, \text{VK}_i, m_i\}_i$. Att wins
 - (a) $\text{Verify}_{i^*} = \mathcal{S}.\text{Verify}, \text{VK}_{i^*} = \text{VK}, m_{i^*} = m^*$,
 - (b) $(\{\text{Verify}_i, \text{VK}_i, m_i\}, (t^*, s^*))$ is not (i^*, sk) -rejecting or $t^* > j + 1$,
 - (c) $\text{AggVerify-2}_{y, K\{y\}, z}(\{\text{Verify}_i, \text{VK}_i, m_i\}, \sigma_{\text{agg}}) = 1$.

Game 3, j, f : In this game, the challenger uses PRF key K in both AggSign and AggVerify-1 instead of using $K\{y\}$ in AggSign-1 and AggVerify-2 .

1. Att sends m^* .
2. Choose $i^* \leftarrow [n]$.
 Choose $(\text{SK}, \text{VK}) \leftarrow \mathcal{S}.\text{Gen}(1^\lambda)$, $(\text{pk}, \text{sk}) \leftarrow \text{HE}.\text{setup}(1^\lambda)$ and $K \leftarrow F.\text{setup}(1^\lambda)$.
 Compute $\text{ct}_i \leftarrow \text{HE}.\text{enc}(\text{pk}, 0)$ for all $i \neq i^*$, $\text{ct}_{i^*} \leftarrow \text{HE}.\text{enc}(\text{pk}, 1)$, $P_1 \leftarrow i\mathcal{O}(\text{AggSign}_{K, \text{pk}, \text{ct}_1, \dots, \text{ct}_n})$,
 $P_2 \leftarrow i\mathcal{O}(\text{AggVerify-1}_K)$.
 Set $\text{PP} = (P_1, P_2)$. Send PP, VK to Att.
3. For each signature query $x_i \neq m^*$, compute $\sigma_i = \mathcal{S}.\text{Sign}(\text{SK}, x_i)$ and send σ_i to Att.
4. Finally, Att sends a forgery $\sigma_{\text{agg}} = (t^*, s^*)$ and n tuples $\{\text{Verify}_i, \text{VK}_i, m_i\}_i$. Att wins
 - (a) $\text{Verify}_{i^*} = \mathcal{S}.\text{Verify}, \text{VK}_{i^*} = \text{VK}, m_{i^*} = m^*$,
 - (b) $(\{\text{Verify}_i, \text{VK}_i, m_i\}, (t^*, s^*))$ is not (i^*, sk) -rejecting or $t^* > j + 1$,
 - (c) $\text{AggVerify-1}_K(\{\text{Verify}_i, \text{VK}_i, m_i\}, \sigma_{\text{agg}}) = 1$.

We will now relate the difference in Att's advantages in these games to either $\text{Adv}^{i\mathcal{O}}$, Adv^F or Adv^f .

Claim 7.4. For any PPT adversary Att,

$$\text{Adv}_{\text{Att}}^{3,j} - \text{Adv}_{\text{Att}}^{3,j,a} \leq \text{Adv}^{i\mathcal{O}}.$$

Proof. To prove this claim, we need to show that the programs AggVerify_K and AggVerify-1_K are functionally identical. This follows from the observation that f is an injective function, and hence, for any t^*, s^* ,

$$\begin{aligned} s^* &= \oplus_i F(K, \text{Verify}_i \| \text{VK}_i \| m_i \| i \| t^*) \\ \iff (\oplus_{i \neq i^*} F(K, \text{Verify}_i \| \text{VK}_i \| m_i \| i \| t^*)) \oplus s^* &= F(K, \text{Verify}_{i^*} \| \text{VK}_{i^*} \| m_{i^*} \| i^* \| t^*) \\ \iff f((\oplus_{i \neq i^*} F(K, \text{Verify}_i \| \text{VK}_i \| m_i \| i \| t^*)) \oplus s^*) &= f(F(K, \text{Verify}_{i^*} \| \text{VK}_{i^*} \| m_{i^*} \| i^* \| t^*)) \end{aligned}$$

■

Claim 7.5. For any PPT adversary Att,

$$\text{Adv}_{\text{Att}}^{3,j,a} - \text{Adv}_{\text{Att}}^{3,j,b} \leq 2\text{Adv}^{i\mathcal{O}}.$$

Proof. Let $K \leftarrow F.\text{setup}(1^\lambda)$, $y = \mathcal{S}.\text{Verify}[\text{VK}][m^*][i^*][j+1]$, $K\{y\} \leftarrow F.\text{puncture}(K, y)$ and $z = f(F(K, y))$. As in the previous proof, it suffices to show that $\text{AggSign}_{K, \text{pk}, \text{ct}_1, \dots, \text{ct}_n}$ and $\text{AggSign-1}_{K\{y\}, \text{pk}, \text{ct}_1, \dots, \text{ct}_n}$ have identical functionality, and AggVerify-1_K and $\text{AggVerify-2}_{y, K\{y\}, z}$ have identical functionality.

Let us first consider $\text{AggSign}_{K, \text{pk}, \text{ct}_1, \dots, \text{ct}_n}$ and $\text{AggSign-1}_{K\{y\}, \text{pk}, \text{ct}_1, \dots, \text{ct}_n}$. Consider input $\{\text{Verify}_i, \text{VK}_i, m_i, \sigma_i\}_i$. Let $t = \sigma_1 \cdot \text{ct}_1 + \dots + \sigma_n \text{ct}_n$. From the correctness property of puncturable PRFs, it follows that the only case in which $\text{AggSign}_{K, \text{pk}, \text{ct}_1, \dots, \text{ct}_n}$ and $\text{AggSign}_{y, K\{y\}, \text{pk}, \text{ct}_1, \dots, \text{ct}_n}$ can possibly differ is when $\text{Verify}_i(\text{VK}_i, m_i, \sigma_i) = 1$ for all $i \leq n$, $\text{Verify}_{i^*} = \mathcal{S}.\text{Verify}$, $\text{VK}_{i^*} = \text{VK}$, $m_{i^*} = m^*$ and $t = j+1$. But this case is not possible, since $\mathcal{S}.\text{Verify}(\text{VK}, m^*, \text{HE}.\text{dec}(\text{sk}, t)) = \mathcal{S}.\text{Verify}(\text{VK}, m^*, \sigma_{i^*}) = 1$, while $\mathcal{S}.\text{Verify}(\text{VK}, m^*, \text{HE}.\text{dec}(\text{sk}, j+1)) = 0$.

Next, let us consider the programs AggVerify-1_K and $\text{AggVerify}_{y, K\{y\}, z}$. Both programs have identical functionality, because $z = f(F(K, y))$ and for all $y' \neq y$, $F(K, y') = F.\text{eval}(K\{y\}, y')$.

This concludes our proof. ■

Claim 7.6. For any PPT adversary Att,

$$\text{Adv}_{\text{Att}}^{3,j,b} - \text{Adv}_{\text{Att}}^{3,j,c} \leq \text{Adv}^F.$$

Proof. We will construct a PPT algorithm $\mathcal{B}$ such that $\text{Adv}_{\mathcal{B}}^F = \text{Adv}_{\text{Att}}^{3,j,b} - \text{Adv}_{\text{Att}}^{3,j,c}$. $\mathcal{B}$ interacts with Att, and receives m^* . It chooses $i^* \leftarrow [n]$, chooses $(\text{SK}, \text{VK}) \leftarrow \mathcal{S}.\text{Gen}(1^\lambda)$, $(\text{pk}, \text{sk}) \leftarrow \text{HE}.\text{setup}(1^\lambda)$. Next, it computes $\text{ct}_i \leftarrow \text{HE}.\text{enc}(\text{pk}, 0)$ for all $i \neq i^*$, $\text{ct}_{i^*} \leftarrow \text{HE}.\text{enc}(\text{pk}, 1)$. It sends $y = \mathcal{S}.\text{Verify}[\text{VK}][m^*][i^*][j+1]$ to the PRF challenger, and receives $K\{y\}, z'$, where either $z' = F(K, y)$ or $z' \leftarrow \{0, 1\}^{2\ell}$. It computes $z = f(z')$, $P_1 \leftarrow i\mathcal{O}(\text{AggSign-1}_{K\{y\}, \text{pk}, \text{ct}_1, \dots, \text{ct}_n})$, $P_2 \leftarrow i\mathcal{O}(\text{AggVerify-2}_{y, K\{y\}, z})$ and sets $\text{PP} = (P_1, P_2)$. It sends PP, VK to Att.

Next, it receives signature queries, and it computes the signature using SK . Finally, it receives $\sigma_{\text{agg}} = (t^*, s^*)$ and n tuples $\{\text{Verify}_i, \text{VK}_i, m_i\}_i$. If Att wins, it outputs 0, indicating $z' = F(K, y)$. Else, it outputs 1. Since both games have the same winning condition, it follows $\text{Adv}_{\mathcal{B}}^F = \text{Adv}_{\text{Att}}^{3,j,b} - \text{Adv}_{\text{Att}}^{3,j,c}$. ■

Claim 7.7. For any PPT adversary Att,

$$\text{Adv}_{\text{Att}}^{3,j,c} - \text{Adv}_{\text{Att}}^{3,j,d} \leq \text{Adv}^f.$$

Proof. Suppose $\text{Adv}_{\text{Att}}^{3,j,c} - \text{Adv}_{\text{Att}}^{3,j,d} = \epsilon$. Then, with probability ϵ , Att receives PP, VK , sends signature queries, and outputs forgery $\sigma_{\text{agg}} = (j+1, s^*)$ and n tuples $\{\text{Verify}_i, \text{VK}_i, m_i\}_i$ such that $\mathcal{S}.\text{Verify}_{i^*} = \text{Verify}_{i^*}$, $\text{VK}_{i^*} = \text{VK}$, $m_{i^*} = m^*$, the output forgery is (i^*, sk) -rejecting and $\text{AggVerify-2}_{y, K\{y\}, z}(\{\text{Verify}_i, \text{VK}_i, m_i\}, \sigma_{\text{agg}}) = 1$. From the definition of AggVerify-2 , it follows that $f((\oplus_{i \neq i^*} F.\text{eval}(K\{y\}, \text{Verify}_i \| \text{VK}_i \| m_i \| i \| j+1)) \oplus s^*) = z$. Therefore, using Att, we can construct a PPT algorithm $\mathcal{B}$ that breaks the security of one way function f with advantage ϵ . $\mathcal{B}$ receives z from the OWF challenger, and uses it to compute PP as in Game 3, j, c and Game 3, j, d . It sends PP, VK to Att, responds to signature queries, and finally receives forgery $(j+1, s^*)$ ²⁵.

²⁵If $\mathcal{B}$ receives any other forgery, then it simply quits.

and n tuples. It sends $z' = (\oplus_{i \neq i^*} F.\text{eval}(K\{y\}, \text{Verify}_i || \text{VK}_i || m_i || i || j + 1)) \oplus s^*$ to the OWF challenger, and clearly, $\mathcal{B}$ wins if Att wins. This completes our proof. $\blacksquare$

Claim 7.8. For any PPT adversary Att,

$$\text{Adv}_{\text{Att}}^{3,j,d} - \text{Adv}_{\text{Att}}^{3,j,e} \leq \text{Adv}^F.$$

Proof. Similar to the proof of Claim 7.6. $\blacksquare$

Claim 7.9. For any PPT adversary Att,

$$\text{Adv}_{\text{Att}}^{3,j,e} - \text{Adv}_{\text{Att}}^{3,j,f} \leq 2\text{Adv}^{i\mathcal{O}}.$$

Proof. Similar to the proof of Claim 7.5. $\blacksquare$

Claim 7.10. For any PPT adversary Att,

$$\text{Adv}_{\text{Att}}^{3,j,f} - \text{Adv}_{\text{Att}}^{3,j+1} \leq \text{Adv}^{i\mathcal{O}}.$$

Proof. Similar to the proof of Claim 7.4. $\blacksquare$

Summing it up, from the above claims, it follows that for any PPT adversary Att, $\text{Adv}_{\text{Att}}^{3,j} - \text{Adv}_{\text{Att}}^{3,j+1} \leq 6\text{Adv}^{i\mathcal{O}} + 2\text{Adv}^F + \text{Adv}^f$. $\blacksquare$

Claim 7.11. For any PPT adversary Att,

$$\text{Adv}_{\text{Att}}^4 \leq \text{Adv}^{\mathcal{S}}.$$

Proof. Suppose there exists a PPT adversary Att such that $\text{Adv}_{\text{Att}}^4 = \epsilon$. We will construct a PPT algorithm $\mathcal{B}$ that breaks the security of $\mathcal{S}$ with advantage ϵ .

$\mathcal{B}$ interacts with Att and the challenger for $\mathcal{S}$. First, it receives m^* from $\mathcal{S}$ and VK from the challenger. It chooses $i^* \leftarrow [n]$, $(\text{pk}, \text{sk}) \leftarrow \text{HE.setup}(1^\lambda)$, $K \leftarrow F.\text{setup}(1^\lambda)$. It computes $\text{ct}_i \leftarrow \text{HE.enc}(\text{pk}, 0)$ for all $i \neq i^*$, $\text{ct}_{i^*} \leftarrow \text{HE.enc}(\text{pk}, 1)$, $P_1 \leftarrow i\mathcal{O}(\text{AggSign}_{K, \text{pk}, \text{ct}_1, \dots, \text{ct}_n})$ and $P_2 \leftarrow i\mathcal{O}(\text{AggVerify}_K)$. It sends $\text{PP} = (P_1, P_2)$, VK to Att.

For each signature query $x_i \neq m^*$ sent by Att, it forwards x_i to the challenger, and receives σ_i , which it sends to Att.

Finally, Att outputs a forgery $\sigma_{\text{agg}} = (t^*, s^*)$ along with n tuples $\{\text{Verify}_i, \text{VK}_i, m_i\}$. If Att wins in Game 4, then $\text{Verify}_{i^*} = \mathcal{S}.\text{Verify}$, $\text{VK}_{i^*} = \text{VK}$, $m_{i^*} = m^*$ and $(\sigma_{\text{agg}}, \{\text{Verify}_i, \text{VK}_i, m_i\})$ must not be (i^*, sk) -rejecting. In other words, $\mathcal{S}.\text{Verify}(\text{VK}, m^*, \text{HE.dec}(\text{sk}, t^*)) = 1$. $\mathcal{B}$ sends $m^*, \text{HE.dec}(\text{sk}, t^*)$ as a forgery to the challenger. This completes our proof. $\blacksquare$

Summing up, it follows that any adversary Att has advantage at most $n(\text{Adv}_{\text{Att}}^{\mathcal{HE}} + 2^{\ell_\alpha}(6\text{Adv}_{\text{Att}}^{i\mathcal{O}} + 2\text{Adv}_{\text{Att}}^F + \text{Adv}_{\text{Att}}^f) + \text{Adv}_{\text{Att}}^{\mathcal{S}})$ in Game 0, where $\text{Adv}_{\text{Att}}^{\mathcal{HE}}$, $\text{Adv}_{\text{Att}}^{i\mathcal{O}}$, $\text{Adv}_{\text{Att}}^F$, $\text{Adv}_{\text{Att}}^f$ and $\text{Adv}_{\text{Att}}^{\mathcal{S}}$ denote the advantages of Att in the security games for HE scheme $\mathcal{HE}$, indistinguishability obfuscator $i\mathcal{O}$, (selectively secure) puncturable PRF F , one-way function f and signature scheme $\mathcal{S}$ respectively. Therefore, if $2^{\ell_\alpha}(\text{Adv}_{\text{Att}}^{i\mathcal{O}} + \text{Adv}_{\text{Att}}^F + \text{Adv}_{\text{Att}}^f)$ is negligible in λ , then the aggregator scheme described in Section 7 is adaptively secure with respect to all signature schemes $\mathcal{S}$. Note that we require sub-exponential hardness assumption for the indistinguishability obfuscator $i\mathcal{O}$, puncturable PRF F and one-way function f .

References

- [AGH10] Jae Hyun Ahn, Matthew Green, and Susan Hohenberger. Synchronized aggregate signatures: new definitions, constructions and applications. In *Proceedings of the 17th ACM Conference on Computer and Communications Security, CCS 2010, Chicago, Illinois, USA, October 4-8, 2010*, pages 473–484, 2010.
- [BB04] Dan Boneh and Xavier Boyen. Secure identity based encryption without random oracles. In *CRYPTO*, pages 443–459, 2004.
- [BCCT13] Nir Bitansky, Ran Canetti, Alessandro Chiesa, and Eran Tromer. Recursive composition and bootstrapping for SNARKS and proof-carrying data. In *Symposium on Theory of Computing Conference, STOC’13, Palo Alto, CA, USA, June 1-4, 2013*, pages 111–120, 2013.
- [Ben87] Josh Daniel Cohen Benaloh. *Verifiable Secret-ballot Elections*. PhD thesis, Yale University, 1987.
- [BGI⁺01] Boaz Barak, Oded Goldreich, Russell Impagliazzo, Steven Rudich, Amit Sahai, Salil P. Vadhan, and Ke Yang. On the (im)possibility of obfuscating programs. In *CRYPTO*, pages 1–18, 2001.
- [BGI⁺12] Boaz Barak, Oded Goldreich, Russell Impagliazzo, Steven Rudich, Amit Sahai, Salil P. Vadhan, and Ke Yang. On the (im)possibility of obfuscating programs. *J. ACM*, 59(2):6, 2012.
- [BGI13] Elette Boyle, Shafi Goldwasser, and Ioana Ivan. Functional signatures and pseudorandom functions. *IACR Cryptology ePrint Archive*, 2013:401, 2013.
- [BGLS03] Dan Boneh, Craig Gentry, Ben Lynn, and Hovav Shacham. Aggregate and verifiably encrypted signatures from bilinear maps. In *EUROCRYPT*, pages 416–432, 2003.
- [BGOY07] Alexandra Boldyreva, Craig Gentry, Adam O’Neill, and Dae Hyun Yum. Ordered multisignatures and identity-based sequential aggregate signatures, with applications to secure routing. In *Proceedings of the 2007 ACM Conference on Computer and Communications Security, CCS 2007, Alexandria, Virginia, USA, October 28-31, 2007*, pages 276–285, 2007.
- [BKM08] Adam Bender, Jonathan Katz, and Ruggero Morselli. Ring signatures: Stronger definitions, and constructions without random oracles. *J. Cryptol.*, 22(1):114–138, December 2008.
- [BLS01] Dan Boneh, Ben Lynn, and Hovav Shacham. Short signatures from the weil pairing. In *ASIACRYPT*, pages 514–532, 2001.
- [BNN07] Mihir Bellare, Chanathip Namprempre, and Gregory Neven. Unrestricted aggregate signatures. In *ICALP*, pages 411–422, 2007.
- [BR93] Mihir Bellare and Phillip Rogaway. Random oracles are practical: A paradigm for designing efficient protocols. In *ACM Conference on Computer and Communications Security*, pages 62–73, 1993.
- [BR96] Mihir Bellare and Phillip Rogaway. The exact security of digital signatures - how to sign with rsa and rabin. In *EUROCRYPT*, pages 399–416, 1996.
- [BW13] Dan Boneh and Brent Waters. Constrained pseudorandom functions and their applications. In *ASIACRYPT*, pages 280–300, 2013.
- [DJ03] Ivan Damgård and Mads Jurik. A length-flexible threshold cryptosystem with applications. In *Information Security and Privacy, 8th Australasian Conference, ACISP 2003, Wollongong, Australia, July 9-11, 2003, Proceedings*, pages 350–364, 2003.
- [FHPS13] Eduarda S. V. Freire, Dennis Hofheinz, Kenneth G. Paterson, and Christoph Striecks. Programmable hash functions in the multilinear setting. In *CRYPTO*, pages 513–530, 2013.

- [GGH⁺13] Sanjam Garg, Craig Gentry, Shai Halevi, Mariana Raykova, Amit Sahai, and Brent Waters. Candidate indistinguishability obfuscation and functional encryption for all circuits. In *FOCS*, 2013.
- [GM84] S. Goldwasser and S. Micali. Probabilistic encryption. *Jour. of Computer and System Science*, 28(2):270–299, 1984.
- [GMR88] Shafi Goldwasser, Silvio Micali, and Ronald L. Rivest. A digital signature scheme secure against adaptive chosen-message attacks. *SIAM J. Comput.*, 17(2):281–308, 1988.
- [GO93] Shafi Goldwasser and Rafail Ostrovsky. Invariant signatures and non-interactive zero-knowledge proofs are equivalent (extended abstract). In *Proceedings of the 12th Annual International Cryptology Conference on Advances in Cryptology*, CRYPTO '92, pages 228–245, London, UK, UK, 1993. Springer-Verlag.
- [GR06] Craig Gentry and Zulfikar Ramzan. Identity-based aggregate signatures. In *Public Key Cryptography - PKC 2006, 9th International Conference on Theory and Practice of Public-Key Cryptography, New York, NY, USA, April 24-26, 2006, Proceedings*, pages 257–273, 2006.
- [Had00] Satoshi Hada. Zero-knowledge and code obfuscation. In *Proceedings of the 6th International Conference on the Theory and Application of Cryptology and Information Security: Advances in Cryptology*, ASIACRYPT '00, pages 443–457, 2000.
- [HDWH12] Nadia Heninger, Zakir Durumeric, Eric Wustrow, and J. Alex Halderman. Mining your ps and qs: Detection of widespread weak keys in network devices. In *Proceedings of the 21th USENIX Security Symposium, Bellevue, WA, USA, August 8-10, 2012*, pages 205–220, 2012.
- [HJK⁺14] Dennis Hofheinz, Tibor Jager, Dakshita Khurana, Amit Sahai, Brent Waters, and Mark Zhandry. How to generate and use universal parameters. Cryptology ePrint Archive, Report 2014/507, 2014. <http://eprint.iacr.org/>.
- [HSW14] Susan Hohenberger, Amit Sahai, and Brent Waters. Replacing a random oracle: Full domain hash from indistinguishability obfuscation. In *EUROCRYPT*, pages 201–220, 2014.
- [KLMS00] Stephen Kent, Charles Lynn, Joanne Mikkelsen, and Karen Seo. Secure border gateway protocol (s-bgp). *IEEE Journal on Selected Areas in Communications*, 18:103–116, 2000.
- [KPTZ13] Aggelos Kiayias, Stavros Papadopoulos, Nikos Triandopoulos, and Thomas Zacharias. Delegatable pseudorandom functions and applications. In *ACM Conference on Computer and Communications Security*, pages 669–684, 2013.
- [KS98] B. Kaliski and J. Staddon. PKCS #1: RSA Cryptography Specifications Version 2.0. In *RFC Editor*, United States, 1998.
- [LOS⁺06] Steve Lu, Rafail Ostrovsky, Amit Sahai, Hovav Shacham, and Brent Waters. Sequential aggregate signatures and multisignatures without random oracles. In *Advances in Cryptology - EUROCRYPT 2006, 25th Annual International Conference on the Theory and Applications of Cryptographic Techniques, St. Petersburg, Russia, May 28 - June 1, 2006, Proceedings*, pages 465–485, 2006.
- [NS98] David Naccache and Jacques Stern. A new public key cryptosystem based on higher residues. In *Proceedings of the 5th ACM Conference on Computer and Communications Security*, CCS '98, pages 59–66, 1998.
- [OO98] Kazuo Ohta and Tatsuki Okamoto. On concrete security treatment of signatures derived from identification. In *Advances in Cryptology - CRYPTO '98, 18th Annual International Cryptology Conference, Santa Barbara, California, USA, August 23-27, 1998, Proceedings*, pages 354–369, 1998.

- [OU98] Tatsuaki Okamoto and Shigenori Uchiyama. A new public-key cryptosystem as secure as factoring. In *Advances in Cryptology - EUROCRYPT '98, International Conference on the Theory and Application of Cryptographic Techniques, Espoo, Finland, May 31 - June 4, 1998, Proceeding*, pages 308–318, 1998.
- [Pai99] Pascal Paillier. Public-key cryptosystems based on composite degree residuosity classes. In *Proceedings of the 17th International Conference on Theory and Application of Cryptographic Techniques*, EUROCRYPT'99, pages 223–238, 1999.
- [RSA78] Ronald L. Rivest, Adi Shamir, and Leonard M. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Commun. ACM*, 21(2):120–126, 1978.
- [SW14] Amit Sahai and Brent Waters. How to use indistinguishability obfuscation: deniable encryption, and more. In *STOC*, pages 475–484, 2014.
- [ZSN05] Meiyuan Zhao, Sean W. Smith, and David M. Nicol. Aggregated path authentication for efficient bgp security. In *ACM Conference on Computer and Communications Security*, pages 128–138, 2005.

A Universally Aggregating Unique Signatures without the RSA Assumption

In this section we show a modification of our universal aggregation of unique signatures construction and proof from Section 4. The primary difference is that the transformed signature output will be a bit string as opposed to an RSA-type group element in $\mathbb{Z}_N$. Thus, we are able to prove security without using the RSA assumption (but keeping indistinguishability obfuscation and punctured PRF security assumptions.) The primary tradeoff is that the setup must commit to an a-priori bound, n on the number of signatures that can be aggregated. The signature length is independent of n .

We will now describe our n -bounded universal signature aggregator $(\ell_{\text{ver}}, \ell_{\text{vk}}, \ell_{\text{msg}}, \ell_{\text{sig}})$ -UniversalSigAgg. Let ℓ and ℓ_{owf} be polynomials such that $\ell(\lambda) \geq \lambda$. We will use a puncturable PRF F with key space $\mathcal{K}$, punctured key space $\mathcal{K}_p$, domain $\mathcal{X} = \{0, 1\}^{\ell_{\text{ver}}} \times \{0, 1\}^{\ell_{\text{vk}}} \times \{0, 1\}^{\ell_{\text{msg}}}$ and range $\mathcal{Y} = \{0, 1\}^{\ell}$, a one-way function $f : \{0, 1\}^{\ell} \rightarrow \{0, 1\}^{\ell_{\text{owf}}}$ and an indistinguishability obfuscator $i\mathcal{O}$. Our scheme consists of the three algorithms UniversalSetup, UniversalAgg and UniversalVerify.

UniversalSetup($1^\lambda, 1^n$): UniversalSetup takes as input the security parameter λ and a bound n on the number of signatures to be aggregated. It chooses a puncturable PRF key $K \leftarrow F.\text{setup}(1^\lambda)$ and computes obfuscations of the circuits Transform_K and AggVerify_K defined below. It sets the public parameters to be $\text{PP} = (i\mathcal{O}(\text{Transform}_K), i\mathcal{O}(\text{AggVerify}_K))$.

Transform $_K$:

Inputs: $\text{Verify}' \in \{0, 1\}^{\ell_{\text{ver}}}$, $\text{VK}' \in \{0, 1\}^{\ell_{\text{vk}}}$, $m' \in \{0, 1\}^{\ell_{\text{msg}}}$, $\sigma' \in \{0, 1\}^{\ell_{\text{sig}}}$.
 Constants : $K \in \mathcal{K}$.

```

if Verify'(VK', m',  $\sigma'$ ) = 0 then
  Output  $\perp$ .
else
  Output  $F(K, \text{Verify}' || \text{VK}' || m')$ .
end if
```

AggVerify_K :

Inputs: $\{(\text{Verify}_i, \text{VK}_i, m_i)\}_{i=1}^n$ where $(\text{Verify}_i, \text{VK}_i, m_i) \in \{0, 1\}^{\ell_{\text{ver}}} \times \{0, 1\}^{\ell_{\text{vk}}} \times \{0, 1\}^{\ell_{\text{msg}}}$ for all $i \leq n$, $\sigma_{\text{agg}} \in \{0, 1\}^\ell$.

Constants : $K \in \mathcal{K}$.

for all $i \leq n$ **do**

 Compute $s_i = F(K, \text{Verify}_i || \text{VK}_i || m_i)$.

end for

Output 1 if $\oplus_{i=1}^n s_i = \sigma_{\text{agg}}$, 0 otherwise.

UniversalAgg(PP, $\{(\text{Verify}_i, \text{VK}_i, m_i, \sigma_i)\}_{i=1}^n$): Let PP = (P_1, P_2) . **UniversalAgg** first checks that the n tuples are distinct. If not, it outputs $\perp$. Else, it computes $t_i = P_1(\text{Verify}_i, \text{VK}_i, m_i, \sigma_i)$ for each $i \leq n$. If $t_i = \perp$ for some i , then **UniversalAgg** outputs $\perp$, else it outputs $\sigma_{\text{agg}} = \oplus_i t_i$.

UniversalVerify(PP, $\{(\text{Verify}_i, \text{VK}_i, m_i)\}_{i=1}^n, \sigma_{\text{agg}}$): Let PP = (P_1, P_2) . **UniversalVerify** first checks if the n tuples are distinct. If not, it outputs 0. Else, it outputs $P_2(\{(\text{Verify}_i, \text{VK}_i, m_i)\}_{i=1}^n, \sigma_{\text{agg}})$.

A.1 Proof of security

In this section, we will show that our construction is selectively secure with respect to unique signature schemes.

Theorem A.1. Assuming $i\mathcal{O}$ is a secure indistinguishability obfuscator, $(F, F.\text{setup}, F.\text{puncture}, F.\text{eval})$ is a puncturable PRF and f is an injective one way function, for all $(\ell_{\text{ver}}, \ell_{\text{vk}}, \ell_{\text{msg}}, \ell_{\text{sig}})$ -length qualified secure unique signature schemes $\mathcal{S}$, the n -bounded universal signature aggregator $(\ell_{\text{ver}}, \ell_{\text{vk}}, \ell_{\text{msg}}, \ell_{\text{sig}})$ -**UniversalSigAgg** is selectively secure with respect to $\mathcal{S}$.

Let $\mathcal{S} = (\mathcal{S}.\text{Gen}, \mathcal{S}.\text{Sign}, \mathcal{S}.\text{Verify})$ be a secure $(\ell_{\text{ver}}, \ell_{\text{vk}}, \ell_{\text{msg}}, \ell_{\text{sig}})$ -length qualified unique signature scheme, and Att a PPT adversary. Assume Att sends q signing queries during the signing phase. In order to prove this theorem, we will define a sequence of experiments **Game 0**, ..., **Game 4**, where **Game 0** = $\text{Exp}_{\text{Att}, \mathcal{S}}^{\text{sel}}$.

A.1.1 Sequence of Games

Game 0: This game corresponds to $\text{Exp}_{\text{Att}, \mathcal{S}}^{\text{sel}}$. The adversary Att first sends message m , and then receives the verification key and public parameters for the aggregator. Next, the adversary makes signing queries, and finally submits the forgery.

1. Att sends message m .
2. Compute $(\text{SK}, \text{VK}) \leftarrow \mathcal{S}.\text{Gen}(1^\lambda, 1^n)$. Choose $K \leftarrow F.\text{setup}(1^\lambda)$ and set PP = $(i\mathcal{O}(\text{Transform}_K), i\mathcal{O}(\text{AggVerify}_K))$. Send PP, VK to Att.
3. For each signing query $x_i \neq m$, compute $\sigma_i \leftarrow \mathcal{S}.\text{Sign}(\text{SK}, x_i)$ and send σ_i to Att.
4. Att sends forgery σ_{agg} and n tuples $\{(\text{Verify}_i, \text{VK}_i, m_i)\}$. Att wins if $\exists i^* \in [n]$ such that $\text{Verify}_{i^*} = \mathcal{S}.\text{Verify}, \text{VK}_{i^*} = \text{VK}$ and $m_{i^*} = m$ and $\text{UniversalVerify}(\text{PP}, \{(\text{Verify}_i, \text{VK}_i, m_i)\}, \sigma_{\text{agg}}) = 1$.

Game 1: This game is exactly similar to the previous one, except that the program Transform_K is replaced by $\text{Transform}'_K$ which outputs $\perp$ if the input tuples is $(\mathcal{S}.\text{Verify}, \text{VK}, m, \sigma)$ where $\mathcal{S}.\text{Verify}(\text{VK}, m, \sigma) = 1$.

1. Att sends message m .
2. Compute $(\text{SK}, \text{VK}) \leftarrow \mathcal{S}.\text{Gen}(1^\lambda, 1^n)$. Choose $K \leftarrow F.\text{setup}(1^\lambda)$. Set $y = \mathcal{S}.\text{Verify} || \text{VK} || m$ and PP = $(i\mathcal{O}(\text{Transform}'_{y, K}), i\mathcal{O}(\text{AggVerify}_K))$. Send PP, VK to Att.
3. For each signing query $x_i \neq m$, compute $\sigma_i \leftarrow \mathcal{S}.\text{Sign}(\text{SK}, x_i)$ and send σ_i to Att.

4. Att sends forgery σ_{agg} and n tuples $\{(\text{Verify}_i, \text{VK}_i, m_i)\}$. Att wins if $\exists i^* \in [n]$ such that $\text{Verify}_{i^*} = \mathcal{S}.\text{Verify}$, $\text{VK}_{i^*} = \text{VK}$ and $m_{i^*} = m$ and $\text{UniversalVerify}(\text{PP}, \{(\text{Verify}_i, \text{VK}_i, m_i)\}, \sigma_{\text{agg}}) = 1$.

Transform $'_{y,K}$:

Inputs: $\text{Verify}' \in \{0, 1\}^{\ell_{\text{ver}}}$, $\text{VK}' \in \{0, 1\}^{\ell_{\text{vk}}}$, $m' \in \{0, 1\}^{\ell_{\text{msg}}}$, $\sigma' \in \{0, 1\}^{\ell_{\text{sig}}}$.

Constants : $y \in \{0, 1\}^{\ell_{\text{ver}}} \times \{0, 1\}^{\ell_{\text{vk}}} \times \{0, 1\}^{\ell_{\text{msg}}}$, $K \in \mathcal{K}$.

```

if  $\text{Verify}'(\text{VK}', m', \sigma') = 0$  then
    Output  $\perp$ .
else if  $\text{Verify}'||\text{VK}'||m' = y$  then
    Output  $\perp$ .
else
    Output  $F(K, \text{Verify}'||\text{VK}'||m')$ .
end if

```

Game 2: This game is similar to the previous one, except that the programs **Transform $'$** and **AggVerify** are replaced by **Transform-1** and **AggVerify-1** respectively. Each of these programs uses a PRF key punctured at $y = \mathcal{S}.\text{Verify}||\text{VK}||m$.

1. Att sends message m .
2. Compute $(\text{SK}, \text{VK}) \leftarrow \mathcal{S}.\text{Gen}(1^\lambda, 1^n)$. Choose $K \leftarrow F.\text{setup}(1^\lambda)$.
Set $y = \mathcal{S}.\text{Verify}||\text{VK}||m$. Compute $K\{y\} \leftarrow F.\text{puncture}(K, y)$ and $z = F(K, y)$.
Set $\text{PP} = (i\mathcal{O}(\text{Transform-1}_{y,K\{y\}}), i\mathcal{O}(\text{AggVerify-1}_{y,K\{y\},z}))$ and send PP, VK to Att.
3. For each signing query $x_i \neq m$, compute $\sigma_i \leftarrow \mathcal{S}.\text{Sign}(\text{SK}, x_i)$ and send σ_i to Att.
4. Att sends forgery σ_{agg} and n tuples $\{(\text{Verify}_i, \text{VK}_i, m_i)\}$. Att wins if $\exists i^* \in [n]$ such that $\text{Verify}_{i^*} = \mathcal{S}.\text{Verify}$, $\text{VK}_{i^*} = \text{VK}$ and $m_{i^*} = m$ and $\text{UniversalVerify}(\text{PP}, \{(\text{Verify}_i, \text{VK}_i, m_i)\}, \sigma_{\text{agg}}) = 1$.

Transform-1 $_{y,K\{y\}}$:

Inputs: $\text{Verify}' \in \{0, 1\}^{\ell_{\text{ver}}}$, $\text{VK}' \in \{0, 1\}^{\ell_{\text{vk}}}$, $m' \in \{0, 1\}^{\ell_{\text{msg}}}$, $\sigma \in \{0, 1\}^{\ell_{\text{sig}}}$.

Constants : $y \in \{0, 1\}^{\ell_{\text{ver}}} \times \{0, 1\}^{\ell_{\text{vk}}} \times \{0, 1\}^{\ell_{\text{msg}}}$, $K\{y\} \in \mathcal{K}_p$.

```

if  $\text{Verify}'(\text{VK}', m', \sigma') = 0$  then
    Output  $\perp$ .
else if  $\text{Verify}'||\text{VK}'||m' = y$  then
    Output  $\perp$ .
else
    Output  $F.\text{eval}(K\{y\}, \text{Verify}'||\text{VK}'||m')$ .
end if

```

AggVerify-1_{y,K{y},z} :

Inputs: $\{(\text{Verify}_i, \text{VK}_i, m_i)\}_{i=1}^n$ where $(\text{Verify}_i, \text{VK}_i, m_i) \in \{0, 1\}^{\ell_{\text{ver}}} \times \{0, 1\}^{\ell_{\text{vk}}} \times \{0, 1\}^{\ell_{\text{msg}}}$ for all $i \leq n$, $\sigma_{\text{agg}} \in \{0, 1\}^\ell$.

Constants : $y \in \{0, 1\}^{\ell_{\text{ver}}} \times \{0, 1\}^{\ell_{\text{vk}}} \times \{0, 1\}^{\ell_{\text{msg}}}$, $K\{y\} \in \mathcal{K}_p$, $z \in \{0, 1\}^\ell$.

```

for all  $i \leq n$  do
  if  $\text{Verify}_i || \text{VK}_i || m_i = y$  then
     $s_i = z$ 
  else
    Compute  $s_i = F.\text{eval}(K\{y\}, \text{Verify}_i || \text{VK}_i || m_i)$ .
  end if
end for
Output 1 if  $\bigoplus_{i=1}^n s_i = \sigma_{\text{agg}}$ , 0 otherwise.

```

Game 3: In this game, the program AggVerify-1 is replaced by AggVerify-2. As before, a punctured key is used in the program. However, instead of directly checking whether $\sigma_{\text{agg}} = \bigoplus s_i$, AggVerify-2 uses a injective one way function f .

1. Att sends message m .
2. Compute $(\text{SK}, \text{VK}) \leftarrow \mathcal{S}.\text{Gen}(1^\lambda, 1^n)$. Choose $K \leftarrow F.\text{setup}(1^\lambda)$.
Set $y = \mathcal{S}.\text{Verify} || \text{VK} || m$. Compute $K\{y\} \leftarrow F.\text{puncture}(K, y)$ and $z = F(K, y)$.
Compute $w = f(z)$, set $\text{PP} = (i\mathcal{O}(\text{Transform-1}_{y,K\{y\}}), i\mathcal{O}(\text{AggVerify-2}_{y,K\{y\},w}))$ and send PP, VK to Att.
3. For each signing query $x_i \neq m$, compute $\sigma_i \leftarrow \mathcal{S}.\text{Sign}(\text{SK}, x_i)$ and send σ_i to Att.
4. Att sends forgery σ_{agg} and n tuples $\{(\text{Verify}_i, \text{VK}_i, m_i)\}$. Att wins if $\exists i^* \in [n]$ such that $\text{Verify}_{i^*} = \mathcal{S}.\text{Verify}$, $\text{VK}_{i^*} = \text{VK}$ and $m_{i^*} = m$ and $\text{AggVerify-2}_{y,K\{y\},w}(\{(\text{Verify}_i, \text{VK}_i, m_i)\}, \sigma_{\text{agg}}) = 1$.

AggVerify-2_{y,K{y},w} :

Inputs: $\{(\text{Verify}_i, \text{VK}_i, m_i)\}_{i=1}^n$ where $(\text{Verify}_i, \text{VK}_i, m_i) \in \{0, 1\}^{\ell_{\text{ver}}} \times \{0, 1\}^{\ell_{\text{vk}}} \times \{0, 1\}^{\ell_{\text{msg}}}$ for all $i \leq n$, $\sigma_{\text{agg}} \in \{0, 1\}^\ell$.

Constants : $y \in \{0, 1\}^{\ell_{\text{ver}}} \times \{0, 1\}^{\ell_{\text{vk}}} \times \{0, 1\}^{\ell_{\text{msg}}}$, $K\{y\} \in \mathcal{K}_p$, $w \in \{0, 1\}^{\ell_{\text{owf}}}$.

```

Set  $\text{present} = \text{False}$ ,  $\text{pos} = 0$ .
for all  $i \leq n$  do
  if  $\text{Verify}_i || \text{VK}_i || m_i = y$  then
    Set  $\text{present} = \text{True}$ ,  $\text{pos} = i$ .
  else
    Compute  $s_i = F.\text{eval}(K\{y\}, \text{Verify}_i || \text{VK}_i || m_i)$ .
  end if
end for
if  $\text{present} = \text{False}$  then
  Output 1 if  $\bigoplus_{i=1}^n s_i = \sigma_{\text{agg}}$ , 0 otherwise.
else
  Output 1 if  $f(\sigma_{\text{agg}} \oplus_{i \neq \text{pos}} s_i) = w$ , 0 otherwise.
end if

```

Game 4: This game is exactly similar to the previous one, except that z is chosen at random.

1. Att sends message m .

2. Compute $(SK, VK) \leftarrow \mathcal{S}.\text{Gen}(1^\lambda, 1^n)$. Choose $K \leftarrow F.\text{setup}(1^\lambda)$.
Set $y = \mathcal{S}.\text{Verify}||VK||m$. Compute $K\{y\} \leftarrow F.\text{puncture}(K, y)$ and $z \leftarrow \{0, 1\}^\ell$.
Compute $w = f(z)$, set $PP = (i\mathcal{O}(\text{Transform-1}_{y,K\{y\}}), i\mathcal{O}(\text{AggVerify-2}_{y,K\{y\},w}))$ and send PP, VK to Att.
3. For each signing query $x_i \neq m$, compute $\sigma_i \leftarrow \mathcal{S}.\text{Sign}(SK, x_i)$ and send σ_i to Att.
4. Att sends forgery σ_{agg} and n tuples $\{(\text{Verify}_i, VK_i, m_i)\}$. Att wins if $\exists i^* \in [n]$ such that $\text{Verify}_{i^*} = \mathcal{S}.\text{Verify}, VK_{i^*} = VK$ and $m_{i^*} = m$ and $\text{AggVerify-2}_{y,K\{y\},w}(\{(\text{Verify}_i, VK_i, m_i)\}_i, \sigma_{\text{agg}}) = 1$.

A.1.2 Analysis

Let $\text{Adv}_{\text{Att}}^j$ denote the advantage of adversary Att in Game j .

Claim A.1. Assuming $i\mathcal{O}$ is a secure indistinguishability obfuscator and $\mathcal{S}$ is a secure $(\ell_{\text{ver}}, \ell_{\text{vk}}, \ell_{\text{msg}}, \ell_{\text{sig}})$ -length qualified unique signature scheme, for any PPT adversary Att,

$$\text{Adv}_{\text{Att}}^0 - \text{Adv}_{\text{Att}}^1 \leq \text{negl}(\lambda).$$

Proof. The proof of this claim is similar to the one for Lemma 4.1. ■

Claim A.2. Assuming $i\mathcal{O}$ is a secure indistinguishability obfuscator, for any PPT adversary Att,

$$\text{Adv}_{\text{Att}}^1 - \text{Adv}_{\text{Att}}^2 \leq \text{negl}(\lambda).$$

Proof. In order to prove this claim, we will define an intermediate hybrid **Game 1.5** which is exactly same as **Game 1** and **Game 2**, except that the challenger sets $PP = (i\mathcal{O}(\text{Transform-1}_{y,K\{y\}}), i\mathcal{O}(\text{AggVerify}_K))$. We will show (a) **Game 1** and **Game 1.5** are computationally indistinguishable, (b) **Game 1.5** and **Game 2** are computationally indistinguishable.

Proof of (a). Suppose there exists a PPT adversary Att such that $\text{Adv}_{\text{Att}}^1 - \text{Adv}_{\text{Att}}^{1.5} = \epsilon$. We will construct a PPT algorithm $\mathcal{B}$ that constructs two circuits C_0 and C_1 with identical functionality, and uses Att to distinguish between $i\mathcal{O}(C_0)$ and $i\mathcal{O}(C_1)$, thereby breaking the security of $i\mathcal{O}$.

$\mathcal{B}$ receives m from Att, chooses $(SK, VK) \leftarrow \mathcal{S}.\text{Gen}(1^\lambda)$ and $K \leftarrow F.\text{setup}(1^\lambda)$. It sets $y = \mathcal{S}.\text{Verify}||VK||m$ and computes $K\{y\} \leftarrow F.\text{puncture}(K, y)$. It sets $C_0 = \text{Transform}'_{y,K}$ and $C_1 = \text{Transform-1}_{y,K\{y\}}$, and sends C_0, C_1 to the $i\mathcal{O}$ challenger. It receives $C = i\mathcal{O}(C_b)$. $\mathcal{B}$ sets $PP = (C, i\mathcal{O}(\text{AggVerify}_K))$ and sends PP, VK to Att.

Note that $\mathcal{B}$ can respond to the signing queries perfectly, since it has SK . Finally, if Att wins, then $\mathcal{B}$ outputs 0, else it outputs 1. Clearly, if $C = i\mathcal{O}(C_0)$, then it corresponds to **Game 1**, else it corresponds to **Game 1.5**.

To conclude, we need to argue that C_0 and C_1 have identical functionality. This follows from the correctness property of puncturable PRFs. Note that both programs output $\perp$ if one of the input tuples is $(\mathcal{S}.\text{Verify}, VK, m, \sigma)$. For all other tuples $(\text{Verify}', VK', m', \sigma')$, $F(K, \text{Verify}'||VK'||m') = F.\text{eval}(K\{y\}, \text{Verify}'||VK'||m')$. This completes the first step of our proof.

Proof of (b). The second step (showing that **Game 1.5** and **Game 2** are computationally indistinguishable) follows along similar lines. ■

Claim A.3. Assuming $i\mathcal{O}$ is a secure indistinguishability obfuscator and f is an injective function, for any PPT adversary Att,

$$\text{Adv}_{\text{Att}}^2 - \text{Adv}_{\text{Att}}^3 \leq \text{negl}(\lambda).$$

Proof. Suppose there exists a PPT adversary Att such that $\text{Adv}_{\text{Att}}^2 - \text{Adv}_{\text{Att}}^3 = \epsilon$. As in the previous proof, we will construct a PPT algorithm $\mathcal{B}$ that constructs two circuits C_0 and C_1 with identical functionality, and uses Att to distinguish between $i\mathcal{O}(C_0)$ and $i\mathcal{O}(C_1)$, thereby breaking the security of $i\mathcal{O}$.

The only difference between **Game 2** and **Game 3** is that in **Game 2**, circuit $\text{AggVerify-1}_{y,K\{y\}.z}$ is used, while in **Game 3**, circuit $\text{AggVerify-2}_{y,K\{y\},w}$ is used. $\mathcal{B}$ interacts with Att and receives m . It chooses

$(SK, VK) \leftarrow \mathcal{S}.\text{Gen}(1^\lambda)$ and $K \leftarrow F.\text{setup}(1^\lambda)$. Next, it computes a key punctured at $y = \mathcal{S}.\text{Verify}||VK||m$, i.e. $K\{y\} \leftarrow F.\text{puncture}(K, y)$ and sets $z = F(K, y)$ and $w = f(z)$. Given $y, K\{y\}, z, w$, $\mathcal{B}$ can now construct circuits $C_0 = \text{AggVerify-1}_{y, K\{y\}, z}$ and $C_1 = \text{AggVerify-2}_{y, K\{y\}, w}$. $\mathcal{B}$ sends C_0 and C_1 to the $i\mathcal{O}$ challenger, and receives $C = i\mathcal{O}(C_b)$. $\mathcal{B}$ sets $PP = (i\mathcal{O}(\text{Transform-1}_{y, K\{y\}}), C)$ and sends PP, VK to Att .

$\mathcal{B}$ now responds to signing queries using SK . Finally Att sends forgery σ_{agg} , along with n tuples $\{(\text{Verify}_i, VK_i, m_i)\}$. If $C(\{\text{Verify}_i, VK_i, m_i\}, \sigma_{\text{agg}}) = 1$, output 0, else output 1. Clearly, if $C = i\mathcal{O}(C_0)$, then this corresponds to **Game 2**, else it corresponds to **Game 3**. Therefore, all that remains is to prove that C_0 and C_1 have identical functionality.

Consider any input $(\{(\text{Verify}_i, VK_i, m_i)\}, \sigma_{\text{agg}})$. If there is no i^* such that $\text{Verify}_{i^*}||VK_{i^*}||m_{i^*} = y$, then both circuits check if $\sigma_{\text{agg}} = \oplus_i F(K\{y\}, \text{Verify}_i||VK_i||m_i)$. If there exists an $i^* \in [n]$ such that $\text{Verify}_{i^*}||VK_{i^*}||m_{i^*} = y$, then C_0 accepts iff

$$\begin{aligned} & (\oplus_{i \neq i^*} F(K\{y\}, \text{Verify}_i||VK_i||m_i)) \oplus F(K, y) = \sigma_{\text{agg}} \\ \iff & (\oplus_{i \neq i^*} F(K\{y\}, \text{Verify}_i||VK_i||m_i)) \oplus \sigma_{\text{agg}} = F(K, y) = z \\ \iff & f((\oplus_{i \neq i^*} F(K\{y\}, \text{Verify}_i||VK_i||m_i)) \oplus \sigma_{\text{agg}}) = f(z) = w. \end{aligned}$$

The last equivalence follows from the fact that f is an injective function. However, note that the last statement is the condition for C_1 accepting. This proves that both C_0 and C_1 have identical functionality, which proves our claim. $\blacksquare$

Claim A.4. Assuming F is a puncturable PRF, for any PPT adversary Att ,

$$\text{Adv}_{\text{Att}}^3 - \text{Adv}_{\text{Att}}^4 \leq \text{negl}(\lambda).$$

Proof. Suppose there exists a PPT adversary Att such that $\text{Adv}_{\text{Att}}^3 - \text{Adv}_{\text{Att}}^4 = \epsilon$. We will construct a PPT algorithm $\mathcal{B}$ that uses Att to break the security of puncturable PRF $(F, F.\text{setup}, F.\text{puncture}, F.\text{eval})$ with advantage ϵ .

First, $\mathcal{B}$ receives the message m from Att . As in **Game 3** and **Game 4**, it computes $(SK, VK) \leftarrow \mathcal{S}.\text{Gen}(1^\lambda)$. Next, it sends $y = \mathcal{S}.\text{Verify}||VK||m$ as the challenge to the PRF challenger. $\mathcal{B}$ receives a punctured key $K\{y\}$ and $z \in \{0, 1\}^\ell$, where $z = F(K, y)$ or $z \leftarrow \{0, 1\}^\ell$. $\mathcal{B}$ computes $w = f(z)$ and sets the public parameters $PP = (i\mathcal{O}(\text{Transform-1}_{y, K\{y\}}), i\mathcal{O}(\text{AggVerify-2}_{y, K\{y\}, w}))$. It sends PP, VK to Att .

The signing phase and forgery phase are exactly similar in **Game 3** and **Game 4**. For each signing query x_i , $\mathcal{B}$ sends $\mathcal{S}.\text{Sign}(SK, x_i)$ to Att . Finally, Att outputs the forgery σ_{agg} and n tuples $\{(\text{Verify}_i, VK_i, m_i)\}$.

Note that if $z = F(K, y)$, then $\mathcal{B}$ simulates **Game 3** perfectly. If $z \leftarrow \{0, 1\}^\ell$, $\mathcal{B}$ simulates **Game 4** perfectly. This concludes our proof. $\blacksquare$

Claim A.5. Assuming f is a one way function, for any PPT adversary Att ,

$$\text{Adv}_{\text{Att}}^4 \leq \text{negl}(\lambda).$$

Proof. Suppose there exists a PPT adversary Att such that $\text{Adv}_{\text{Att}}^4 = \epsilon$. We will construct a PPT algorithm $\mathcal{B}$ that, using Att , inverts the one way function f with probability ϵ .

$\mathcal{B}$ receives w from the one way function challenger and m from Att . It chooses $(SK, VK) \leftarrow \mathcal{S}.\text{Gen}(1^\lambda)$, $K \leftarrow F.\text{setup}(1^\lambda)$ and computes $K\{y\} \leftarrow F.\text{puncture}(K, y)$ (where $y = \mathcal{S}.\text{Verify}||VK||m$). It sets the public parameters $PP = (i\mathcal{O}(\text{Transform-1}_{y, K\{y\}}), i\mathcal{O}(\text{AggVerify-2}_{y, K\{y\}, w}))$ and sends PP, VK to Att .

For each signing query x_i , it computes $\mathcal{S}.\text{Sign}(SK, x_i)$.

Finally, $\mathcal{B}$ receives $\sigma_{\text{agg}} \in \{0, 1\}^\ell$ and n tuples $\{(\text{Verify}_i, VK_i, m_i)\}$. If $\text{AggVerify-2}_{y, K\{y\}, w}(\{\text{Verify}_i, VK_i, m_i\}, \sigma_{\text{agg}}) = 1$ and $\exists i^*$ such that $\text{Verify}_{i^*}||VK_{i^*}||m_{i^*} = y$, then $\mathcal{B}$ can successfully find an inverse for w . $\mathcal{B}$ computes $s_i = F(K, \text{Verify}_i||VK_i||m_i)$ for $i \neq i^*$ and sends $\sigma_{\text{agg}} \oplus_{i \neq i^*} s_i$ to the one way function challenger. Clearly, if Att wins in **Game 4**, then $\mathcal{B}$ inverts the one way function. $\blacksquare$

To conclude, it follows from the above claims that any PPT adversary has at most negligible advantage in **Game 0** (assuming $i\mathcal{O}$, F and f are secure), and therefore the n -bounded aggregator described in A is selectively secure with respect to secure unique signature schemes.

B Making our VBB proof Adaptively Secure

We now show how a minor adaptation of our selectively secure universal aggregator from VBB of Section 5 can be proven adaptively secure. The primary change is to first hash every message with an “admissible hash function” introduced by Boneh and Boyen [BB04]. From there the additional RSA-type techniques needed fall in line with those used by Hohenberger, Sahai and Waters [HSW14].

We now describe our construction and proof. Let $\mathcal{O}$ be a virtual black-box obfuscator, $\tilde{F}$ a secure PRF with key space $\tilde{K}$, domain $\{0, 1\}^{\ell_{\text{sig}}}$ and range $\{0, 1\}^{\ell_{\text{rnd}}}$, PRG a secure pseudorandom generator and h a θ -admissible hash function mapping ℓ_{msg} bits to d_1 bits. Let $d_2 = d_1 + \ell_{\text{ver}} + \ell_{\text{vk}}$. Our universal signature aggregator $(\ell_{\text{ver}}, \ell_{\text{vk}}, \ell_{\text{msg}}, \ell_{\text{sig}})$ -UniversalSigAgg consists of three algorithms UniversalSetup, UniversalAgg and UniversalVerify described below.

UniversalSetup(1^λ): The setup algorithm first chooses an RSA modulus N , $v \in \mathbb{Z}_N^*$ and $e \in \mathbb{Z}_{\phi(N)}^*$. Next, it chooses $2d_2$ constants $c_{i,b} \leftarrow \mathbb{Z}_{\phi(N)}$. Let $\mathbf{c} = \{c_{i,b}\}_{i \in [d_2], b \in \{0,1\}}$. It sets $\text{PP} = (\mathcal{O}(\text{Transform}_{N,v,\mathbf{c}}), \mathcal{O}(\text{Transform-Image}_{N,v,\mathbf{c},e}), N, e)$, where Transform²⁶ and Transform-Image²⁷ are as follows.

Transform $_{N,v,\mathbf{c}}$:

Inputs: $b \in \{0, 1\}$, $a' \in \{0, 1\}^\ell$, $\text{Verify}' \in \{0, 1\}^{\ell_{\text{ver}}}$, $\text{VK}' \in \{0, 1\}^{\ell_{\text{vk}}}$, $m' \in \{0, 1\}^{\ell_{\text{msg}}}$, $\sigma' \in \{0, 1\}^{\ell_{\text{sig}}}$.

Constants : RSA modulus $N \in \mathbb{N}$, $v \in \mathbb{Z}_N^*$, $\mathbf{c} = \{c_{i,b}\} \in \mathbb{Z}_{\phi(N)}^{2d_2}$.

if $b = 0$ then

 Output $\perp$.

else if $\text{Verify}'(\text{VK}', m', \sigma') = 0$ then

 Output $\perp$.

else

 Compute $h(m') = x$ and let $z = x || \text{Verify}' || \text{VK}'$.

 Output $v \prod_i c_{i,z_i} \pmod{N}$.

end if

Transform-Image $_{N,v,\mathbf{c},e}$:

Inputs: $\text{Verify}' \in \{0, 1\}^{\ell_{\text{ver}}}$, $\text{VK}' \in \{0, 1\}^{\ell_{\text{vk}}}$, $m' \in \{0, 1\}^{\ell_{\text{msg}}}$.

Constants : RSA modulus $N \in \mathbb{N}$, $v \in \mathbb{Z}_N^*$, $\mathbf{c} = \{c_{i,b}\} \in \mathbb{Z}_{\phi(N)}^{2d_2}$, $e \in \mathbb{Z}_{\phi(N)}^*$.

 Compute $h(m') = x$ and let $z = x || \text{Verify}' || \text{VK}'$.

 Output $(v \prod_i c_{i,z_i})^e \pmod{N}$.

UniversalAgg($\text{PP}, \{(\text{Verify}_i, \text{VK}_i, m_i, \sigma_i)\}_{i=1}^n$): Let $\text{PP} = (P_1, P_2, N, e)$. UniversalAgg first checks if the n tuples are distinct. If not, it outputs $\perp$. Else, it computes $t_i = P_1(\text{Verify}_i, \text{VK}_i, m_i, \sigma_i)$ for each $i \leq n$. If $t_i = \perp$ for some i , then UniversalAgg outputs $\perp$, else it outputs $\sigma_{\text{agg}} = \prod_i t_i \pmod{N}$.

UniversalVerify($\text{PP}, \{(\text{Verify}_i, \text{VK}_i, m_i)\}_{i=1}^n, \sigma_{\text{agg}}$): Let $\text{PP} = (P_1, P_2, N, e)$. UniversalVerify first checks if all n tuples are distinct. If not, it outputs 0. Else, it computes, for all $i \leq n$, $s_i = \text{Transform-Image}(\text{Verify}_i, \text{VK}_i, m_i)$. If $(\prod_i s_i) = \sigma_{\text{agg}}^e \pmod{N}$, it outputs 1, else it outputs 0.

²⁶Padded appropriately to be of the same size as Transform-1, Transform-2, Transform-3 and Transform-4.

²⁷Padded appropriately to be of the same size as Transform-Image-1 and Transform-Image-2.

B.1 Proof of Security

We will now prove that the scheme described in Section B is an adaptively secure universal signature aggregator with respect to all secure length-qualified signature schemes.

Theorem B.1. Assuming $\mathcal{O}$ is a secure virtual black-box obfuscator, F is a secure puncturable PRF, $\tilde{F}$ is a secure PRF, PRG is a secure pseudorandom generator and RSA is secure, for all $(\ell_{\text{ver}}, \ell_{\text{vk}}, \ell_{\text{msg}}, \ell_{\text{sig}})$ -length qualified secure signature schemes $\mathcal{S}$, the universal signature aggregator $(\ell_{\text{ver}}, \ell_{\text{vk}}, \ell_{\text{msg}}, \ell_{\text{sig}})$ -UniversalSigAgg is adaptively secure with respect to $\mathcal{S}$.

To prove the above theorem, we will first describe a sequence of hybrid experiments.

B.1.1 Sequence of Games

Game 0 This corresponds to the adaptive security game $\text{Exp}_{\text{Att}, \mathcal{S}}(\lambda)$ in which the challenger interacts with adversary Att.

1. Compute $(\text{SK}, \text{VK}) \leftarrow \mathcal{S}.\text{Gen}(1^\lambda)$. Choose an RSA modulus N , $e \in \mathbb{Z}_{\phi(N)}^*$, $v \in \mathbb{Z}_N^*$ and $c_{i,b} \in \mathbb{Z}_{\phi(N)}$ for all $i \leq d_2, b \in \{0, 1\}$. Choose $(\text{SK}, \text{VK}) \leftarrow \mathcal{S}.\text{Gen}(1^\lambda)$ and set $\text{PP} = (\mathcal{O}(\text{Transform}_{N,v,c}), N, e)$. Send (PP, VK) to Att.
2. For each signature query x_i , choose $r_i \leftarrow \{0, 1\}^{\ell_{\text{rnd}}}$ and compute $\sigma_i \leftarrow \mathcal{S}.\text{Sign}(\text{SK}, x_i)$. Send σ_i to Att.
3. Att sends a forgery σ_{agg} along with n tuples $\{\text{Verify}_i, \text{VK}_i, m_i\}_{i=1}^n$. Att wins if
 - (a) $\exists i^* \in [n]$ such that $\text{Verify}_{i^*} = \mathcal{S}.\text{Verify}$, $\text{VK}_{i^*} = \text{VK}$ and m_{i^*} was not queried during the signature phase
 - (b) $\text{UniversalVerify}(\text{PP}, \{\text{Verify}_i, \text{VK}_i, m_i\}, \sigma_{\text{agg}}) = 1$

Game 1 In this game, the challenger computes the signatures using the PRF $\tilde{F}$.

1. Compute $(\text{SK}, \text{VK}) \leftarrow \mathcal{S}.\text{Gen}(1^\lambda)$. Choose an RSA modulus N , $e \in \mathbb{Z}_{\phi(N)}^*$, $v \in \mathbb{Z}_N^*$ and $c_{i,b} \in \mathbb{Z}_{\phi(N)}$ for all $i \leq d_2, b \in \{0, 1\}$.
Choose $\tilde{K} \leftarrow F.\text{setup}(1^\lambda)$.
Set $\text{PP} = (\mathcal{O}(\text{Transform}_{N,v,c}), \mathcal{O}(\text{Transform-Image}_{N,v,c,e}), N, e)$. Send (PP, VK) to Att.
2. For each signature query x_i , choose $\rho_i \leftarrow \{0, 1\}^{\ell_{\text{sig}}}$, compute $r_i \leftarrow \tilde{F}(\tilde{K}, \rho_i)$ and $\sigma_i = \mathcal{S}.\text{Sign}(\text{SK}, x_i; r_i)$. Send σ_i to Att.
3. Att sends a forgery σ_{agg} along with n tuples $\{\text{Verify}_i, \text{VK}_i, m_i\}_{i=1}^n$. Att wins if
 - (a) $\exists i^* \in [n]$ such that $\text{Verify}_{i^*} = \mathcal{S}.\text{Verify}$, $\text{VK}_{i^*} = \text{VK}$ and m_{i^*} was not queried during the signature phase
 - (b) $\text{UniversalVerify}(\text{PP}, \{\text{Verify}_i, \text{VK}_i, m_i\}, \sigma_{\text{agg}}) = 1$

Game 2 In this game, the challenger uses program Transform-1 to compute the public parameters PP. This program is similar to the program Transform. However, it has the additional functionality that it allows user to receive signatures using secret key SK, provided the user has a ‘trapdoor’ for Transform-1. In this game, since $\alpha \leftarrow \{0, 1\}^{2\ell}$, it is unlikely that there exists a ‘trapdoor’.

1. Compute $(\text{SK}, \text{VK}) \leftarrow \mathcal{S}.\text{Gen}(1^\lambda)$. Choose an RSA modulus N , $e \in \mathbb{Z}_{\phi(N)}^*$, $v \in \mathbb{Z}_N^*$ and $c_{i,b} \in \mathbb{Z}_{\phi(N)}$ for all $i \leq d_2, b \in \{0, 1\}$.
Choose $\tilde{K} \leftarrow F.\text{setup}(1^\lambda)$ and $\alpha \leftarrow \{0, 1\}^{2\ell}$.
Let Transform-1²⁸ be the circuit defined below.
Set $\text{PP} = (\mathcal{O}(\text{Transform-1}_{N,v,c,\alpha,\text{SK},\tilde{K}}), \mathcal{O}(\text{Transform-Image}_{N,v,c,e}), N, e)$. Send (PP, VK) to Att.

²⁸Padded appropriately to be of the same size as Transform, Transform-2, Transform-3 and Transform-4.

2. For each signature query x_i , choose $\rho_i \leftarrow \{0, 1\}^{\ell_{\text{sig}}}$, compute $r_i \leftarrow \tilde{F}(\tilde{K}, \rho_i)$ and $\sigma_i = \mathcal{S}.\text{Sign}(\text{SK}, x_i; r_i)$. Send σ_i to Att.
3. Att sends a forgery σ_{agg} along with n tuples $\{\text{Verify}_i, \text{VK}_i, m_i\}_{i=1}^n$. Att wins if
 - (a) $\exists i^* \in [n]$ such that $\text{Verify}_{i^*} = \mathcal{S}.\text{Verify}$, $\text{VK}_{i^*} = \text{VK}$ and m_{i^*} was not queried during the signature phase
 - (b) $\text{UniversalVerify}(\text{PP}, \{\text{Verify}_i, \text{VK}_i, m_i\}, \sigma_{\text{agg}}) = 1$

Transform-1 _{$N, v, c, \alpha, \text{SK}, \tilde{K}$} :

Inputs: $b \in \{0, 1\}$, $a' \in \{0, 1\}^\ell$, $\text{Verify}' \in \{0, 1\}^{\ell_{\text{ver}}}$, $\text{VK}' \in \{0, 1\}^{\ell_{\text{vk}}}$, $m' \in \{0, 1\}^{\ell_{\text{msg}}}$, $\sigma' \in \{0, 1\}^{\ell_{\text{sig}}}$.

Constants : RSA modulus $N \in \mathbb{N}$, $v \in \mathbb{Z}_N^*$, $\mathbf{c} = \{c_{i,b}\} \in \mathbb{Z}_{\phi(N)}^{2d_2}$, $\alpha \in \{0, 1\}^{2\ell}$, $\text{SK} \in \mathcal{SK}$, $\tilde{K} \in \tilde{\mathcal{K}}$.

```

if  $b = 0$  then
  if  $\text{PRG}(a') \neq \alpha$  then
    Output  $\perp$ .
  else
    Output  $(\text{VK}, \mathcal{S}.\text{Sign}(\text{SK}, m'; \tilde{F}(\tilde{K}, \sigma')))$ .
  end if
else if  $\text{Verify}'(\text{VK}', m', \sigma') = 0$  then
  Output  $\perp$ .
else
  Compute  $h(m') = x$  and let  $z = x || \text{Verify}' || \text{VK}'$ .
  Output  $v \Pi^{c_{i,z_i}} \pmod{N}$ 
end if

```

Game 3 This game is exactly similar to the previous experiment, except that the challenger chooses α such that there exists a trapdoor for program Transform-1. For this, the challenger chooses $a \leftarrow \{0, 1\}^\ell$ and sets $\alpha = \text{PRG}(a)$.

1. Compute $(\text{SK}, \text{VK}) \leftarrow \mathcal{S}.\text{Gen}(1^\lambda)$. Choose an RSA modulus N , $e \in \mathbb{Z}_{\phi(N)}^*$, $v \in \mathbb{Z}_N^*$ and $c_{i,b} \in \mathbb{Z}_{\phi(N)}$ for all $i \leq d_2, b \in \{0, 1\}$. Choose $\tilde{K} \leftarrow \tilde{F}.\text{setup}(1^\lambda)$, $a \leftarrow \{0, 1\}^\ell$ and set $\alpha = \text{PRG}(a)$. Set $\text{PP} = (\mathcal{O}(\text{Transform-1}_{N, v, \mathbf{c}, \alpha, \text{SK}, \tilde{K}}), \mathcal{O}(\text{Transform-Image}_{N, v, \mathbf{c}, e}), N, e)$. Send (PP, VK) to Att.
2. For each signature query x_i , choose $\rho_i \leftarrow \{0, 1\}^{\ell_{\text{sig}}}$, compute $r_i \leftarrow \tilde{F}(\tilde{K}, \rho_i)$ and $\sigma_i = \mathcal{S}.\text{Sign}(\text{SK}, x_i; r_i)$. Send σ_i to Att.
3. Att sends a forgery σ_{agg} along with n tuples $\{\text{Verify}_i, \text{VK}_i, m_i\}_{i=1}^n$. Att wins if
 - (a) $\exists i^* \in [n]$ such that $\text{Verify}_{i^*} = \mathcal{S}.\text{Verify}$, $\text{VK}_{i^*} = \text{VK}$ and m_{i^*} was not queried during the signature phase
 - (b) $\text{UniversalVerify}(\text{PP}, \{\text{Verify}_i, \text{VK}_i, m_i\}, \sigma_{\text{agg}}) = 1$

Game 4 In this experiment, the challenger defines a random challenge subspace of the message space. The adversary wins if all signature queries lie outside the challenge space and the message m_{i^*} corresponding to $\mathcal{S}.\text{Verify}$, VK lies in the challenge space.

1. Compute $(\text{SK}, \text{VK}) \leftarrow \mathcal{S}.\text{Gen}(1^\lambda)$. Choose an RSA modulus N , $e \in \mathbb{Z}_{\phi(N)}^*$, $v \in \mathbb{Z}_N^*$ and $c_{i,b} \in \mathbb{Z}_{\phi(N)}$ for all $i \leq d_2, b \in \{0, 1\}$. Choose $\tilde{K} \leftarrow \tilde{F}.\text{setup}(1^\lambda)$, $a \leftarrow \{0, 1\}^\ell$ and set $\alpha = \text{PRG}(a)$. Choose $u \leftarrow \text{AdmSample}(1^\lambda, q_1 + n)$. Set $\text{PP} = (\mathcal{O}(\text{Transform-1}_{N, v, \mathbf{c}, \alpha, \text{SK}, \tilde{K}}), \mathcal{O}(\text{Transform-Image}_{N, v, \mathbf{c}, e}), N, e)$. Send (PP, VK) to Att.

2. For each signature query x_i ,
 - (a) If $P_u(x_i) = 0$, flip a coin $\gamma_i \in \{0, 1\}$ and abort. Att wins if $\gamma_i = 1$.
 - (b) Else, choose $\rho_i \leftarrow \{0, 1\}^{\ell_{\text{sig}}}$, compute $r_i \leftarrow \tilde{F}(\tilde{K}, \rho_i)$ and $\sigma_i = \mathcal{S}.\text{Sign}(\text{SK}, x_i; r_i)$. Send σ_i to Att.
3. Att sends a forgery σ_{agg} along with n tuples $\{\text{Verify}_i, \text{VK}_i, m_i\}_{i=1}^n$. Att wins if
 - (a) $\exists i^* \in [n]$ such that $\text{Verify}_{i^*} = \mathcal{S}.\text{Verify}$, $\text{VK}_{i^*} = \text{VK}$ and m_{i^*} was not queried during the signature phase. Let i^* be the smallest such index. Then,
 - (b) $(\forall i \neq i^* \text{ such that } \text{Verify}_i = \mathcal{S}.\text{Verify}, \text{VK}_i = \text{VK}, P_u(m_i) = 1) \text{ and } P_u(m_{i^*}) = 0$
 - (c) $\text{UniversalVerify}(\text{PP}, \{\text{Verify}_i, \text{VK}_i, m_i\}, \sigma_{\text{agg}}) = 1$

Game 5 In this experiment, the challenger uses program Transform-2 instead of Transform-1. The only difference between Transform-1 and Transform-2 is that Transform-2 rejects inputs of the form $(1, a, \mathcal{S}.\text{Verify}, \text{VK}, m', \sigma')$ such that $\mathcal{S}.\text{Verify}(\text{VK}, m', \sigma') = 1$ and m' lies in the challenge space.

1. Compute $(\text{SK}, \text{VK}) \leftarrow \mathcal{S}.\text{Gen}(1^\lambda)$. Choose an RSA modulus N , $e \in \mathbb{Z}_{\phi(N)}^*$, $v \in \mathbb{Z}_N^*$ and $c_{i,b} \in \mathbb{Z}_{\phi(N)}$ for all $i \leq d_2, b \in \{0, 1\}$.
 Choose $\tilde{K} \leftarrow \tilde{F}.\text{setup}(1^\lambda)$, $a \leftarrow \{0, 1\}^\ell$ and set $\alpha = \text{PRG}(a)$. Choose $u \leftarrow \text{AdmSample}(1^\lambda, q_1 + n)$.
 Let Transform-2²⁹ be the circuit defined below.
 Set $\text{PP} = (\mathcal{O}(\text{Transform-2}_{u, N, v, c, \alpha, \text{SK}, \tilde{K}}), \mathcal{O}(\text{Transform-Image}_{N, v, c, e}), N, e)$. Send (PP, VK) to Att.
2. For each signature query x_i ,
 - (a) If $P_u(x_i) = 0$, flip a coin $\gamma_i \in \{0, 1\}$ and abort. Att wins if $\gamma_i = 1$.
 - (b) Else, choose $\rho_i \leftarrow \{0, 1\}^{\ell_{\text{sig}}}$, compute $r_i \leftarrow \tilde{F}(\tilde{K}, \rho_i)$ and $\sigma_i = \mathcal{S}.\text{Sign}(\text{SK}, x_i; r_i)$. Send σ_i to Att.
3. Att sends a forgery σ_{agg} along with n tuples $\{\text{Verify}_i, \text{VK}_i, m_i\}_{i=1}^n$. Att wins if
 - (a) $\exists i^* \in [n]$ such that $\text{Verify}_{i^*} = \mathcal{S}.\text{Verify}$, $\text{VK}_{i^*} = \text{VK}$ and m_{i^*} was not queried during the signature phase. Let i^* be the smallest such index. Then,
 - (b) $(\forall i \neq i^* \text{ such that } \text{Verify}_i = \mathcal{S}.\text{Verify}, \text{VK}_i = \text{VK}, P_u(m_i) = 1) \text{ and } P_u(m_{i^*}) = 0$
 - (c) $\text{UniversalVerify}(\text{PP}, \{\text{Verify}_i, \text{VK}_i, m_i\}, \sigma_{\text{agg}}) = 1$

²⁹Padded appropriately to be of the same size as Transform, Transform-1, Transform-3 and Transform-4.

Transform-2 _{$u, N, v, \mathbf{c}, \alpha, \text{SK}, \tilde{K}$} :

Inputs: $b \in \{0, 1\}$, $a' \in \{0, 1\}^\ell$, $\text{Verify}' \in \{0, 1\}^{\ell_{\text{ver}}}$, $\text{VK}' \in \{0, 1\}^{\ell_{\text{vk}}}$, $m' \in \{0, 1\}^{\ell_{\text{msg}}}$, $\sigma' \in \{0, 1\}^{\ell_{\text{sig}}}$.

Constants : $u \in \{0, 1, \perp\}^{d_2}$, RSA modulus $N \in \mathbb{N}$, $v \in \mathbb{Z}_N^*$, $\mathbf{c} = \{c_{i,b}\} \in \mathbb{Z}_{\phi(N)}^{2d_2}$, $\alpha \in \{0, 1\}^{2\ell}$, $\text{SK} \in \mathcal{SK}$, $\tilde{K} \in \tilde{\mathcal{K}}$.

```

if  $b = 0$  then
  if  $\text{PRG}(a') \neq \alpha$  then
    Output  $\perp$ .
  else
    Output  $(\text{VK}, \mathcal{S}.\text{Sign}(\text{SK}, m'; \tilde{F}(\tilde{K}, \sigma')))$ .
  end if
else if  $\text{Verify}'(\text{VK}', m', \sigma') = 0$  then
  Output  $\perp$ .
else
  Compute  $h(m') = x$  and let  $z = \text{Verify}' || \text{VK}' || x$ .
  if  $\text{Verify}' = \mathcal{S}.\text{Verify}$ ,  $\text{VK}' = \text{VK}$  and  $P_u(m') = 0$  then
    Output  $\perp$ .
  end if
  Output  $v \prod c_{i, z_i} \pmod{N}$ .
end if

```

Game 6 This game is similar to the previous one, except for the manner in which the constants $c_{1,b}$ are chosen.

1. Compute $(\text{SK}, \text{VK}) \leftarrow \mathcal{S}.\text{Gen}(1^\lambda)$. Choose an RSA modulus N , $e \leftarrow \mathbb{Z}_{\phi(N)}^*$, $v \in \mathbb{Z}_N^*$.
Choose $c'_{i,b} \in \mathbb{Z}_{\phi(N)}$ for all $i \leq d_2, b \in \{0, 1\}$. Set $c_{1,b} = c'_{1,b} \cdot e^{-1}$ and $c_{i,b} = c'_{i,b}$ for all $i > 1$.
Choose $\tilde{K} \leftarrow \tilde{F}.\text{setup}(1^\lambda)$, $a \leftarrow \{0, 1\}^\ell$ and set $\alpha = \text{PRG}(a)$. Choose $u \leftarrow \text{AdmSample}(1^\lambda, q_1 + n)$.
Set $\text{PP} = (\mathcal{O}(\text{Transform-2}_{u, N, v, \mathbf{c}, \alpha, \text{SK}, \tilde{K}}), \mathcal{O}(\text{Transform-Image}_{N, v, \mathbf{c}, e}), N, e)$. Send (PP, VK) to Att.
2. For each signature query x_i ,
 - (a) If $P_u(x_i) = 0$, flip a coin $\gamma_i \in \{0, 1\}$ and abort. Att wins if $\gamma_i = 1$.
 - (b) Else, choose $\rho_i \leftarrow \{0, 1\}^{\ell_{\text{sig}}}$, compute $r_i \leftarrow \tilde{F}(\tilde{K}, \rho_i)$ and $\sigma_i = \mathcal{S}.\text{Sign}(\text{SK}, x_i; r_i)$. Send σ_i to Att.
3. Att sends a forgery σ_{agg} along with n tuples $\{\text{Verify}_i, \text{VK}_i, m_i\}_{i=1}^n$. Att wins if
 - (a) $\exists i^* \in [n]$ such that $\text{Verify}_{i^*} = \mathcal{S}.\text{Verify}$, $\text{VK}_{i^*} = \text{VK}$ and m_{i^*} was not queried during the signature phase. Let i^* be the smallest such index. Then,
 - (b) $(\forall i \neq i^* \text{ such that } \text{Verify}_i = \mathcal{S}.\text{Verify}, \text{VK}_i = \text{VK}, P_u(m_i) = 1) \text{ and } P_u(m_{i^*}) = 0$
 - (c) $\text{UniversalVerify}(\text{PP}, \{\text{Verify}_i, \text{VK}_i, m_i\}, \sigma_{\text{agg}}) = 1$

Game 7 In this game, the challenger uses programs Transform-3 and Transform-Image-1.

1. Compute $(\text{SK}, \text{VK}) \leftarrow \mathcal{S}.\text{Gen}(1^\lambda)$. Choose an RSA modulus N , $e \leftarrow \mathbb{Z}_{\phi(N)}^*$, $v \in \mathbb{Z}_N^*$.
Choose $c_{i,b} \in \mathbb{Z}_{\phi(N)}$ for all $i \leq d_2, b \in \{0, 1\}$.
Choose $\tilde{K} \leftarrow \tilde{F}.\text{setup}(1^\lambda)$, $a \leftarrow \{0, 1\}^\ell$ and set $\alpha = \text{PRG}(a)$. Choose $u \leftarrow \text{AdmSample}(1^\lambda, q_1 + n)$.
Let Transform-3³⁰ and Transform-Image-1³¹ be the circuits defined below.

³⁰Padded appropriately to be of the same size as Transform, Transform-1, Transform-2 and Transform-4.

³¹Padded appropriately to be of the same size as Transform-Image and Transform-Image-2.

Set $PP = (\mathcal{O}(\text{Transform-3}_{u,N,v,\mathbf{c},\alpha,\text{SK},\tilde{K},e^{-1}}), \mathcal{O}(\text{Transform-Image-1}_{N,v,\mathbf{c}}), N, e)$ where the circuits Transform-3 and Transform-Image-1 are defined below. Send (PP, VK) to Att.

2. For each signature query x_i ,
 - (a) If $P_u(x_i) = 0$, flip a coin $\gamma_i \in \{0, 1\}$ and abort. Att wins if $\gamma_i = 1$.
 - (b) Else, choose $\rho_i \leftarrow \{0, 1\}^{\ell_{\text{sig}}}$, compute $r_i \leftarrow \tilde{F}(\tilde{K}, \rho_i)$ and $\sigma_i = \mathcal{S}.\text{Sign}(\text{SK}, x_i; r_i)$. Send σ_i to Att.
3. Att sends a forgery σ_{agg} along with n tuples $\{\text{Verify}_i, \text{VK}_i, m_i\}_{i=1}^n$. Att wins if
 - (a) $\exists i^* \in [n]$ such that $\text{Verify}_{i^*} = \mathcal{S}.\text{Verify}$, $\text{VK}_{i^*} = \text{VK}$ and m_{i^*} was not queried during the signature phase. Let i^* be the smallest such index. Then,
 - (b) $(\forall i \neq i^* \text{ such that } \text{Verify}_i = \mathcal{S}.\text{Verify}, \text{VK}_i = \text{VK}, P_u(m_i) = 1) \text{ and } P_u(m_{i^*}) = 0$
 - (c) $\text{UniversalVerify}(PP, \{\text{Verify}_i, \text{VK}_i, m_i\}, \sigma_{\text{agg}}) = 1$

Transform-3 _{$u,N,v,\mathbf{a},\alpha,\text{SK},\tilde{K},e^{-1}$} :

Inputs: $b \in \{0, 1\}$, $a \in \{0, 1\}^\ell$, $\text{Verify}' \in \{0, 1\}^{\ell_{\text{ver}}}$, $\text{VK}' \in \{0, 1\}^{\ell_{\text{vk}}}$, $m' \in \{0, 1\}^{\ell_{\text{msg}}}$, $\sigma' \in \{0, 1\}^{\ell_{\text{sig}}}$.

Constants : $u \in \{0, 1, \perp\}^{d_2}$, RSA modulus $N \in \mathbb{N}$, $v \in \mathbb{Z}_N^*$, $\mathbf{c} = \{a_{i,b}\} \in \mathbb{Z}_N^{2d_2}$, $\alpha \in \{0, 1\}^{2\ell}$, $\text{SK} \in \mathcal{SK}$, $\tilde{K} \in \tilde{\mathcal{K}}$, $e^{-1} \in \mathbb{Z}_{\phi(N)}^*$.

```

if  $b = 0$  then
  if  $\text{PRG}(a) \neq \alpha$  then
    Output  $\perp$ .
  else
    Output  $(\text{VK}, \mathcal{S}.\text{Sign}(\text{SK}, m'; \tilde{F}(\tilde{K}, \sigma')))$ .
  end if
else if  $\text{Verify}'(\text{VK}', m', \sigma') = 0$  then
  Output  $\perp$ .
else
  Compute  $h(m') = x$  and let  $z = x || \text{Verify}' || \text{VK}'$ .
  if  $\text{Verify}' = \mathcal{S}.\text{Verify}$  and  $\text{VK}' = \text{VK}$  and  $P_u(m') = 0$  then
    Output  $\perp$ .
  end if
  Output  $v(\prod_i c_{i,z_i}) \cdot e^{-1} \pmod{N}$ .
end if

```

Transform-Image-1 _{$N,v,\mathbf{c}$} :

Inputs: $\text{Verify}' \in \{0, 1\}^{\ell_{\text{ver}}}$, $\text{VK}' \in \{0, 1\}^{\ell_{\text{vk}}}$, $m' \in \{0, 1\}^{\ell_{\text{msg}}}$.

Constants : RSA modulus $N \in \mathbb{N}$, $v \in \mathbb{Z}_N^*$, $\mathbf{c} = \{c_{i,b}\} \in \mathbb{Z}_N^{2d_2}$.

```

Compute  $h(m') = x$  and let  $z = x || \text{Verify}' || \text{VK}'$ .
Output  $v \prod_i c_{i,z_i} \pmod{N}$ .

```

Game 8 In this game, the challenger modifies the manner in which constants $c_{i,b}$ are chosen. Instead of choosing them uniformly at random from $\mathbb{Z}_{\phi(N)}$, the challenger now chooses $a_{i,b} \leftarrow \mathbb{Z}_{\phi(N)}$ and sets $c_{i,b}$ appropriately, depending on u and $y = \mathcal{S}.\text{Verify} || \text{VK}$.

1. Compute $(\text{SK}, \text{VK}) \leftarrow \mathcal{S}.\text{Gen}(1^\lambda)$. Choose an RSA modulus N , $e \leftarrow \mathbb{Z}_{\phi(N)}^*$, $v \in \mathbb{Z}_N^*$. Choose $\tilde{K} \leftarrow \tilde{F}.\text{setup}(1^\lambda)$, $a \leftarrow \{0, 1\}^\ell$ and set $\alpha = \text{PRG}(a)$. Choose $u \leftarrow \text{AdmSample}(1^\lambda, q_1 + n)$. Choose $a_{i,b} \in \mathbb{Z}_N$ for all $i \leq d_2, b \in \{0, 1\}$. Let $y = \mathcal{S}.\text{Verify} || \text{VK}$.

For $i \leq d_1$, set $c_{i,b} = e \cdot a_{i,b} \bmod \phi(N)$ if $u_i = b$, else $c_{i,b} = e \cdot a_{i,b} + 1 \bmod \phi(N)$.

For $i > d_1$, set $c_{i,b} = e \cdot a_{i,b} \bmod \phi(N)$ if $y_{i-d_1} \neq b$, else $c_{i,b} = e \cdot a_{i,b} + 1 \bmod \phi(N)$.

Set $PP = (\mathcal{O}(\text{Transform-3}_{u,N,v,c,\alpha,\text{SK},\tilde{K},e^{-1}}), \mathcal{O}(\text{Transform-Image-1}_{N,v,c}), N, e)$. Send (PP, VK) to Att.

2. For each signature query x_i ,

- (a) If $P_u(x_i) = 0$, flip a coin $\gamma_i \in \{0, 1\}$ and abort. Att wins if $\gamma_i = 1$.
- (b) Else, choose $\rho_i \leftarrow \{0, 1\}^{\ell_{\text{sig}}}$, compute $r_i \leftarrow \tilde{F}(\tilde{K}, \rho_i)$ and $\sigma_i = \mathcal{S}.\text{Sign}(\text{SK}, x_i; r_i)$. Send σ_i to Att.

3. Att sends a forgery σ_{agg} along with n tuples $\{\text{Verify}_i, \text{VK}_i, m_i\}_{i=1}^n$. Att wins if

- (a) $\exists i^* \in [n]$ such that $\text{Verify}_{i^*} = \mathcal{S}.\text{Verify}$, $\text{VK}_{i^*} = \text{VK}$ and m_{i^*} was not queried during the signature phase. Let i^* be the smallest such index. Then,
- (b) $(\forall i \neq i^* \text{ such that } \text{Verify}_i = \mathcal{S}.\text{Verify}, \text{VK}_i = \text{VK}, P_u(m_i) = 1) \text{ and } P_u(m_{i^*}) = 0$
- (c) $\text{UniversalVerify}(PP, \{\text{Verify}_i, \text{VK}_i, m_i\}, \sigma_{\text{agg}}) = 1$

Game 9 This game is similar to the previous one, except that the constants $c_{i,b}$ are computed within the program Transform-4 (which is used instead of Transform-3). Similarly, program Transform-Image-2 is used instead of Transform-Image-1.

1. Compute $(\text{SK}, \text{VK}) \leftarrow \mathcal{S}.\text{Gen}(1^\lambda)$. Choose an RSA modulus N , $e \leftarrow \mathbb{Z}_{\phi(N)}^*$, $v \in \mathbb{Z}_N^*$.

Choose $\tilde{K} \leftarrow \tilde{F}.\text{setup}(1^\lambda)$, $a \leftarrow \{0, 1\}^\ell$ and set $\alpha = \text{PRG}(a)$. Choose $u \leftarrow \text{AdmSample}(1^\lambda, q_1 + n)$.

Choose $a_{i,b} \in \mathbb{Z}_N$ for all $i \leq d_2, b \in \{0, 1\}$. Let $\mathbf{a} = \{a_{i,b}\}$ and $y = \mathcal{S}.\text{Verify}||\text{VK}$.

Let Transform-4³² and Transform-Image-2³³ be the circuits defined below.

Set $PP = (\mathcal{O}(\text{Transform-4}_{u,N,v,\mathbf{a},\alpha,\text{SK},\tilde{K},e,y}), \mathcal{O}(\text{Transform-Image-2}_{N,v,\mathbf{a},e,y}), N, e)$ where Transform-4 and Transform-Image-2 are defined below. Send (PP, VK) to Att.

2. For each signature query x_i ,

- (a) If $P_u(x_i) = 0$, flip a coin $\gamma_i \in \{0, 1\}$ and abort. Att wins if $\gamma_i = 1$.
- (b) Else, choose $\rho_i \leftarrow \{0, 1\}^{\ell_{\text{sig}}}$, compute $r_i \leftarrow \tilde{F}(\tilde{K}, \rho_i)$ and $\sigma_i = \mathcal{S}.\text{Sign}(\text{SK}, x_i; r_i)$. Send σ_i to Att.

3. Att sends a forgery σ_{agg} along with n tuples $\{\text{Verify}_i, \text{VK}_i, m_i\}_{i=1}^n$. Att wins if

- (a) $\exists i^* \in [n]$ such that $\text{Verify}_{i^*} = \mathcal{S}.\text{Verify}$, $\text{VK}_{i^*} = \text{VK}$ and m_{i^*} was not queried during the signature phase. Let i^* be the smallest such index. Then,
- (b) $(\forall i \neq i^* \text{ such that } \text{Verify}_i = \mathcal{S}.\text{Verify}, \text{VK}_i = \text{VK}, P_u(m_i) = 1) \text{ and } P_u(m_{i^*}) = 0$
- (c) $\text{UniversalVerify}(PP, \{\text{Verify}_i, \text{VK}_i, m_i\}, \sigma_{\text{agg}}) = 1$

³²Padded appropriately to be of the same size as Transform, Transform-1, Transform-2 and Transform-3.

³³Padded appropriately to be of the same size as Transform-Image and Transform-Image-1.

Transform-4 _{$u, N, v, \mathbf{a}, \alpha, \text{SK}, \tilde{K}, e$} :

Inputs: $b \in \{0, 1\}$, $a \in \{0, 1\}^\ell$, $\text{Verify}' \in \{0, 1\}^{\ell_{\text{ver}}}$, $\text{VK}' \in \{0, 1\}^{\ell_{\text{vk}}}$, $m' \in \{0, 1\}^{\ell_{\text{msg}}}$, $\sigma' \in \{0, 1\}^{\ell_{\text{sig}}}$.

Constants : $u \in \{0, 1, \perp\}^{d_2}$, RSA modulus $N \in \mathbb{N}$, $v \in \mathbb{Z}_N^*$, $\mathbf{a} = \{a_{i,b}\} \in \mathbb{Z}_N^{2d_2}$, $\alpha \in \{0, 1\}^{2\ell}$, $\text{SK} \in \mathcal{SK}$, $\tilde{K} \in \tilde{\mathcal{K}}$, $e \in \mathbb{Z}_{\phi(N)}^*$, $y \in \{0, 1\}^{\ell_{\text{ver}}} \times \{0, 1\}^{\ell_{\text{vk}}}$.

```

Let  $y' = \text{Verify}' || \text{VK}'$ .
if  $b = 0$  then
  if  $\text{PRG}(a) \neq \alpha$  then
    Output  $\perp$ .
  else
    Output  $(\text{VK}, \mathcal{S}.\text{Sign}(\text{SK}, m'; \tilde{F}(\tilde{K}, \sigma')))$ .
  end if
else if  $\text{Verify}'(\text{VK}', m', \sigma') = 0$  then
  Output  $\perp$ .
else
  Compute  $h(m') = x$  and let  $z = x || \text{Verify}' || \text{VK}'$ .
  if  $\text{Verify}' = \mathcal{S}.\text{Verify}$  and  $\text{VK}' = \text{VK}$  and  $P_u(m') = 0$  then
    Output  $\perp$ .
  end if
  if  $y = y'$  then
    Let  $i'$  be the first index such that  $u_{i'} = x_{i'}$ . Set  $c_{i', x_{i'}} = a_{i', x_{i'}}$ .
     $\forall i \leq d_1, i \neq i'$ , set  $c_{i,b} = e \cdot a_{i,b}$  if  $u_i = b$ , else  $c_{i,b} = e \cdot a_{i,b} + 1$ .
     $\forall i > d_1$ , set  $c_{i,b} = e \cdot a_{i,b}$  if  $y_{i-d_1} \neq b$ , else  $c_{i,b} = e \cdot a_{i,b} + 1$ .
  else
    Let  $i'$  be the first index such that  $y_{i'} \neq y'_{i'}$ . Set  $c_{d_1+i', y'_{i'}} = a_{d_1+i', y'_{i'}}$ .
     $\forall i \leq d_1$ , set  $c_{i,b} = e \cdot a_{i,b}$  if  $u_i = b$ , else  $c_{i,b} = e \cdot a_{i,b} + 1$ .
     $\forall i > d_1, i - d_1 \neq i'$ , set  $c_{i,b} = e \cdot a_{i,b}$  if  $y_{i-d_1} \neq b$ , else  $c_{i,b} = e \cdot a_{i,b} + 1$ .
  end if
  Output  $v^{\prod c_{i,z_i}} \pmod{N}$ .
end if

```

Transform-Image-2 _{$N, v, \mathbf{a}, e$} :

Inputs: $\text{Verify}' \in \{0, 1\}^{\ell_{\text{ver}}}$, $\text{VK}' \in \{0, 1\}^{\ell_{\text{vk}}}$, $m' \in \{0, 1\}^{\ell_{\text{msg}}}$.

Constants : RSA modulus $N \in \mathbb{N}$, $v \in \mathbb{Z}_N^*$, $\mathbf{a} = \{a_{i,b}\} \in \mathbb{Z}_N^{2d_2}$, $e \in \mathbb{Z}_{\phi(N)}^*$, $y \in \{0, 1\}^{\ell_{\text{ver}}} \times \{0, 1\}^{\ell_{\text{vk}}}$.

```

Compute  $h(m') = x$  and set  $z = x || \text{Verify}' || \text{VK}'$ .
 $\forall i \leq d_1$ , set  $c_{i,b} = e \cdot a_{i,b}$  if  $u_i = b$ , else  $c_{i,b} = e \cdot a_{i,b} + 1$ .
 $\forall i > d_1$ , set  $c_{i,b} = e \cdot a_{i,b}$  if  $y_{i-d_1} \neq b$ , else  $c_{i,b} = e \cdot a_{i,b} + 1$ .
Output  $v^{\prod c_{i,x_i}} \pmod{N}$ .

```

B.1.2 Adversary's Advantage in the Games

Let $\text{Adv}_{\text{Att}}^j$ denote the advantage of adversary Att in Game j .

Claim B.1. Assuming $\tilde{F}$ is a secure PRF, for any PPT adversary Att,

$$\text{Adv}_{\text{Att}}^0 - \text{Adv}_{\text{Att}}^1 \leq \text{negl}(\lambda).$$

Proof. Similar to proof of Claim 5.1. ■

Claim B.2. Assuming $\mathcal{O}$ is a secure indistinguishability obfuscator, for any PPT adversary Att,

$$\text{Adv}_{\text{Att}}^1 - \text{Adv}_{\text{Att}}^2 \leq \text{negl}(\lambda).$$

Proof. Similar to the proof of Claim 5.2. ■

Claim B.3. Assuming PRG is a secure pseudorandom generator, for any PPT adversary Att,

$$\text{Adv}_{\text{Att}}^2 - \text{Adv}_{\text{Att}}^3 \leq \text{negl}(\lambda).$$

Proof. Similar to proof of Claim 5.3. ■

Claim B.4. For any adversary Att,

$$\text{Adv}_{\text{Att}}^4 \geq \text{Adv}_{\text{Att}}^3 / \theta(q_1 + n).$$

Proof. This follows from the θ -admissibility of h . Let $\mathcal{I} = \{i' : \text{Verify}_{i'} = \mathcal{S}.\text{Verify}, \text{VK}_{i'} = \text{VK}\}$. Let $y_i = h(x_i)$ for all $i \leq q_1$, and $y_{q_1+j} = h(m_j)$ for all $j \in \mathcal{I}$. Note that $m_{i^*} \neq x_i$ for all $i \leq q_1$, and $m_{i^*} \neq m_j$ for all $j \in \mathcal{I}, j \neq i^*$. Therefore,

$$\Pr[\forall i \leq q_1, P_u(x_i) = 1 \text{ and } \forall j \in \mathcal{I}, j \neq i^* P_u(m_j) = 1 \text{ and } P_u(m_{i^*}) = 0] \geq 1/\theta(q_1 + n)$$

where the probability is only over the choice of $u \leftarrow \text{AdmSample}(1^\lambda, q_1 + n)$. ■

Claim B.5. Assuming $\mathcal{O}$ is a secure virtual black box obfuscator, $\tilde{F}$ is a secure pseudorandom function, PRG is a secure pseudorandom generator and $\mathcal{S}$ is a $(\ell_{\text{ver}}, \ell_{\text{vk}}, \ell_{\text{msg}}, \ell_{\text{sig}})$ -length qualified secure signature scheme,

$$\text{Adv}_{\text{Att}}^4 - \text{Adv}_{\text{Att}}^5 \leq \text{negl}(\lambda).$$

Proof. The proof of this claim is along the lines of the proof of Claim 5.1. Given only oracle access to either $\text{Transform-1}_{N,v,c,\alpha,\text{SK},\tilde{K}}$ or $\text{Transform-2}_{u,N,v,c,\alpha,\text{SK},\tilde{K}}$, the only way in which an adversary S can distinguish between the two is by sending a query of the form $(1, a', \text{Verify}', \text{VK}', m', \sigma')$ such that $\text{Verify}' = \mathcal{S}.\text{Verify}$, $\text{VK}' = \text{VK}$, $\text{Verify}'(\text{VK}', m', \sigma') = 1$ and $P_u(m') = 0$. Note that m' was not queried during the signature phase, since $P_u(m') = 0$. This implies (m', σ') is a valid forgery, thereby breaking the signature scheme $\mathcal{S}$. ■

Claim B.6. For any adversary Att,

$$\text{Adv}_{\text{Att}}^5 = \text{Adv}_{\text{Att}}^6.$$

Proof. The only difference between Game 5 and Game 6 is the choice of $c_{1,b}$. In Game 5, $c_{1,b} \leftarrow \mathbb{Z}_{\phi(N)}$, while in Game 6, $c'_{1,b} \leftarrow \mathbb{Z}_{\phi(N)}$ and $c_{1,b} = c'_{1,b} \cdot e^{-1} \pmod{\phi(N)}$. However, the distributions $\mathcal{D}_1 = \{(c, e) | c \leftarrow \mathbb{Z}_{\phi(N)}\}$ and $\mathcal{D}_2 = \{(c \cdot e^{-1} \pmod{\phi(N)}, e) | c \leftarrow \mathbb{Z}_{\phi(N)}\}$ are identical, which implies that Game 5 and Game 6 are identical. ■

Claim B.7. Assuming $\mathcal{O}$ is a secure indistinguishability obfuscator, for any PPT adversary Att,

$$\text{Adv}_{\text{Att}}^6 - \text{Adv}_{\text{Att}}^7 \leq \text{negl}(\lambda).$$

Proof. Let $(SK, VK) \leftarrow \mathcal{S}.\text{Gen}(1^\lambda)$, $\tilde{K} \leftarrow \tilde{F}.\text{setup}(1^\lambda)$, $a \leftarrow \{0, 1\}^\ell$, $\alpha = \text{PRG}(a)$ and $u \leftarrow \text{AdmSample}(1^\lambda, q_1 + n)$. Choose an RSA modulus N , let $e \leftarrow \mathbb{Z}_{\phi(N)}^*$, $v \leftarrow \mathbb{Z}_N^*$ and $c'_{i,b} \leftarrow \mathbb{Z}_{\phi(N)}$. Let $\mathbf{c} = \{c_{i,b}\}$ where $c_{1,b} = c'_{1,b} \cdot e^{-1}$ and $c_{i,b} = c'_{i,b}$ for all $i > 1$. Let $\tilde{\mathbf{c}} = \{\tilde{c}_{i,b}\}$ where $\tilde{c}_{i,b} = c'_{i,b}$ for all i .

To prove this claim, it suffices to show that $\text{Transform-2}_{u,N,v,\mathbf{c},\alpha,SK,\tilde{K}}$ and $\text{Transform-3}_{u,N,v,\tilde{\mathbf{c}},\alpha,SK,\tilde{K},e^{-1}}$ are functionally identical. Let us consider the behavior of the two programs on input $(b, a', \text{Verify}', VK', m', \sigma')$. Let $x' = h(m')$. The only case where Transform-2 and Transform-3 can possibly differ is when $b = 1$, $\text{Verify}'(VK', m', \sigma') = 1$, $\text{Verify}' = \mathcal{S}.\text{Verify}$, $VK' = VK$ and $P_u(m') = 1$. Transform-2 outputs $v^{\prod c_{i,z'_i}} \pmod{N}$ while Transform-3 outputs $v^{\prod \tilde{c}_{i,z'_i} \cdot e^{-1}} \pmod{N}$. However, note that $\prod_i c_{i,z'_i} = (\prod_i \tilde{c}_{i,z'_i}) \cdot e^{-1}$ since $c_{1,b} = c'_{1,b} \cdot e^{-1} = \tilde{c}_{1,b}$. This concludes our proof. $\blacksquare$

Claim B.8. For any adversary Att,

$$\text{Adv}_{\text{Att}}^7 - \text{Adv}_{\text{Att}}^8 \leq \text{negl}(\lambda).$$

Proof. Let us consider the two distributions $\mathcal{D}_1 = \{a | a \leftarrow \mathbb{Z}_{\phi(N)}\}$ and $\mathcal{D}_2 = \{a \pmod{\phi(N)} | a \leftarrow \mathbb{Z}_N\}$. The statistical distance between $\mathcal{D}_1$ and $\mathcal{D}_2$ is $(p + q - 1)/N$, where $N = pq$. Since $p, q \in \Theta(2^\lambda)$, the statistical distance between $\mathcal{D}_1$ and $\mathcal{D}_2$ is negligible in λ .

Next, note that the distributions $\mathcal{D}'_1 = \{(a, e) | a \leftarrow \mathbb{Z}_{\phi(N)}\}$, $\mathcal{D}'_2 = \{(a \cdot e \pmod{\phi(N)}, e) | a \leftarrow \mathbb{Z}_{\phi(N)}\}$ and $\mathcal{D}'_3 = \{(a \cdot e + 1 \pmod{\phi(N)}, e) | a \leftarrow \mathbb{Z}_{\phi(N)}\}$ are identical. As a result, Game 6 and Game 7 are statistically indistinguishable. $\blacksquare$

Claim B.9. Assuming $\mathcal{O}$ is a secure indistinguishability obfuscator, for any PPT adversary Att,

$$\text{Adv}_{\text{Att}}^8 - \text{Adv}_{\text{Att}}^9 \leq \text{negl}(\lambda).$$

Proof. Let N be an RSA modulus, $e \leftarrow \mathbb{Z}_{\phi(N)}^*$, $a_{i,b} \leftarrow \mathbb{Z}_N$ and $u \leftarrow \text{AdmSample}(1^\lambda, q_1 + n)$, $a \leftarrow \{0, 1\}^\ell$, $\alpha = \text{PRG}(a)$, $(SK, VK) \leftarrow \mathcal{S}.\text{Gen}(1^\lambda)$ and $\tilde{K} \leftarrow \tilde{F}.\text{setup}(1^\lambda)$. Let $c_{i,b} = e \cdot a_{i,b} \pmod{\phi(N)}$ if $u_i = b$, else $c_{i,b} = e \cdot a_{i,b} + 1 \pmod{\phi(N)}$. In order to prove this claim, it suffices to prove that the programs $\text{Transform-3}_{u,N,v,\mathbf{c},\alpha,SK,\tilde{K},e^{-1}}$ and $\text{Transform-4}_{u,N,v,\mathbf{a},\alpha,SK,\tilde{K},e}$ are functionally identical, and similarly, the programs $\text{Transform-Image-1}_{N,v,\mathbf{c}}$ and $\text{Transform-Image-1}_{N,v,\mathbf{a},e}$ are functionally identical. We will use the following observation.

Observation B.1. Let $v \in \mathbb{Z}_N^*$, $w_i \in \mathbb{Z}$ for $i \leq n$. Let $w' = w \pmod{\phi(N)}$. Then $v^{\prod_i w_i} = v^{\prod_i w'_i}$.

This follows from the fact that $v^{\phi(N)} = 1$.

Let us first consider the circuits $\text{Transform-3}_{u,N,v,\mathbf{c},\alpha,SK,\tilde{K},e^{-1}}$ and $\text{Transform-4}_{u,N,v,\mathbf{a},\alpha,SK,\tilde{K},e}$. On input $(b, a', \text{Verify}', VK', m', \sigma')$, the only case Transform-3 and Transform-4 can possibly differ is when $b = 1$, $\text{Verify}'(VK', m', \sigma') = 1$, and either $\text{Verify}' || VK' \neq \mathcal{S}.\text{Verify} || VK$ or $P_u(m') = 1$. Let $y = \mathcal{S}.\text{Verify} || VK$, $y' = \text{Verify}' || VK'$ and $z' = h(m') || \text{Verify}' || VK'$. Either there exists an index i' such that $u_{i'} = h(m')_{i'}$, in which case set j' to be the first such index, or there exists an index i' such that $y_{i'} \neq y'_{i'}$, in which case set $j' = d_1 + i'$. Note that $c_{j',z'_{j'}} = e \cdot a_{j',z'_{j'}}$.

$$\begin{aligned} & \text{Transform-3}_{u,N,v,\mathbf{c},\alpha,SK,\tilde{K},e^{-1}}(\text{Verify}', VK', m', \sigma') \\ &= v^{\prod c_{j,z'_j} \cdot e^{-1}} \\ &= v^{\prod_{j \neq j'} c_{j,z'_j} \cdot c_{j',z'_{j'}} \cdot e^{-1}} \\ &= v^{\prod_{j \neq j'} c_{j,z'_j} \cdot a_{j',z'_{j'}} \pmod{\phi(N)}} \\ &= \text{Transform-4}_{u,N,v,\mathbf{a},\alpha,SK,\tilde{K},e}(\text{Verify}', VK', m', \sigma') \end{aligned}$$

where the last step follows from Observation B.1.

Let us now consider **Transform-Image-1** and **Transform-Image-2**. This case follows directly from Observation B.1, since the only difference between the two programs is that **Transform-Image-1** _{N,v,c} has c hardwired, while in **Transform-Image-2** _{N,v,a,e} , a is hardwired. ■

Claim B.10. Assuming RSA is secure, for any PPT adversary Att ,

$$\text{Adv}_{\text{Att}}^9 \leq \text{negl}(\lambda).$$

Proof. Suppose there exists a PPT adversary Att such that $\text{Adv}_{\text{Att}}^9 = \epsilon$. We will construct a PPT algorithm $\mathcal{B}$ that breaks the RSA assumption with advantage ϵ .

$\mathcal{B}$ receives as input N, e, v . It chooses $(\text{SK}, \text{VK}) \leftarrow \mathcal{S}.\text{Gen}(1^\lambda)$. It chooses $\tilde{K} \leftarrow \tilde{F}.\text{setup}(1^\lambda)$, $a \leftarrow \{0, 1\}^\ell$, sets $\alpha = \text{PRG}(a)$. It chooses $u \leftarrow \text{AdmSample}(1^\lambda, q_1 + n)$, $a_{i,b} \leftarrow \mathbb{Z}_N$. It sends $\text{PP} = (\mathcal{O}(\text{Transform-4}_{N,v,a,\alpha,\text{SK},\tilde{K},e}), \mathcal{O}(\text{Transform-Image-2}_{N,v,a,e}), e)$ and VK to Att .

For each signing query x_i , $\mathcal{B}$ checks that $P_u(x_i) = 1$ and sends $\sigma_i = \mathcal{S}.\text{Sign}(\text{SK}, x_i)$ to Att .

Finally, Att outputs a forgery σ_{agg} along with n tuples $\{\text{Verify}_i, \text{VK}_i, m_i\}$. Let $z_i = h(m_i) \parallel \mathcal{S}.\text{Verify} \parallel \text{VK}$, $y = \mathcal{S}.\text{Verify} \parallel \text{VK}$, $y_i = \text{Verify}_i \parallel \text{VK}_i$ and $\mathcal{I} = \{i \mid \text{Verify}_i = \mathcal{S}.\text{Verify}, \text{VK}_i = \text{VK}\}$.

If Att wins, then $\exists i^* \in [n]$ such that $\text{Verify}_{i^*} = \mathcal{S}.\text{Verify}$, $\text{VK}_{i^*} = \text{VK}$, m_{i^*} was not queried during the signature phase, $P_u(m_{i^*}) = 0$, for all $i \in \mathcal{I}, i \neq i^*$ $P_u(m_i) = 1$, and $\sigma_{\text{agg}}^e = \prod_i \text{Transform-Image-2}_{N,v,a,e}(\text{Verify}_i, \text{VK}_i, m_i)$.

Let us consider $\text{Transform-Image-2}_{N,v,a,e}(\text{Verify}_i, \text{VK}_i, m_i)$ for some $i \neq i^*$. For $j \leq d_1$, let $c_{j,b} = e \cdot a_{j,b}$ if $u_j = b$, else $c_{j,b} = e \cdot a_{j,b} + 1$. For $j > d_1$, let $c_{j,b} = e \cdot a_{j,b}$ if $y_{j-d_1} \neq b$, else $c_{j,b} = e \cdot a_{j,b} + 1$. If $y = y'$, let $j' \in [d_1]$ be the first index such that $u_{j'} = z_{i,j'}$. Else, let j'' be the first index such that $y_{j''} \neq y_{i,j''}$ and set $j' = d_1 + j''$. Then,

$$\begin{aligned} & \text{Transform-Image-2}_{N,v,a,e}(\text{Verify}_i, \text{VK}_i, m_i) \\ &= v^{\prod c_{j,z_{i,j}}} \pmod{N} \\ &= v^{(\prod_{j \neq j'} c_{j,z_{i,j}}) \cdot e \cdot a_{j',z_{i,j'}}} \pmod{N} \\ &= v^{e \cdot \tau_i} \pmod{N} \text{ where } \tau_i = a_{j',z_{i,j'}} \cdot \left(\prod_{j \neq j'} c_{j,z_{i,j}} \right). \end{aligned}$$

On the other hand, if we consider the term corresponding to i^* , then

$$\begin{aligned} & \text{Transform-Image-2}_{N,v,c,e} \\ &= v^{\prod_j c_{j,z_{i^*,j}}} \pmod{N} \\ &= v^{\prod_j (e \cdot a_{j,z_{i^*,j}} + 1)} \pmod{N} \\ &= v \cdot v^{e \cdot \tau_{i^*}} \pmod{N} \text{ for some } \tau_{i^*} \text{ that can be efficiently computed using } e, a_{i,b}. \end{aligned}$$

Hence, $\sigma_{\text{agg}}^e = v \cdot (v^{\sum \tau_i})^e \pmod{N}$. $\mathcal{B}$ finally outputs $x = \sigma_{\text{agg}} / (v^{\sum \tau_i}) \pmod{N}$, and wins with advantage ϵ . ■

Therefore, assuming $\mathcal{O}$ is a secure VBB obfuscator for class $\mathcal{C}$, F is a selectively secure puncturable PRF, $\tilde{F}$ is a secure PRF, PRG is a secure pseudorandom generator and the RSA assumption holds, the construction described in Section B is adaptive secure with respect to all length-qualified signature schemes.

Machine-Generated Algorithms, Proofs and Software for the Batch Verification of Digital Signature Schemes

Joseph A. Akinyele^{*§}

Matthew Green^{*†}

Susan Hohenberger^{*‡}

Matthew W. Pagano^{*§}

February 26, 2014

Abstract

As devices everywhere increasingly communicate with each other, many security applications will require low-bandwidth signatures that can be processed quickly. Pairing-based signatures can be very short, but are often costly to verify. Fortunately, they also tend to have efficient batch verification algorithms. Finding these batching algorithms by hand, however, can be tedious and error prone.

We address this by presenting AutoBatch, an automated tool for generating batch verification code in either Python or C++ from a high level representation of a signature scheme. AutoBatch outputs both software and, for transparency, a LaTeX file describing the batching algorithm and arguing that it preserves the unforgeability of the original scheme.

We tested AutoBatch on over a dozen pairing-based schemes to demonstrate that a computer could find competitive batching solutions in a reasonable amount of time. In particular, it found an algorithm that is faster than a batching algorithm from Eurocrypt 2010. Another novel contribution is that it handles *cross-scheme* batching, where it searches for a common algebraic structure between two distinct schemes and attempts to batch them together.

In this work, we expand upon our paper on AutoBatch appearing in ACM CCS 2012 [2] in a number of ways. We add a new loop-unrolling technique and show that it helps cut the batch verification cost of one scheme by roughly half. We describe our pruning and search algorithms in greater detail, including pseudocode and diagrams. All experiments were also re-run using the RELIC pairing library. We compare those results to our earlier results using the MIRACL library, and discuss why RELIC outperforms MIRACL in all but two cases. Automated proofs of several new batching algorithms are also included.

AutoBatch is a useful tool for cryptographic designers and implementors, and to our knowledge, it is the first attempt to outsource to machines the design, proof writing and implementation of signature batch verification schemes.

1 Introduction

We anticipate a future where computers are everywhere as an integrated part of our surroundings, continuously exchanging messages, e.g., sensor networks, smartphones, vehicular communications. For these systems

^{*}Johns Hopkins University, {akinyelj, mgreen, susan, mpagano}@cs.jhu.edu

[†]Supported in part by the Defense Advanced Research Projects Agency (DARPA) and the Air Force Research Laboratory (AFRL) under contract FA8750-11-2-0211, the Office of Naval Research under contract N00014-11-1-0470, NSF grant CNS 1010928 and HHS 90TR0003/01. Its contents are solely the responsibility of the authors and do not necessarily represent the official views of the HHS.

[‡]Supported in part by the Defense Advanced Research Projects Agency (DARPA) and the Air Force Research Laboratory (AFRL) under contract FA8750-11-2-0211, the Office of Naval Research under contract N00014-11-1-0470, NSF grant CNS 1154035, and a Microsoft Faculty Fellowship. Applying to all authors, the views expressed are those of the authors and do not reflect the official policy or position of the Department of Defense or the U.S. Government.

[§]Supported in part by NSF grant CNS 1010928 and HHS 90TR0003/01.

to work properly, messages must carry some form of authentication, and yet the system requirements on this authentication are particularly demanding. Applications such as vehicular communications [23, 60], where cars communicate with each other and the highway infrastructure to report on road conditions, traffic congestion, etc., require both that signatures be short (due to the limited spectrum available) and that many messages from different sources can be processed quickly.

Pairing-based signatures are attractive due to their small size, but they often carry a costly verification procedure. Fortunately, these schemes also lend themselves well to *batch verification*, where valuable time is saved by processing many messages at once. E.g., Boneh, Lynn and Shacham [15] presented a 160-bit signature together with a batching algorithm over signatures by the same signer, where verification time could be reduced from 47.6ms to 2.28ms per signature in a batch of 200 [28] — a 95% saving!

To prepare for a future of ubiquitous messaging, we would like batching algorithms for as many pairing-based schemes as possible. Designing batch verification algorithms by hand, however, is challenging. First, it can be tedious. It requires knowledge of many batching rules and exploration of a potentially huge space of algebraic manipulations in the hunt for a good candidate algorithm. Second, it can be error prone. In Section 1.3, we discuss both the success and failure of the past fifteen years in batching digital signatures. The clear lesson is that mistakes are common and that even when generic methods for batching have been suggested, they have often been misapplied (e.g., a critical step is forgotten). This paper demonstrates that it is feasible for humans to turn over some of the design, proof writing and implementation work in batch verification to machines.

1.1 Our Contributions

We present AutoBatch,¹ an automated tool that transforms a high-level description of a signature scheme² into an optimized batch verification program in either Python or C++. This high-level specification is written in a language called *Scheme Description Language* (SDL), which is designed specifically for automation. AutoBatch takes as input an SDL specification of a signature scheme and searches for a batching algorithm by repeatedly applying a combination of novel and existing batching techniques. Because some loops or other infinite paths could occur, AutoBatch prunes its search using a set of carefully designed heuristics. Despite these heuristics, AutoBatch is not guaranteed to terminate but we conjecture that it does in practice. Our tool produces a modified SDL file and executable code, which includes logic for altering the behavior of the batching algorithm based on its input size or past input.

To our knowledge, this is the first attempt to automatically identify when certain batching techniques are applicable and to apply them in a secure manner. Importantly, the way in which we combine these techniques and optimizations preserves the unforgeability of the original scheme. Specifically, with all but a negligible probability, the batch verifier will accept a batch S of signatures if and only if every $s \in S$ would have been accepted by the individual verification algorithm. AutoBatch also produces a machine-generated LaTeX file that specifies each technique applied and an argument for why security is preserved.

AutoBatch was tested on several pairing-based schemes. It produced the first batching algorithms, to our knowledge, for the Camenisch-Lysyanskaya [20] and Hohenberger-Waters [36] signatures.³ It also discovered a faster algorithm for batching the proofs of the verifiable random functions (VRF) of Hohenberger and Waters [37]. Moreover, AutoBatch is able to handle batches with more than one type of signature. Indeed, we found that the Hess [35] and Cha-Cheon [24] identity-based signatures can be processed twice as fast when batched together compared to sorting by type and batching within the type. The capability to do *cross-scheme* batching is a novel contribution of this paper, and we feel could be of great value for applications, such as mail servers, which may encounter many signature types at once.

AutoBatch is a tool with many applications for both existing and future signature schemes. It helps to enable the secure, but rapid processing of authenticated messages, which we believe will be of increasing importance in a wide-variety of future security applications.

¹The AutoBatch source and test cases described herein are publicly available at <https://github.com/JHUISI/auto-tools>.

²Optionally, one can start with an existing implementation, from which AutoBatch will extract a representation.

³It also produced a candidate batching scheme for the Waters dual-system [66] signatures, although this signature scheme does not have perfect correctness and therefore our techniques do not immediately apply to it. See Section 2.1.1 for more.

1.2 Overview of Our Approach

We present a detailed explanation of AutoBatch in §3. In this section and in Figure 1 we provide a brief overview of the techniques. At a high level, AutoBatch is designed to analyze a scheme, extract the signature verification equation, and derive working code for a batch verifier. This involves three distinct components:

1. (Optional) A Code Parser, which retrieves the verification equation and variable types from some existing scheme implementation. Our parser assumes that the scheme has been implemented in Python following a specific structure (see Section 3.5 for more details). Given such an implementation, the Parser obtains the signature verification equation and encodes it into an intermediate representation in SDL.
2. A Batcher, which takes as input an SDL file describing a signature verification equation. In addition to the signature verification equation, the Batcher requires details in SDL such as types, variable names of public parameters and signatures, and estimated batch size. It first consolidates the set of individual verification equations into a single equation, then derives a batch verification equation. The Batcher then searches through a series of rules, which may be applied repeatedly, to optimize the equation and thus derive a new equation of a batch verifier. The output of the Batcher is a second SDL file, which includes the individual and batch verifiers, along with an analysis of the batcher’s estimated running time. For transparency, the Batcher optionally outputs a LaTeX file that can be compiled into a human-readable document describing the batching algorithm and arguing that it maintains the unforgeability of the original scheme.
3. A Code Generator, which takes the output of the Batcher and generates working source code to implement the batch verifier. The batch verifier implementation includes group membership checks, a recursive divide-and-conquer process to handle batches that contain *invalid* signatures, and additional logic to identify cases where individual verification is likely to outperform batching. The user can choose either Python or C++ as the output language, either building on the MIRACL [59] or RELIC [4] library.

There are two usage scenarios for AutoBatch. The most common may be that a user begins with a hand-coded SDL file and feeds this directly into the Batcher. Since SDL files are human-readable ASCII-based files containing a mathematical representation of the scheme, some developers may prefer to implement new schemes directly in this language, which is agnostic to the programming language of the final implementation.

As a second scenario, if the user has a working implementation of the scheme in Charm [1], then she can save time. This program can be given to the Code Parser, which will extract the necessary information from the code to generate an SDL file. Charm is a Python and C++ based prototyping framework created by Akinyele et al. [1] that provides infrastructure for developing advanced cryptographic schemes. There is already a library of pairing-based signatures publicly available in Charm/Python, so we provide this as a second interface option to our tool.

1.3 Related Work

Computer-aided security is a goal of high importance. Recently, the best paper award at CRYPTO 2011 was given to Barthe, Grégoire, Heraud and Zanella Béguelin [10] for their invention of EasyCrypt, an automated tool for generating security proofs of cryptographic systems from proof sketches. The reader is referred there for a summary of efforts to automate the verification of cryptographic security proofs.

In 1989, batch cryptography was introduced by Fiat [29] for a variant of RSA. In 1994, an interactive batch verifier for DSA presented in an early version of [55] was broken by Lim and Lee [44]. In 1995, Lai and Yen proposed a new method for batch verification of DSA and RSA signatures [41], but the RSA batch verifier was broken five years later by Boyd and Pavlovski [17]. In 1998, two batch verification techniques were presented for DSA and RSA [32, 33] but both were later broken [17, 38, 39]. The same year, Bellare, Garay and Rabin took the first systematic look at batch verification [11] and presented three generic methods for batching modular exponentiations, one of which is called the *small exponents test*. Unfortunately, in 2000, Boyd and Pavlovski [17] published attacks against various batching schemes which were using the small exponents test incorrectly. In 2003-2004, several batch verification schemes based on bilinear maps (a.k.a.,

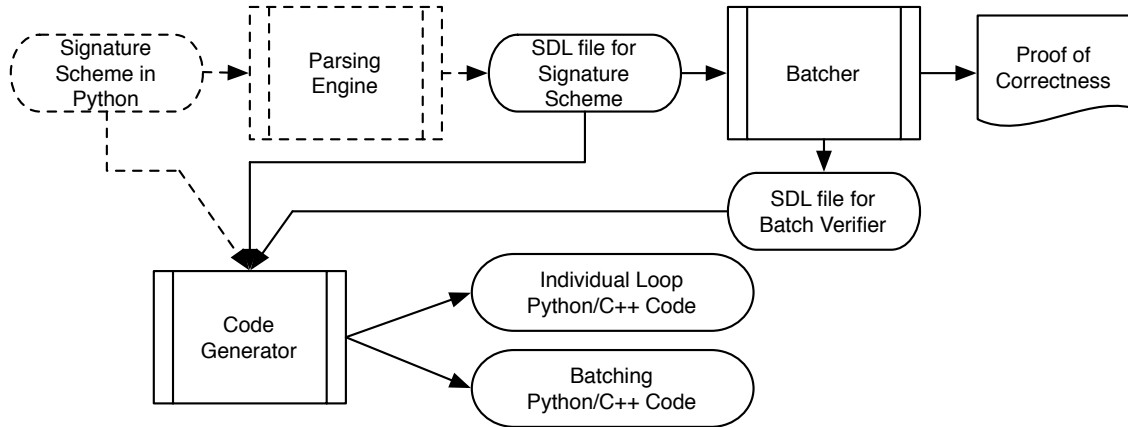


Figure 1: The flow of AutoBatch. The input is a signature scheme comprised of key generation, signing and verification algorithms, represented in the domain-specific SDL language. The scheme is processed by a Batcher, which applies the techniques and optimizations from Section 3 to produce a new SDL file containing a *batch verification* algorithm. Optionally, the Batcher outputs a proof of correctness (as a PDF typeset using LaTeX) that explains, line by line, each technique applied and its security justification. Finally, the Code Generator produces executable C++ or Python code implementing both the resulting batch verifier, and the original (unbatched) verification algorithm. An optional component, the Parsing Engine, allows for the automatic derivation of SDL inputs based on existing scheme implementations.

pairings) were proposed [24, 68, 70, 71] but all were later broken by Cao, Lin and Xue [22]. In 2006, a method was given for identifying invalid signatures in RSA-type batches [43], but it was also flawed [64].

It is natural to ask what the source of the errors were in these papers. In several cases, the mathematics of the scheme were simply unsound and the proof of correctness was either missing or lacking in rigor. However, there were two other common problems. One was that the paper claimed *in English* to be doing batch verification, but the security definition provided in the paper was insufficient to establish this guarantee. Most commonly this matched the strictly weaker *screening* guarantee; see [19] for more. A second problem was more insidious: the security definition and proof were “correct”, but the scheme was still subject to a practical attack because the authors started the proof by explicitly *assuming* that elements of the signature were members of certain algebraic groups and this was not a reasonable assumption to make in practice. Boyd and Pavlovski [17] provide numerous examples of this case.

AutoBatch addresses these common pitfalls. It uses one security definition (in Section 2.1) and provides a proof of correctness for every algorithm it outputs relative to this definition (in Section 3.3), where no assumptions about the algebraic structure of the input are made and therefore any necessary tests are explicitly performed by the algorithm.

In addition to the works on batch verification mentioned above, we mention a few more. Shacham and Boneh presented a modified version of Fiat’s batch verifier for RSA to improve the efficiency of SSL handshakes on a busy server [61]. Boneh, Lynn and Shacham provided a single-signer batch verifier for BLS signatures [15]. Camenisch, Hohenberger and Pedersen [19] gave multiple-signer batch verifiers for Waters identity-based signatures [65] and a novel construction. Ferrara, Green, Hohenberger and Pedersen outlined techniques for batching pairing-based signatures and showed how to batch group and ring signatures [28]. Blazy, Fuchsbaauer, Izabachéne, Jambert, Sibert and Vergnaud [12] applied batch verification techniques to the Groth-Sahai zero-knowledge proof system as well as group signatures and anonymous credential systems relying on them, obtaining significant savings.

Law and Matt describe methods for identifying invalid signatures in a batch [42, 50, 51].

Lastly, there have been several research efforts toward automatically generating cryptographic protocols and executable code. This compiler-like approach has been applied to cryptographic applications such as

security protocols [40, 45, 46, 57, 63], optimizations to software implementations involving elliptic-curve cryptography [9] and bilinear-map functions [56], secure two-party computation [34, 48, 49], and zero-knowledge proofs [3, 5–7, 21, 30, 52].

2 Background

Definition 2.1 (A Digital Signature) A digital signature scheme is a tuple of probabilistic polynomial-time (p.p.t.) algorithms $(\text{Gen}, \text{Sign}, \text{Verify})$:

1. $\text{Gen}(1^\lambda) \rightarrow (pk, sk)$: the key generation algorithm takes as input the security parameter 1^λ and outputs a pair of keys (pk, sk) .
2. $\text{Sign}(sk, m) \rightarrow \sigma$: the signing algorithm takes as input a secret key sk and a message m from the message space and outputs a signature σ .
3. $\text{Verify}(pk, m, \sigma) \rightarrow \{0, 1\}$: the verification algorithm takes as input a public key pk , a message m and a purported signature σ , and outputs a bit indicating the validity of the signature.

A scheme is typically said to be correct (or perfectly correct) if for all $\text{Gen}(1^\ell) \rightarrow (pk, sk)$,

$$\text{Verify}(pk, m, \text{Sign}(sk, m)) = 1 \text{ for all } m.$$

That is, a scheme is correct if all honestly generated signatures pass the verification test. Our focus will be on perfectly correct schemes, however, we discuss in Section 2.1.1 the implications for batch verification if some correctness error is allowed.

A scheme is defined to be *unforgeable* as follows [31]: Let $\text{Gen}(1^\ell) \rightarrow (pk, sk)$. Suppose (m, σ) is output by a p.p.t. adversary with access to a signing oracle $\mathcal{O}_{sk}(\cdot)$ and input pk . Then the probability that m was not queried to $\mathcal{O}_{sk}(\cdot)$ and yet $\text{Verify}(pk, m, \sigma) = 1$ must be negligible in ℓ .

In this work, we explore three variants:

1. **Identity-Based Signatures** [62]: Gen is executed by a master authority who publishes pk and uses sk to generate signing keys for users according to their public identity string, e.g., email address. To verify a signature on a given message, one only needs the public key of the master authority and the public identity string of the purported signer.
2. **Privacy Signatures**: Group [26] and ring [58] signatures are associated with a group of users, where verification shows that at least one member of the group signed the message, but it is difficult to tell who.
3. **Verifiable Random Functions** [53]: A VRF is a pseudo-random function, where the computing party publishes a public key pk and then can offer a short non-interactive *proof* that the function was correctly evaluated for a given input. This proof can be viewed as a signature by the computing party on the input to the pseudo-random function.

2.1 The Basics of Batch Verification for Signatures

Our security focus here is not directly on unforgeability [31]. Rather we are interested in designing batch verification algorithms that accept a set of signatures *if and only if* each signature would have been accepted by its verification algorithm individually (except perhaps with a negligible probability).⁴ *If an input scheme is unforgeable, then our batching algorithm will preserve this property in the output scheme.* If an insecure scheme is provided as input, then all bets are off on the output.

⁴We assume perfectly correct schemes here.

Specifically, we consider the case where we want to quickly verify a set of signatures on possibly different messages by possibly different signers. The input is $\{(t_1, m_1, \sigma_1), \dots, (t_n, m_n, \sigma_n)\}$, where t_i specifies the verification key against which σ_i is purported to be a signature on message m_i . It is important to understand that here one or more *signers* may be maliciously colluding against the batch verifier.

We recall the definition of batch verification from Bellare, Garay and Rabin [11] as extended in [19] to deal with multiple signers. We note that this definition is well specified for perfectly correct schemes, but not for schemes that allow some correctness error. We discuss this further shortly.

Definition 2.2 (Batch Verification of Signatures) *Let ℓ be the security parameter. Suppose $(\text{Gen}, \text{Sign}, \text{Verify})$ is a signature scheme with perfect correctness, $k, n \in \text{poly}(\ell)$, and $(pk_1, sk_1), \dots, (pk_k, sk_k)$ are generated independently according to $\text{Gen}(1^\ell)$. Let $PK = \{pk_1, \dots, pk_k\}$. We call a probabilistic algorithm *Batch* a batch verification algorithm when the following conditions hold:*

- *If $pk_{t_i} \in PK$ and $\text{Verify}(pk_{t_i}, m_i, \sigma_i) = 1$ for all $i \in [1, n]$, then $\text{Batch}((pk_{t_1}, m_1, \sigma_1), \dots, (pk_{t_n}, m_n, \sigma_n)) = 1$.*
- *If $pk_{t_i} \in PK$ for all $i \in [1, n]$ and $\text{Verify}(pk_{t_j}, m_j, \sigma_j) = 0$ for some $j \in [1, n]$, then $\text{Batch}((pk_{t_1}, m_1, \sigma_1), \dots, (pk_{t_n}, m_n, \sigma_n)) = 0$ except with probability negligible in ℓ , taken over the randomness of *Batch*.*

The above definition can be generalized beyond signatures to apply to any keyed scheme with a perfectly-correct verification algorithm. This includes zero-knowledge proofs, verifiable random functions, and variants of regular signatures, such as identity-based, attribute-based, ring, group, aggregate, etc. The above definition requires that signing keys be generated honestly. In practice, users could register their keys and prove some necessary properties of the keys at registration time [8].

2.1.1 On Schemes with a Correctness Error

The standard definition for signature batch verification (as presented in Definition 2.2)⁵ assumes that the basic signature scheme has perfect correctness. That is, the first part of the definition inherently assumes that all valid signatures will pass the individual verification test. This is the case for the majority of signature schemes as well as all signature schemes that we are aware of being actively used in practice.

However, one could imagine a signature scheme with a negligible or small constant correctness error. One example of a scheme with a negligible correctness error is the Waters09 scheme as derived from the Waters Dual-System IBE [66] using the technique described by Naor [14]. In this scheme, a signature on message m corresponds to the IBE private key on identity m . The verification test operates by choosing a random message m' , encrypting it for identity m , running the decryption algorithm using the signature as the private key, and testing to see that decryption successfully recovers m' . Since the Dual-System IBE [66] has a negligible correctness error in the decryption algorithm, this signature scheme also has a negligible correctness error in verification. This leaves the question: what is the right batching definition for such a scheme?

For a scheme that allows an arbitrary amount of correctness error, the first requirement of Definition 2.2 no longer makes sense. Rather in this setting it seems to us that one could no longer base the batching security on the base signature security, but rather would have to create a new game-based definition that simulated the batching scenario and directly prove that the algorithm matches the definition. Direct proofs of this sort are currently beyond our ability to automate.

One might instead narrow the focus to schemes that allow at most a negligible correctness error. In this case, we suggest relaxing both of the batching requirements by a negligible probability taken over the randomness of the *individual and batch* verification algorithms. We leave as an open problem a formal treatment of batching for schemes in this class.

We tested AutoBatch on one scheme with a correctness error, Waters09 [66], because its complication made it a challenging test case. We report on the candidate batching algorithm we found in Section 4,

⁵We added the restriction to perfect correctness in Definition 2.2. It was assumed in prior works but not always made explicit.

although we note there and in Appendix D that our automated proofs were only written to handle schemes with perfect correctness. This is a correction over the conference version of this work [2] which did not make this distinction.

2.2 Algebraic Setting

Bilinear Groups. Let $\mathbb{G}_1$, $\mathbb{G}_2$ and $\mathbb{G}_T$ be groups of prime order q . A map $e : \mathbb{G}_1 \times \mathbb{G}_2 \rightarrow \mathbb{G}_T$ is an admissible bilinear map (or pairing) if it satisfies the following three properties:

1. Bilinearity: for all $g \in \mathbb{G}_1$, $h \in \mathbb{G}_2$, and $a, b \in \mathbb{Z}_q$, it holds that $e(g^a, h^b) = e(g, h)^{ab}$.
2. Non-degeneracy: if g and h are generators of $\mathbb{G}_1$ and $\mathbb{G}_2$, respectively, then $e(g, h)$ is a generator of $\mathbb{G}_T$.
3. Efficiency: there exists an efficiently computable function that given any $g \in \mathbb{G}_1$ and $h \in \mathbb{G}_2$, computes $e(g, h)$.

An admissible bilinear map generator **BSetup** is an algorithm that on input a security parameter 1^ℓ , outputs the parameters for a bilinear group $(q, g, h, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e)$ such that groups of prime order $q \in \Theta(2^\ell)$, $\mathbb{G}_1$, $\mathbb{G}_2$ and $\mathbb{G}_T$ are groups of order q where g generates $\mathbb{G}_1$, h generates $\mathbb{G}_2$ and $e : \mathbb{G}_1 \times \mathbb{G}_2 \rightarrow \mathbb{G}_T$ is an admissible bilinear map.

The above bilinear map is called *asymmetric* and our implementations use this highly efficient setting. We also consider *symmetric* maps where there is an efficient isomorphism $\psi : \mathbb{G}_1 \rightarrow \mathbb{G}_2$ (and vice versa) such that a symmetric map $\hat{e}$ is defined as $\hat{e} : \mathbb{G}_1 \times \psi(\mathbb{G}_1) \rightarrow \mathbb{G}_T$. We abstractly treat symmetric groups equally ($\mathbb{G}_1 = \mathbb{G}_2$) for simplicity.

Testing Membership in Bilinear Groups. When batching, it is critical to test that the elements of each signature are members of the appropriate algebraic group. Boyd and Pavlovski [17] demonstrated efficient attacks on batching algorithms for DSA signature verification which omitted a subgroup membership test.

In this paper, we must test membership in bilinear groups. We require that elements of purported signatures are members of $\mathbb{G}_1$ and *not*, say, members of $E(\mathbb{F}_p) \setminus \mathbb{G}_1$. Determining whether some data represents a point on a curve is easy. The question is whether it is in the correct subgroup. If the order of $\mathbb{G}_1$ is a prime q , one option is to verify that an element y is in $\mathbb{G}_1$ by checking that $y^q \bmod q = 1$ [19]. Although this costs an extra modular exponentiation per group element, this will largely be dwarfed by the savings from reducing the total pairings, as experimentally verified first by Ferrara et al. [28] and confirmed by our tests.

2.3 Batch Verification in Bilinear Groups

Let us recall from [28] the formal definition of a *bilinear-based* (or pairing-based) batch verifier. A pairing-based verification equation is represented by a *generic pairing-based claim* X corresponding to a boolean relation of the following form: $\prod_{i=1}^k e(f_i, h_i)^{c_i} \stackrel{?}{=} A$, for $k \in \text{poly}(\tau)$ and $f_i \in \mathbb{G}_1, h_i \in \mathbb{G}_2$ and $c_i \in \mathbb{Z}_q^*$, for each $i = 1, \dots, k$. A pairing-based verifier **Verify** for a generic pairing-based claim is a probabilistic $\text{poly}(\tau)$ -time algorithm which on input the representation $\langle A, f_1, \dots, f_k, h_1, \dots, h_k, c_1, \dots, c_k \rangle$ of a claim X , outputs *accept* if X holds and *reject* otherwise. We define a batch verifier for pairing-based claims.

Definition 2.3 (Bilinear-based Batch Verifier)

Let $\text{BSetup}(1^\tau) \rightarrow (q, g_1, g_2, \mathbb{G}_a, \mathbb{G}_b, \mathbb{G}_T, e)$. For each $j \in [1, \eta]$, where $\eta \in \text{poly}(\tau)$, let $X^{(j)}$ be a generic pairing-based claim and let **Verify** be a pairing-based verifier. We define a pairing-based batch verifier for **Verify** as a probabilistic $\text{poly}(\tau)$ -time algorithm which outputs:

- accept if $X^{(j)}$ holds for all $j \in [1, \eta]$;
- reject if $X^{(j)}$ does not hold for any $j \in [1, \eta]$ except with negligible probability.

2.4 Small Exponents Test Applied to Bilinear Groups

Bellare, Garay and Rabin [11] proposed methods for verifying multiple equations of the form $y_i = g^{x_i}$ for $i = 1$ to n , where g is a generator for a group of prime order. One might be tempted to just multiply these equations together and check if $\prod_{i=1}^n y_i = g^{\sum_{i=1}^n x_i}$. However, it would be easy to produce two pairs (x_1, y_1) and (x_2, y_2) such that the product of them verifies correctly, but each individual verification does not, e.g. by submitting the pairs $(x_1 - \alpha, y_1)$ and $(x_2 + \alpha, y_2)$ for any α . Instead, Bellare et al. proposed the following method for batching the verification of these equations, which we will shortly apply to bilinear groups.

The Small Exponents Test of Bellare, Garay and Rabin: Choose exponents δ_i of (a small number of) ℓ_b bits and compute $\prod_{i=1}^n y_i^{\delta_i} = g^{\sum_{i=1}^n x_i \delta_i}$. Then the probability of accepting a bad pair is $2^{-\ell_b}$. The size of ℓ_b is a tradeoff between efficiency and security. (By default in AutoBatch, we set $\ell_b = 80$ bits and select random exponents from the range $[1, 2^\lambda - 1]$. Even though 0 is allowed for the test, we forbid it in our implementation.)

Subsequently, Ferrara, Green, Hohenberger and Pedersen [28] proved that the Small Exponents Test could be securely applied to bilinear groups as well. We recall the following theorem from their work which encapsulates the test as well.

Theorem 2.4 (Small Exponents Test Applied to Bilinear Groups [28]) *Let $\text{BSetup}(1^\tau) \rightarrow (q, g_1, g_2, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e)$ where q is prime. For each $j \in [1, \eta]$, where $\eta \in \text{poly}(\tau)$, let $X^{(j)}$ be a generic claim as in Definition 2.3. For simplicity, assume that $X^{(j)}$ is of the form $A \stackrel{?}{=} Y^{(j)}$ where A is fixed for all j and all the input values to the claim $X^{(j)}$ are in the correct groups. For any random vector $\Delta = (\delta_1, \dots, \delta_\eta)$ of ℓ_b bit elements from $\mathbb{Z}_q$, an algorithm **Batch** which tests the following equation $\prod_{j=1}^\eta A^{\delta_j} \stackrel{?}{=} \prod_{j=1}^\eta Y^{(j)\delta_j}$ is a pairing-based batch verifier that accepts an invalid batch with probability at most $2^{-\ell_b}$.*

In later sections, we will frequently make use of the small exponents tests and rely on the security guarantees of Theorem 2.4 as proven by Ferrara et al. [28].

3 The AutoBatch Toolchain

In this section we summarize the techniques used by AutoBatch to programmatically generate batch verifiers from standard signature schemes. A high level abstraction is provided in Figure 1. The main stages are as follows.

1. Derive the scheme’s SDL representation. The AutoBatch toolchain begins with an SDL representation of a signature scheme. While SDL is not a full programming language, it provides sufficient flexibility to represent most pairing-based signature schemes. We provide a description of the SDL grammar in Appendix E, as well as a description of the SDL semantics and several examples in Appendix F. For developers who already have an existing Charm/Python implementation, we also provide a Parsing Engine that can optionally *derive* an SDL representation directly from this Python code.⁶

2. Apply techniques and optimize the batch verification equation. We first apply a set of techniques designed to convert the SDL signature verification equation into a batch verifier. These techniques optimize the verification equation by combining pairing equations and re-arranging the components to minimize the number of expensive operations. To prevent known attacks, we apply the small exponents test of Bellare, Garay and Rabin [11], and optimize the resulting equation to ensure that all signature elements are in the group with the smallest representation (typically, $\mathbb{G}_1$). Additionally, the Batchifier embeds a recursive *divide-and-conquer* strategy to handle cases where batch verification fails due to invalid signatures. This binary search strategy is borrowed from Law and Matt [42] and could be extended to support other methods

⁶We developed this capability for two reasons. First, there is already a library of pairing-based signature schemes available in Charm/Python (in fact, the number of Charm implementations is greater than all other settings combined). Secondly, we believe that there is value in providing multiple interfaces to our tools, particularly interfaces that work with real implementations.

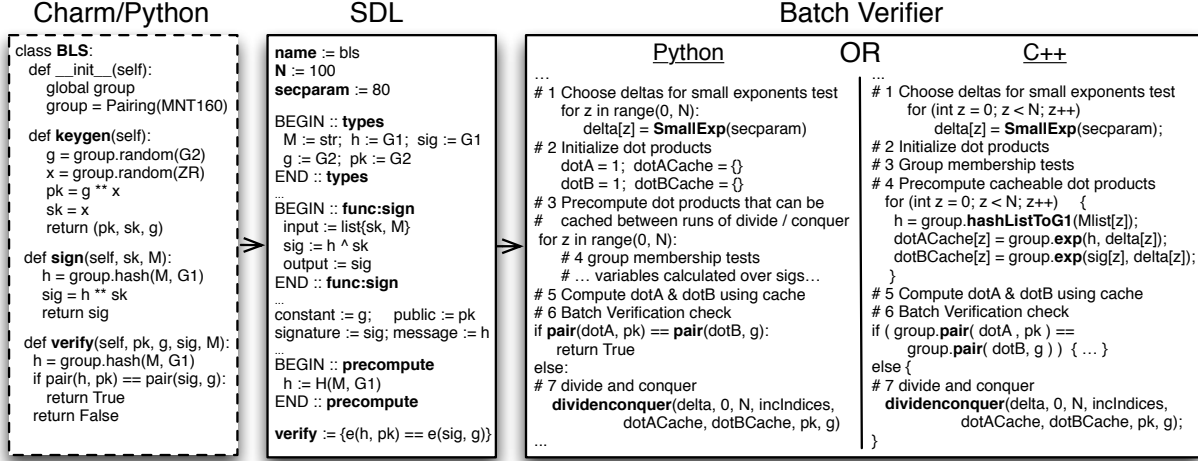


Figure 2: The Boneh-Lynn-Shacham (BLS) signature scheme [15] at various stages in the AutoBatch toolchain. At the left, an initial Charm-Python implementation of the scheme. In the center, an SDL representation of the same scheme, programmatically extracted by the Parsing Engine. At right, a fragment of the resulting batch verifier generated after applying the Batcher and Code Generator.

that outperform this approach. Finally, the output of this phase is a modified SDL file, and (optionally) a human-readable proof that the resulting equation is a secure batch verifier.

3. Evaluate the capabilities of the batch verifier. Given the optimized batching equation produced in the previous step, we estimate the performance of the verifier under various conditions. This is done by counting the operations in the verifier, and deriving an estimate of the runtime based on the expected cost of each mathematical operation (e.g., pairing, exponentiation, multiplication). The cost of each operation is determined via a set of diagnostic tests conducted when the library is initialized.⁷

4. Generate code for the resulting batch verifier. Finally, we translate the resulting SDL file into a working batch verifier. This verifier can be implemented in either Python or C++ using the Charm framework. It implements the SDL-specified batch verification equation as well as the individual verification equation. Based on the calculations of the previous step, the generated code embeds logic to automatically determine *which* verifier is most appropriate for a given dataset (individual or batch). Two fragments of generated code (Python and C++) are shown in Figure 2.

We will now describe each of the above steps in detail.

3.1 Batching and Optimization

Given an SDL file containing the verification equation and variable types, the Batcher first securely consolidates the individual verification equations into a single equation using the small exponents test. Then, the Batcher applies a series of optimizations to the batch verification equation in order to derive an efficient batch verifier. Many of these techniques were first explored in previous works [19,28]. However, the intended audience of those works is *humans* performing manual batching of signatures. Hence, they are in many cases somewhat less “general” than the techniques we describe here.⁸ Furthermore, unlike previous works we are

⁷Obviously these experiments are very specific to the machine and curve parameters on which they are run. Our implementation re-runs these experiments whenever the library is initialized with a given set of parameters.

⁸For example: techniques 2 and 3 of [19] each combine a series of logical operations that are more widely applicable and easily managed by splitting them into finer-grained sub-techniques.

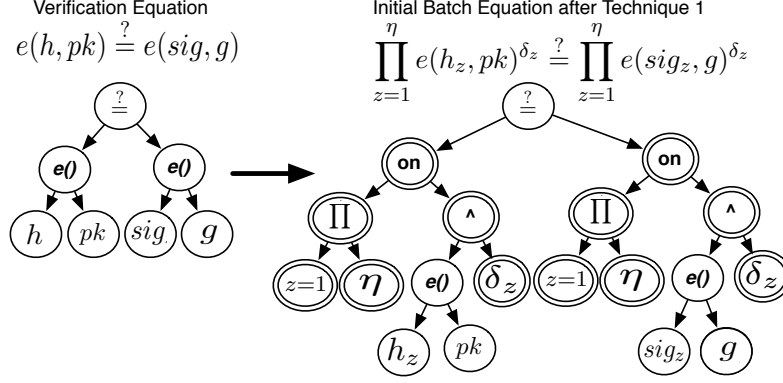


Figure 3: The Boneh-Lynn-Shacham (BLS) signature scheme [15] with same signer and η signatures in a batch. We show the abstract syntax tree (AST) of the unoptimized batch equation after Batchter has applied technique 1 by combining all instances of the verification equations (denoted by Π node) and applying the small exponents test (denoted by δ_z node).

able to programmatically identify when these techniques are applicable, and apply them to the verification equation in a consistent way.

The Batchter assumes that the input will be a collection of η signatures, possibly on different messages and public keys (or identities). To construct a batch verifier, the Batchter first parses and performs type checking on the SDL input file to extract an abstract syntax tree (AST) representing the verification equation. During the type checking, it informs users if there are type mismatches or if the typing information is incomplete in SDL. Next, the Batchter traverses the AST of the verification equation, applying various techniques at various nodes in the tree.

We now list those techniques and provide details on how some of these techniques are implemented on the AST.

Technique 0a: Consolidate the verification equation. Many pairing-based signature schemes actually require the verifier to check more than one pairing equation. During the first phase of the batching process, the batcher applies the small exponents test from [11] to combine these equations into a single verification equation.⁹ A variation of this is *Technique 0b* which is applicable for schemes that utilize for loops in the verification equation (e.g., VRF [37]). If the bounds over the loop are known it might be useful to unroll the loop to allow application of other techniques.

$$\text{Replace for } i = 1 \text{ to } t : e(g, h_i) \stackrel{?}{=} e(c, d_i) \text{ with } e(g, h_1)^{\delta_1} \cdot \dots \cdot e(g, h_t)^{-\delta_t} \stackrel{?}{=} e(c, d_1)^{\delta_1} \cdot \dots \cdot e(c, d_t)^{-\delta_t}$$

Technique 1: Combine equations. Assume we are given η signature instances that can be verified using the consolidated equation from the previous step. We now combine all instances into one equation by applying the Combination Step of [28], which employs as a subroutine the small exponents test. This results in a single verification equation. The correctness of the resulting equation requires that all elements be in the correct subgroup, i.e., that group membership has already been checked. AutoBatch ensures that this check will be explicitly conducted in the final batch verifier program. See Figure 3 for an example.

Technique 2: Move exponents inside the pairing. When a term of the form $e(g_i, h_i)^{\delta_i}$ appears, move the exponent δ_i into $e()$. Since elements of $\mathbb{G}_1$ and $\mathbb{G}_2$ are usually smaller than elements of $\mathbb{G}_T$, this gives a noticeable speedup when computing the exponentiation.

$$\text{Replace } e(g_i, h_i)^{\delta_i} \text{ with } e(g_i^{\delta_i}, h_i)$$

⁹For example, consider two verification conditions $e(a, b) = e(c, d)$ and $e(a, c) = e(g, h)$. These can be verified simultaneously by selecting random δ_1, δ_2 and evaluating the single equation $e(a, b)^{\delta_1} e(c, d)^{-\delta_1} e(a, c)^{\delta_2} e(g, h)^{-\delta_2} = 1$.

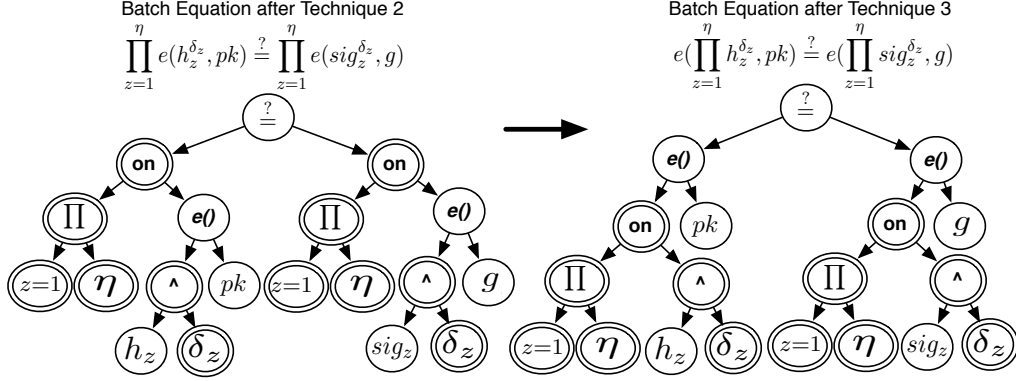


Figure 4: The Boneh-Lynn-Shacham (BLS) signature scheme [15] with same signer and η signatures in a batch. Upon applying technique 1 from Figure 3 to obtain the initial secure batch verifier, the goal is to optimize the equation. We first show the AST of the equation *after* the Batcher has applied technique 2 (move exponents inside the pairing). Then, we show the result of applying technique 3a (move products inside the pairing) to arrive at an optimized batch equation.

Wherever possible, we move the exponent into the group with the lowest exponentiation cost. We identify this group based on a series of operation microbenchmarks that run automatically at code initialization.¹⁰

Technique 3a: Move products inside the pairing. When a term of the form $\prod_{i=1}^{\eta} e(a_i, g)$ with a constant first or second argument appears, move the product inside to reduce the number of pairings from η to 1.

$$\text{Replace } \prod_{i=1}^{\eta} e(a_i, g) \text{ with } e(\prod_{i=1}^{\eta} a_i, g)$$

A special case of this technique is *Technique 3b* where $\eta = 2$. In this case, when two terms share a common first or second argument, they can also be combined. For example:

$$\text{Replace } e(a, g) \cdot e(b, g) \text{ with } e(a \cdot b, g) \text{ where } a \neq 1 \wedge b \neq 1.$$

For a concrete example, we show how techniques 2 and 3a are programmatically applied to the BLS scheme [15] in Figure 4.

Technique 4: Optimize the Waters Hash. A variety of identity-based signature schemes employ a hash function by Waters [65], which can be generalized [25, 54]. Verifying signatures generated by these schemes requires hashing identity strings of the form $V = v_1 v_2 \dots v_z$ where each v_i is a short string. The hash function is evaluated as $u' \prod_{i=1}^z u_i^{v_i}$ where u' and $u_1 u_2 \dots u_z$ are public generators in $\mathbb{G}_1$ or $\mathbb{G}_2$.

When batching η equations containing the Waters hash, one often encounters terms of the form $\prod_{j=1}^{\eta} e(g_j, \prod_{i=1}^z u_i^{v_{ij}})$. This can be rewritten to make the number of pairings independent of the number of equations one wants to batch. This is most useful when $\eta > z$.

$$\text{Replace } \prod_{j=1}^{\eta} e(g_j, \prod_{i=1}^z u_i^{v_{ij}}) \text{ with } \prod_{i=1}^z e(\prod_{j=1}^{\eta} g_j^{v_{ij}}, u_i)$$

In future versions, AutoBatch will output code to switch between the most efficient verifier when $\eta > z$ and $\eta \leq z$.

¹⁰For many common elliptic curves, this is the $\mathbb{G}_1$ base group. However, in some curves the groups $\mathbb{G}_1$ and $\mathbb{G}_2$ have similar operation costs; this may give us some flexibility in modifying the equation.

Technique 5: Distribute products. When a product is applied to two or more terms, distribute the product to each term to allow application of other techniques such as techniques 3 or 4. For example:

$$\text{Replace } \prod_{i=1}^{\eta} (e(a_i, g_i) \cdot e(b_i, h_i)) \text{ with } \prod_{i=1}^{\eta} e(a_i, g_i) \cdot \prod_{i=1}^{\eta} e(b_i, h_i)$$

Technique 6: Move known exponents outside pairing and precompute pairings. In some cases it may be necessary to move exponents outside of a pairing. For example, when $\prod_{i=1}^{\eta} e(g^{a_i}, h^{b_i})$ appears, move the exponents outside of pairing. When multiple such exponents appear, we can pre-compute the sum of $a_i \cdot b_i$ for all η and exponentiate once in $\mathbb{G}_T$.

$$\text{Replace } \prod_{i=1}^{\eta} e(g^{a_i}, h^{b_i}) \text{ with } e(g, h)^{\sum_i (a_i \cdot b_i)}$$

Technique 7: Precompute constant pairings. When pairings have a constant first and second argument, we can simply remove these from the equation and pre-compute them once at the beginning of verification (equivalent to making them a public parameter).

Technique 8: Split pairings. In some rare cases it can be useful to apply technique 3b in reverse: splitting a single pairing into two or more pairings. This temporarily increases the number of pairings in the verification equation, but may be necessary in order to apply subsequent techniques. For example, this optimization is necessary so that we can apply the Waters hash optimization (technique 4) to the ring signature of Boyen [18].

Discussion: Several of the above techniques are quite simple, in that they perform optimizations that would seem “obvious” to an experienced cryptographer. However, many optimizations (e.g., technique 7) *could* have been applied in published algorithm descriptions [20, 37], and yet were not. Moreover, it is a computer and not a human that is performing the search for us, so an important contribution of this work is providing a detailed list of which optimizations we tell the computer to try out and in which order, and verifying that such an approach can find competitive solutions in a reasonable amount of time. This is nontrivial: we discovered that many orderings lead to “dead ends”, where the optimal solution is *not* discovered. We now describe our approach to finding the order of techniques.

3.2 Technique Search Approach

The challenge in automating the batching process is to identify the *order* in which techniques should be applied to a given verifier. This is surprisingly difficult, as there are many possible orderings, many of which require several (possibly repeated) invocations of specific techniques. Moreover, some techniques might actually worsen the performance of the verifier in the hope of applying other techniques to obtain a better solution. An automated search algorithm must balance all of these issues and must also identify the orderings in an efficient manner.

The naive approach to this problem is simply to try all possible combinations up to a certain limit, then identify the best resulting verifier based on an estimate of total running time. This limit can be vastly different as the complexity of the scheme increases. While this approach is feasible for simple schemes, it is quite inefficient for schemes that require the application of several techniques. Moreover, there is the separate difficulty of determining when the algorithm should halt, as the application of one technique will sometimes produce a new equation that is amenable to further optimization, and this process can continue for several operations.

Our Search Approach: Our approach is a “pruned” breadth-first search (PBFS) which utilizes a finite state transition function to constrain the transitions between techniques. This transition function determines which techniques can be applied to the current state and was constructed with our experience of how the optimization techniques work together logically. For instance, if technique 5 is applied to the current state (i.e., distribute products to pairings), then techniques 2-4 most likely will apply given that these techniques

move exponents or products inside pairings. From the current state, only the subset of techniques in which the conditions for the transformation are met are pursued further in the search.

Our search algorithm is broken down into three stages. The first stage of the search is to try technique 0a if there are multiple verification equations. After consolidating the verification equations, we try technique 3b since there may have been pairings with common elements from separate equations. Our intuition for attempting technique 3b in this stage is to combine as many pairings as possible before embarking on the search. The side effect is that it reduces the number of paths explored by the PBFS, thereby making the search more efficient. Moreover, it is useful to attempt technique 7 at this stage and precompute pairings that utilize generators. We then apply technique 1 to combine η instances of the equations to form an initial batch verifier. However, if the scheme specifies a single verification equation, then only technique 1 is applied in the first stage.

The second stage of the search employs the PBFS (starting with technique 2) and terminates when none of the techniques can be applied to the current state of a batch verifier. Each path from the set of ordering paths uncovered during the PBFS is evaluated in terms of total running time. The algorithm selects the path from the candidate paths that provides the highest cost savings. From the selected path, the final (or post-processing) stage of the search attempts to apply technique 0b (unroll loops) if the equation utilizes for loops. We delay testing for technique 0b until the post-processing stage to limit the search space for an efficient batch verifier. If technique 0b is applied, then we always attempt technique 3b given that there may now be pairings that can be further combined.

To prevent infinite loops during our PBFS, the state function disallows the application of certain techniques that might potentially *undo* optimizations. For example, technique 8 performs a reverse split on pairings to allow further optimizations; this might affect technique 3b, which combines pairings that have common elements. Certain combinations of techniques 8 and 3b lead to an infinite cycle that combines and splits the same pairings. Thus, the state function only allows a transition from Technique 8 to 3b to occur once on a given path. We provide the pseudocode of our search in Algorithm 1 & 2 and our state transition function in Table 1.

Our approach is effective and enables efficiently deriving batch verification algorithms. While our approach does not guarantee the optimal batch equation, in practice we rediscover all existing lower bounds on batch verification performance, and in some cases we improve on results developed by humans.

Current State	Next States
(2, $\_$)	{3a, 3b, 4, 5, 6, 7, 8}
(3a, $\_$)	{2, 3b, 4, 5, 6}
(3b, $\_$)	{2, 3a, 3b, 5, 8}
(4, $\_$)	{2, 3a, 3b, 5}
(5, $\_$)	{2, 3a, 4}
(6, $\_$)	{2, 3b, 5}
(7, $\_$)	{2, 3a, 6}
(8, true)	{2, 4, 5, 6}
(8, false)	{2, 4, 5, 3b, 6}

Table 1: This represents the **transitionFun** of Algorithm 2 developed for pruning our breath-first search (PBFS) algorithm. The function accepts as input the current state which represents the technique that was previously applied to the batch equation and whether there exists a transition from technique 8 to 3b along the path. In an effort to ensure that all paths terminate, the function restricts the transition from technique 8 to 3b to occur once on a given path. Although we do not prove that our algorithm is guaranteed to terminate, we conjecture that it does in practice. In fact, it terminated promptly for all of our test cases.

Algorithm 1 Global search: Our overall search procedure takes as input an AST representation of the initial verification equation, then attempts technique 0a, 3b, 7 and 1 in the pre-processing step. The rest of the algorithm executes the **PBFSearch** algorithm (shown in Algorithm 2) to determine ordering. Then, it applies the post-processing step by attempting technique 0b and 3b if there are loops involved. The search returns the best batch verification algorithm and the order of techniques applied.

```

1: procedure GLOBALSEARCH(eq)
2:   path1  $\leftarrow \{\}$ 
3:   pre_techniques  $\leftarrow \{3b, 7\}$             $\rightarrow$  Pre-processing stage
4:   applied, eq1  $\leftarrow$  applyTechnique(technique = 0a, eq)            $\rightarrow$  Try to consolidate equations
5:   if applied = True then            $\rightarrow$  Technique 0a condition is satisfied
6:     for all x  $\in$  pre_techniques do
7:       applied, new_eq  $\leftarrow$  applyTechnique(technique = x, eq1)
8:       if applied = True then
9:         path1  $\leftarrow$  path1 + [x]
10:        eq1  $\leftarrow$  new_eq
11:      end if
12:    end for
13:  end if
14:  applied, eq1  $\leftarrow$  applyTechnique(technique = 1, eq1)            $\rightarrow$  Combine  $\eta$  instances of equations
15:  assert(applied = True)
16:  path1  $\leftarrow$  path1 + [1]
17:
18:  AllThePaths  $\leftarrow$  PBFSEARCH(eq1, path1, allPaths =  $\emptyset$ , start_technique = 2)
19:  (bestEq, path2)  $\leftarrow$  findMin(AllThePaths)
20:     $\rightarrow$  Finds path with lowest runtime estimate recorded during PBFS
21:
22:  post_techniques  $\leftarrow \{0b, 3b\}$             $\rightarrow$  Post-processing stage
23:  for all x  $\in$  post_techniques do
24:    applied, new_eq  $\leftarrow$  applyTechnique(technique = x, bestEq)
25:    if applied = True then
26:      path2  $\leftarrow$  path2 + [x]
27:      bestEq  $\leftarrow$  new_eq
28:    end if
29:  end for
30:
31:  return bestEq, path2
32: end procedure

```

Algorithm 2 Pruned Breadth-First Search: the PBFS algorithm takes as input an AST of the equation, sequence of applied techniques (called *path*), an empty set for storing all uncovered paths (called *allPaths*), and a start technique for the search. The *path* argument records the techniques being explored in the search execution. The algorithm returns a set of paths dictated by **transitionFunc** which is illustrated in Table 1 and an estimate of the batch verifier runtime that is associated with each path. Our algorithm selects whichever path yields the lowest runtime.

```

1: procedure PBFSEARCH(eq, path, allPaths, technique)
2:   applied, new_eq  $\leftarrow$  applyTechnique(technique, eq)
3:
4:   if applied = True then            $\rightarrow$  Technique condition is satisfied
5:     path  $\leftarrow$  path + [technique]    $\rightarrow$  Append technique to path
6:     checkRes  $\leftarrow$  checkForEdge(8, 3b, path)    $\rightarrow$  check if transition from 8 to 3b exists in path
7:     tech_set  $\leftarrow$  transitionFunc(technique, checkRes)    $\rightarrow$  return pruned set
8:     for all x  $\in$  tech_set do
9:       newAllPaths  $\leftarrow$  PBFSearch(new_eq, path, allPaths, x)
10:      allPaths  $\leftarrow$  allPaths  $\cup$  newAllPaths
11:    end for
12:  else            $\rightarrow$  Reached dead end with this path
13:    if path  $\notin$  allPaths then
14:      allPaths  $\leftarrow$  allPaths  $\cup$  path        $\rightarrow$  Add path to set of all paths
15:      time  $\leftarrow$  estimateRuntime(eq, N, T)    $\rightarrow$  N for batch size & T for group op. costs
16:      recordTime(time, path)        $\rightarrow$  record in a global database
17:    end if
18:  end if
19:  return allPaths
20: end procedure

```

We begin with the original verification equation.

$$e(Y, a) \stackrel{?}{=} e(g, b) \text{ and } e(X, a) \cdot e(X, b)^m \stackrel{?}{=} e(g, c)$$

Step 1: Consolidate the verification equations (tech. 0a), and apply the small exponents test as follows: For each of the $z = 1$ to η signatures, choose random $\delta_1, \delta_2 \in [1, 2^\lambda - 1]$ and compute for each equation:

$$e(g, b)^{\delta_1} \cdot e(Y, a)^{-\delta_1} \stackrel{?}{=} e(X, a)^{\delta_2} \cdot e(X, b)^{m \cdot \delta_2} \cdot e(g, c)^{-\delta_2}$$

Step 2: Combine η signatures (tech. 1), move the exponent(s) inside pairing (tech. 2):

$$\prod_{z=1}^{\eta} e(g, b_z^{\delta_{z,1}}) \cdot e(Y, a_z^{-\delta_{z,1}}) \stackrel{?}{=} \prod_{z=1}^{\eta} e(X, a_z^{\delta_{z,2}}) \cdot e(X, b_z^{m_z \cdot \delta_{z,2}}) \cdot e(g, c_z^{-\delta_{z,2}})$$

Step 3: Merge pairings with common first or second argument (tech. 3b):

$$\prod_{z=1}^{\eta} e(g, b_z^{\delta_{z,1}} \cdot c_z^{\delta_{z,2}}) \cdot e(Y, a_z^{-\delta_{z,1}}) \stackrel{?}{=} \prod_{z=1}^{\eta} e(X, a_z^{\delta_{z,2}}) \cdot e(X, b_z^{m_z \cdot \delta_{z,2}})$$

Step 4: Merge pairings with common first or second argument (tech. 3b):

$$\prod_{z=1}^{\eta} e(g, b_z^{\delta_{z,1}} \cdot c_z^{\delta_{z,2}}) \cdot e(Y, a_z^{-\delta_{z,1}}) \stackrel{?}{=} \prod_{z=1}^{\eta} e(X, a_z^{\delta_{z,2}} \cdot b_z^{m_z \cdot \delta_{z,2}})$$

Step 5: Move products inside pairings to reduce η pairings to 1 (tech. 3a):

$$\prod_{z=1}^{\eta} e(g, b_z^{\delta_{z,1}} \cdot c_z^{\delta_{z,2}}) \cdot e(Y, a_z^{-\delta_{z,1}}) \stackrel{?}{=} e(X, \prod_{z=1}^{\eta} a_z^{\delta_{z,2}} \cdot b_z^{m_z \cdot \delta_{z,2}})$$

Step 6: Distribute products (tech. 5):

$$\prod_{z=1}^{\eta} e(g, b_z^{\delta_{z,1}} \cdot c_z^{\delta_{z,2}}) \cdot \prod_{z=1}^{\eta} e(Y, a_z^{-\delta_{z,1}}) \stackrel{?}{=} e(X, \prod_{z=1}^{\eta} a_z^{\delta_{z,2}} \cdot b_z^{m_z \cdot \delta_{z,2}})$$

Step 7: Move products inside pairings to reduce η pairings to 1 (tech. 3a):

$$e(g, \prod_{z=1}^{\eta} b_z^{\delta_{z,1}} \cdot c_z^{\delta_{z,2}}) \cdot e(Y, \prod_{z=1}^{\eta} a_z^{-\delta_{z,1}}) \stackrel{?}{=} e(X, \prod_{z=1}^{\eta} a_z^{\delta_{z,2}} \cdot b_z^{m_z \cdot \delta_{z,2}})$$

Figure 5: A fragment of the machine-generated security proof of a single-signer batch verifier for the bilinear CL signature scheme [20]. An earlier portion of the proof asserted that a group membership test would be done prior to checking the final equation. Here the value g is a generator of a bilinear group, the values X, Y are part of the public key, a signature is a tuple (a, b, c) and the message signed is m .

3.3 Security and Machine-Aided Analysis

Efficiency Analysis. Efficiency of the batch verifiers is computed in two separate ways. During the PBFS algorithm, the Batcher uses the batch size specified by the user to compute an estimate of the runtime for all batch verifiers. The resulting estimates enable selection of an efficient batch verifier from many candidate verifiers. As indicated in Algorithm 2, the estimates are calculated using a database of

average operation times measured at library initialization. Once the Batchers has selected the most efficient batch equation, it performs another analysis to determine a “crossover point”, i.e., the batch size where batch verification becomes more efficient than individual verification. This analysis is done by counting the number of operations required as a function of the batch size. These operations also include group operations, pairings, hashes, as well as random element generation. It then combines this operation count with the database of average operation times to compute the crossover point.

Security Analysis. We have two points to make regarding the security of AutoBatch. First, we argue that the algorithm used by AutoBatch to produce a batch verification equation *unconditionally* satisfies Definition 2.2. That is, the batch verification equation will hold if and only if each of the individual signatures would have passed the individual verification test (up to a negligible error probability).¹¹

Theorem 3.1 (Security of AutoBatch) *Let an AutoBatch algorithm be generalized as any algorithm that transforms an individual pairing-based signature verification test with perfect correctness into a pairing-based batch verification equation as follows:*

1. *Check the group membership of all input elements, and if no errors, apply Techniques 0a and 1 to the individual verification equation(s) using security parameter λ to obtain a single equation X .*
2. *Apply any of Techniques 2-8 to X to obtain equation X' and set $X := X'$.*
3. *Repeat previous step any number of times to X .*
4. *Check if there are loops in X and the bounds are known, then apply Technique 0b with security parameter λ to X and further attempt Technique 3b if applicable. Then, return X .*

Then all AutoBatch algorithms unconditionally satisfy Definition 2.2, where the probability of accepting an invalid batch is at most $2^{-\lambda'}$, where $\lambda' \leq \lambda + 2$.

Proof. We analyze this proof in three parts. First, after Step 1 (the application of Techniques 0a and 1), there will be one batch equation X (possibly with a loop over it) and it will satisfy the security requirements of Definition 2.2 with error probability $2^{-\lambda}$. These two techniques combine a set of equations (possibly with loops over them) into a single equation (possibly with loops over them) using the small exponents test with security parameter λ . Ferrara et al. [28, Theorem 3.2] prove that this equation will verify if and only if all individual equations from the set verify, except with probability at most $2^{-\lambda}$. By default in AutoBatch, we set $\lambda = 80$.

Second, given a single arbitrary, pairing-based equation X (possibly with a loop over it), we apply one of Techniques 2-8 (in Steps 2 and 3). For each Technique 2-8, we argue that the output equation X' holds if and only if the input equation X holds; that is, the equations are identical up to algebraic manipulations. If this is true, the final batch equation output by AutoBatch satisfies Definition 2.2 with the same error probability as the equation output after Techniques 0a and 1 were applied, completing the theorem.

It remains to argue that for each Technique 2-8, it is indeed the case that the input and output equations are identical, up to algebraic manipulations. Techniques 2, 3, 4, 6 and 8 follow relatively straightforwardly from the bilinearity of the groups. As an example, consider Technique 3b which claims that $e(a, g) \cdot e(b, g) = e(a \cdot b, g)$, for all $a, b \in \mathbb{G}_1$ and $g \in G_2$ where $a \neq 1 \wedge b \neq 1$. Let $b = a^k$ for some $k \in \mathbb{Z}_p$. Then we have $e(a, g) \cdot e(a^k, g)$ as the LHS, which is $e(a, g) \cdot e(a, g)^k$ by the bilinearity, which is $e(a, g)^{k+1}$ by multiplication in $\mathbb{G}_T$. The RHS is similarly $e(a \cdot a^k, g) = e(a^{k+1}, g) = e(a, g)^{k+1}$. Technique 5 requires only associativity in $\mathbb{G}_T$. Technique 7 pre-computes and caches values instead of re-computing them on the fly.

Finally, we come to Step 4 with a single equation X , possibly having a loop over it. If bounds for the loop are known, then this step unrolls the loop using Technique 0b, whereby a loop representing $i = 1$ to t iterations over an equation X_i is replaced logically by the set of those equations $\{X_1, X_2, \dots, X_t\}$. This set of

¹¹The security of the underlying signature scheme depends on a computational assumption, but the batcher unconditionally maintains whatever security is offered by the scheme.

equations is then combined into a single equation using the small exponents test, as described above. Finally, Technique 3b is applied if applicable; as discussed above, this technique is a simple algebraic manipulation of the equation and does not change it.

An error of 2^λ is introduced with each small exponents test in Technique 0a, 1 and then 0b, thus the total batch verification error is at most $3(2^{-\lambda})$. This completes the theorem. $\square$

To offer transparency on how AutoBatch derived any given batch verifier, the Batcher produces both an SDL file and, optionally, a human-readable proof that the resulting batch verifier is as secure as verifying the signatures individually. This proof is a LaTeX file that includes the individual and batch verification equations, with an enumeration of the various steps used to convert the former into the latter. Thus, while *Theorem 3.1 already argues that this proof is valid*, this provides a means for independently verifying the security of any given batching equation. Interestingly, the first proof for the batch verification of the HW signatures [36] was produced automatically by AutoBatch.

We show a fragment of this human-readable proof for the Camenisch-Lysyanskaya (CL) scheme [20] in Figure 5. Full proofs for the Hohenberger-Waters (HW) scheme [36], the Camenisch-Lysyanskaya (CL) scheme [20], and the Verifiable Random Functions (VRF) scheme [37] are given in Appendices B, ??, and C, respectively. In Appendix D, we detail the results of AutoBatch on the Waters09 scheme (derived from the Waters Dual-System IBE of [66]); because this scheme has a negligible correctness error our automated proof techniques do not directly apply, although we conjecture that the resulting scheme is secure up to an additional negligible error rate. In particular, there will be a negligible chance that the batcher will output reject on a set of valid signatures.

The security analysis provided in this section applies to the mathematics only. AutoBatch goes on to convert this mathematical batching equation into code, which could potentially introduce *software* errors. However, our hope is that the deliberate process by which AutoBatch generates code would actually help reduce software errors by systematically including steps, such as the group membership test, which could easily be accidentally omitted by a human implementor.

3.4 Code Generation

The output of the Batcher is a batch verification equation encoded in SDL. This file defines all of the datatypes for the signature, message and public key (or identity and public parameters in the case of an identity-based signature). The Code Generator converts this SDL representation into usable Python or C++ source code that can operate on real batch inputs. The SDL representation consists of the individual *and* batch verification equations including logic for the following components:

1. **Group membership tests.** For each element in the signature (and optionally the public key, if the user requests)¹² the membership to the group is tested using an exponentiation. Section 2.2 discusses the importance and details of this test.
2. **Pre-computation.** Several values often will be re-used within a verification equation. When this happens, the batch verifier can *pre-compute* certain results once, rather than needlessly compute them several times.
3. **Verification method.** For relatively small batch sizes, it may be *more* efficient to bypass the batch verifier and simply verify the signatures using the individual verification function. For this reason, our Code Generator generates this function as well (the output of the Batcher contains both functions), and adds logic to programmatically choose between batch and individual verification when the batch size is below a crossover point automatically determined in the Analysis phase.
4. **Invalid signature detection.** To handle the presence of invalid signatures in a batch, our batch verifier code includes a recursive *divide-and-conquer* strategy to recover from a batching failure (see

¹²In many applications we can assume that the public keys are trusted, thus we can omit group membership testing on these values.

e.g., [28] for a discussion of this). On failure, this verifier divides the signature collection into two halves and recurses by repeating verification on each half until all of the invalid signatures have been identified.

The Code Generator consists of two “back-end” modules, which produce Charm/Python and Charm/C++ representations of the batch verifiers. It would be relatively easy to extend this module to add support for additional languages and settings.

3.5 Code Parsing

While SDL is the primary input language for our batcher, we also support batching from a pre-existing implementation of a signature scheme. To facilitate this, we provide a Code Parsing engine that interprets signature schemes written in a high level language, derives their verification equation and data types, and produces a resulting SDL file. While our techniques should work with various languages (provided that the signature implementation is somewhat constrained), our prototype implementation is based on Charm/Python. This means we can take advantage of a relatively large library of pre-existing Charm implementations. Additionally, in this setting we are assisted by the Python interpreter, which grants programatic access to the Python Abstract Syntax Tree via the `compiler.ast` module.

While Charm implementations are relatively constrained in terms of their structure, a challenging aspect of code parsing is identifying the type of each variable. We stress that this problem is not unique to Python: indeed, many standard libraries (such as the the C-based Stanford Pairing-Based Crypto library [47]) employ abstract data types to represent group elements. Interpreting code written using these languages will also require techniques similar to the ones we use.

Code parsing consists of the following stages. First, we parse the entire signature scheme file to identify the AST node of the signature `verify()` method, and then identify the equality comparisons in this function that are fundamentally responsible for the signature verification process. We next build a map of variable names, types, structure, and operations. For each assignment, we check the properties of that assignment using a further set of heuristics. If we determine that a given assignment is relevant, we extract certain information about it, such as the *type* of the variables. We obtain this information by applying known rules to infer types. For example, we know that certain hash calls indicate an element of $\mathbb{G}_1$, a pairing indicates an element in $\mathbb{G}_T$, random element generation calls typically indicate the type of element being generated, and so on.¹³

To simplify the parsing, we restrict the subset of Python converted to SDL. In particular, we do not support the use of functional constructs in Python such as lambda functions. Our database currently includes signatures for the following types:

1. All pairings and their parameters and types.
2. All hashes and their parameters and types.
3. All Python dictionaries, their key names, their value names, and their types. Charm makes extensive use of this data structure, so this is important.
4. All constant numbers and strings.

4 Implementation & Performance

Subsequent to our initial publication of the conference version of this work [2], we identified a software bug in the group membership function of Charm v0.42 that affected our results. The results in this paper include the corrections to the affected group membership test which reduces the efficiency gains of batch verification in all our test cases. In particular, there are noticeable reductions in performance for CL [20], Waters09 [66] and HW (with different signers) [36]. Although an optional feature, our membership tests include public

¹³We believe that this approach may also be useful in the future for static checking and formal verification of dynamically-typed cryptographic implementations.

Scheme	Type Model Ind-Verify			By Hand		By AutoBatch	
				Batch-Verify	Reference	Batch-Verify	Techniques
1. Boyen-Lynn-Shacham (BLS) (ss)	S	RO	2η	2	[16]	2	1,2,3a
2. Camenisch et al. (CHP) (same period)	S	RO	3η	3	[19]	3	1,2,3a,5,3a
3. Camenisch-Lysyanskaya (CL) (ss)	S	P	5η	5η	none	3	0a,1,2,3b,3b,3a,5,3a
4. Hohenberger-Waters (HW) (ss)	S	P	2η	2η	none	4	1,2,3a,8,6,5,3a
5. Hohenberger-Waters (HW)	S	P	2η	2η	none	4	1,2,3a,5,3a
6. Waters09 (ss)	S	P	9η	9η	none	13	1,2,8,5,3a,6,3b
7. Hess	I	RO	2η	2	[28]	2	1,2,3a
8. Cha-Cheon (ChCh)	I	RO	2η	2	[42]	2	1,2,3a
9. Waters05	I	P	3η	$z + 3$	[19]	$z + 3$	1,2,3a,8,6,5,3a,4,3b
10. Chow-Yiu-Hui (CYH)	IR	RO	2η	2	[28]	2	1,2,3a,2
11. Boyen (same ring)	R	P	$\ell\eta + \ell$	$3\ell + 1$	[28]	$3\ell + 1$	1,2,8,4,3b,8,5,3a
12. Boneh-Boyen-Shacham (BBS)	G	RO	5η	2	[28]	2	1,2,3b,3b,5,3a
13. VRF equations 1,3,4 & 2 (ss)	V	P	$3\eta + 2\ell$	$3\ell + 1$	[37]	$\ell + 3$	0a,3b,1,2,3a,1,2,3a,5,3a,3b,0b,3b
<i>14. ChCh and Hess together</i>	M	RO	2η	4	none	2	0a,1,2,3a,5,3a,3b

Table 2: Digital Signature Schemes used as test cases in AutoBatch. We show a comparison between naive batch verifiers designed by hand or discovered in the literature and ones found by AutoBatch. Scheme names followed by an “ss” were only batched for the same signers; otherwise, different signers were allowed. For types, S stands for regular signature, I stands for identity-based, M stands for a batch that contains a mix of two different types of signatures, R stands for ring, G stands for group and V stands for verifiable random function. For models, RO stands for random oracle and P stands for plain. Let ℓ be either the size of the ring or the number of bits in the VRF input. Let z be a security parameter for the Waters hash [65] and can be set to 5 in practice. To approximate verification performance, we count the total number of pairings needed to process η valid signatures. Unless otherwise noted, the inputs are from different signers. The final column indicates the order of the techniques from Section 3 that AutoBatch applied to obtain the resulting batch verifier. The **rows in bold** are the schemes where AutoBatch discovered new or improved algorithms. Finally, the *italicized row* represents the ability of AutoBatch to construct batch verifiers for different signature types. This is an instance of cross-scheme batching and we compare it to batching naively per signature type.

keys to reflect the worst case performance of batch verification without invalid signatures in the batch. See Figure 6 for the new graphs.

4.1 Experimental Setup

To evaluate the performance of our techniques we implemented them as part of the Charm prototyping framework [1]. Charm is a Python-based cryptographic prototyping framework, and provides native support for bilinear-map based cryptography and other useful primitives, e.g., hashing and serialization. We used a version of Charm that implements all bilinear group operations using the C-based MIRACL library [59].¹⁴ The necessary MIRACL calls are accessed from within our Python code via the C module interface.

To determine the performance of our system in isolation, we first conducted a number of experiments on various components of our code. First, we used the code parsing component to convert several Python signature implementations into our intermediate SDL representation. Next, we applied our batcher to the SDL result in order to obtain an optimized equation for a *batch verifier*. We then applied our code generator to convert this representation into a functioning batch verifier program, which we applied to various test data sets.

Hardware configuration. For consistent results we ran all of our experiments on a single hardware platform: a 2 x 2.66 GHz 6-Core Intel Xeon Macintosh Pro running MacOS version 10.7.3 with 12GB of RAM. We ran all of our tests within a single thread, and thus used resources from only a single core of the Intel processor.

¹⁴The version of Charm we used (0.42) can be found in the Charm github repository at www.charm-crypto.com. It uses MIRACL 5.5.4 for bilinear group operations.

	Approx. Signature Size		MIRACL w/ BN256		RELIC w/ BN256	
	MNT160	BN256	Individual	Batched*	Individual	Batched*
<i>Signatures</i>						
BLS [16] (same signer)	160 bits	256 bits	26.6 ms	2.2 ms	11.9 ms	1.5 ms
CHP [19] (same time period)	160 bits	256 bits	46.1 ms	7.2 ms	24.0 ms	7.8 ms
HW [36] (same signer)	320 bits	512 bits	40.5 ms	4.7 ms	22.4 ms	3.0 ms
HW [36] (diff signer)	320 bits	512 bits	40.5 ms	61.1 ms	22.4 ms	29.2 ms
Waters09 [66, §6.1] (same signer)	6240 bits	6912 bits	153.2 ms	33.1 ms	93.7 ms	44.2 ms
CL [20] (same signer)	480 bits	768 bits	72.0 ms	15.9 ms	34.6 ms	18.0 ms
<i>ID-Based Signatures</i>						
Hess [35]	1120 bits	3328 bits	32.7 ms	22.0 ms	17.1 ms	8.4 ms
ChCh [24]	320 bits	512 bits	27.5 ms	4.6 ms	12.6 ms	2.4 ms
Waters05 [65]	480 bits	768 bits	45.3 ms	11.8 ms	21.5 ms	11.0 ms
<i>Group, Ring and ID-based Ring Signatures</i>						
BBS [13] Group signature	2400 bits	5376 bits	99.9 ms	31.2 ms	63.9 ms	18.7 ms
Boyen [18] Ring signature, 3-member ring	960 bits	1536 bits	64.2 ms	15.0 ms	41.5 ms	9.8 ms
CYH [27] Ring signature, 10-member ring	1760 bits	2816 bits	34.2 ms	22.3 ms	20.7 ms	16.2 ms
<i>VRFs</i>						
HW VRF [Hohenberger-Waters 2010] (same signer, $\ell = 8$)	2240 bits	5120 bits	251.4 ms	36.1 ms	112.5 ms	18.3 ms
<i>Combinations</i>						
ChCh + Hess	1440 bits	3840 bits	55.6 ms	26.2 ms	25.7 ms	10.4 ms

*Verification time *per signature* when batching 100 signatures.

Table 3: Cryptographic overhead and verification time for all of the pairing-based signatures in an alternative implementation of AutoBatch. RELIC is faster on 12 of 14 schemes, but MIRACL is better on CL and Waters09. We speculate that this is because modular exponentiation in $\mathbb{G}_1$ and $\mathbb{G}_2$ is slightly slower in RELIC compared to MIRACL. Since RELIC is an actively developed library, we believe this issue can be addressed in future versions. In the case of HW (with different signers), individual verification outperforms batch verification in both libraries because batch time is dominated by group membership tests.

We instantiated all of our cryptographic implementations using a 160-bit MNT elliptic curve and a 256-bit Barreto-Naehrig (BN) curve provided with MIRACL. Results are shown in Table 3 and Figure 6.

A note on the library. We chose MIRACL because it is mature and well supported. However, some research libraries like RELIC [4] provide alternative pairing implementations that may outperform MIRACL in specific settings. We note that our results will apply to any implementation where there is a substantial difference between group operation and pairing times. In our experiments with RELIC using a provided BN256 curve, we observed a 6-to-1 differential between pairings and operations in $\mathbb{G}_1$. Our main results do hold in this setting, and in fact improve the overall performance in that we can process a higher number of signatures with batch verification. We provide the details of this alternative version of AutoBatch and a complete comparison against the BN256 curve MIRACL implementation in Table 3.

4.2 Signature Schemes Used as Test Cases and Summary of the Results

We ran our experiments using two sets of test cases. The first set was comprised of a variety of existing schemes, including regular, identity-based, ring and group signatures, as well as verifiable random functions. To make AutoBatch as robust as possible, we also tested it on a second set of fabricated pairing-product equations that we designed by hand to trigger many different orderings on the techniques. We summarize AutoBatch’s performance on existing schemes in Table 2.

In eight cases, the batching algorithm output by AutoBatch matched the prior best known result. In the remaining cases, AutoBatch provided a faster algorithm. We now describe these cases in more detail.

We briefly recall the verification equations in VRF [37]. The public key is represented by $\hat{U}, U, g_1, g_2, h$, the signature is represented by $y, \pi = \pi_0\pi_1, \dots, \pi_\ell$, and the message is $x = x_1, \dots, x_\ell$, where ℓ denotes the number of bits in the VRF input. The equations are as follows:

1. $e(\pi_1, g_2) \stackrel{?}{=} e(g_1^{(1-x_1)} \cdot U_1^{x_1}, \hat{U})$
2. for $t = 2$ to ℓ it holds: $e(\pi_t, g_2) \stackrel{?}{=} e(\pi_{t-1}^{(1-x_t)}, g_2) \cdot e(\pi_{t-1}^{x_t}, U_t)$
3. $e(\pi_0, g_2) \stackrel{?}{=} e(\pi_\ell, U_0)$
4. $e(\pi_0, h) \stackrel{?}{=} y$

AutoBatch first realized a batching algorithm for the VRF [37] that takes only two-thirds the time of the one provided in [37] (or $2\ell + 2$ total pairings). Then, after we double-checked this result by hand, we realized that the verification of equation 2 could be further optimized to only $\ell - 1$ pairings by unrolling the loop and combining the individual verification equations checked at each iteration. Moreover, a portion of the unrolled loop with the g_2 term could be combined with the corresponding term in the combined equations 1,3,4 for a total pairing count of only $\ell + 3$ pairings to batch an arbitrary number of VRF proofs for ℓ -bit inputs. We implemented this loop unrolling technique, incorporated it into AutoBatch and automatically applied it to VRF to obtain $\ell + 3$ pairings. The VRF batching algorithm and proof appear in Appendix C.

In test case 14 shown in Table 2 (ChCh [24] and Hess [35] together), we simulated a scenario where a batch contains a mix of two different types of signatures. In this case, the batch consisted of both ChCh [24] signatures and Hess [35] signatures in a randomized order. Instead of sorting the signatures into two groups and batching them individually, AutoBatch automatically looked for the common algebraic structure between the two distinct schemes and applied the batching techniques described in Section 3.1. As a generalized example, if two signature schemes both use the same generator g , where the first signature scheme uses $e(A, g)$ in its verification equation and the second signature scheme uses $e(B, g)$ in its verification equation, then AutoBatch will apply Technique 6 to obtain $e(A \cdot B, g)$ in the combined verification equation (as well as apply the small exponents test). In the case of the ChCh [24] and Hess [35] batch, this cuts the total number of pairings in half. To the best of our knowledge, this is the first documented result for *cross-scheme* signature batch verification.

For the Hohenberger-Waters signatures [36], we assume that each public key includes the precomputed values as suggested in [36, Section 4.2]. For the case of different signers, we assume that the base group elements g, u, v, d, w, z, h are chosen by a trusted third party and shared by all users. The Waters09 scheme is derived from the Waters Dual-System IBE of [66] using the technique described by Naor [14]. Because the decryption algorithm of this IBE scheme has a negligibly small correctness error, the resulting signature scheme also has a negligible correctness error. That is, there is a small chance that a valid signature will be rejected by the verification test. Although this means that our automated proof techniques do not immediately apply, we still wanted to run the program on this complicated test case to see how efficient of a candidate batching scheme it could produce. The details of these batching algorithms appear in Appendices B and D respectively.

4.3 Microbenchmarks

To evaluate the efficiency of AutoBatch, we implemented several pairing-based signature schemes in Charm. We ran AutoBatch to extract an SDL-based intermediate representation of the scheme’s verification equation, an optimized batch verifier for the scheme, and Python and C++ code for implementing the batch verifier. We measured the processing time for each of the above steps. Our timings, averaged over 100 runs, are presented in Table 4.

To obtain our microbenchmarks, we ran AutoBatch on several exemplary pairing-based schemes as listed in Table 2. We then experimented with these schemes at different batch sizes, in order to evaluate their raw performance. The results are presented in Figure 6.

Each graph shows the average per-signature verification time for a batch of η signatures, for η ranging from 1 to 100. We conducted these tests by first generating a collection of η keypairs and random messages,¹⁵

¹⁵We used 100-byte random strings for each message. In the case of the stateful HW signature, we batched only signatures with the same counter value.

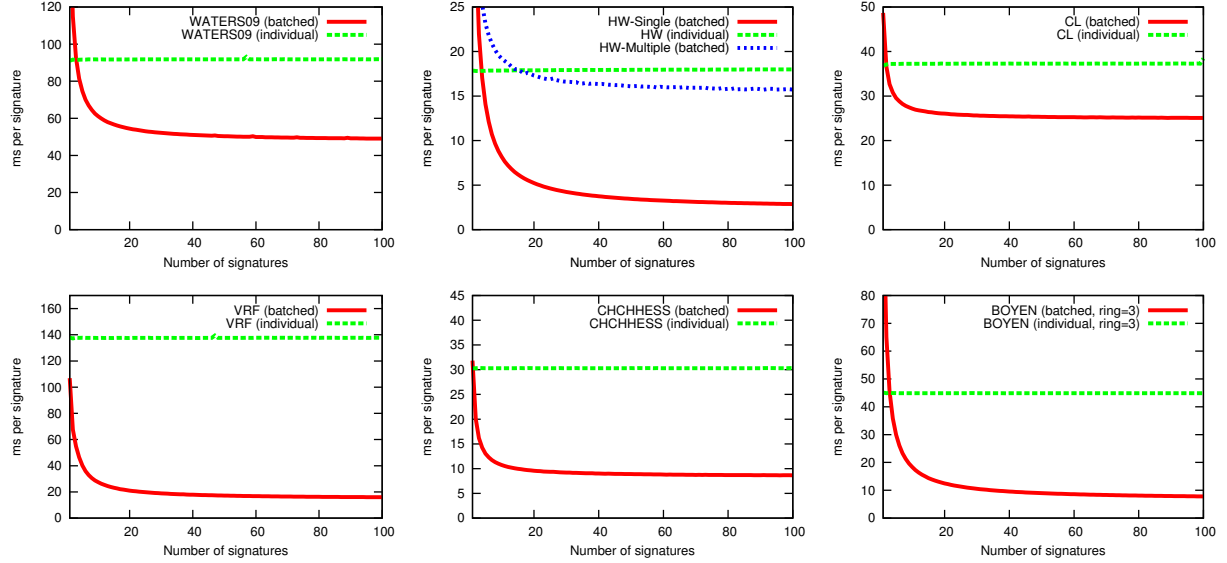


Figure 6: Signature scheme microbenchmarks for Waters09 [66], HW [36] and CL [20] public-key signatures (same signer), the VRF [37] (with block size of 8), combined verification of ChCh+Hess IBS [24, 35], and Boyen ring signature (3 signer ring) [18]. Per-signature times were computed by dividing total batch verification time by the number of signatures verified. All trials were conducted with 10 iterations and were instantiated using a 160-bit MNT elliptic curve. Variation in running time between trials of the same signature size were minimal for each scheme. Note that in one HW case, all signatures are formulated by the same signer (as for certificate generation). All other schemes are without such restrictions. Individual verification times are included for comparison.

then computing a valid signature over each message. We fed each collection to the batch verifier. ID-based signatures were handled in a similar manner, although we substitute random identities in place of keys. For the Boyen ring signature, we generated a group of three signing keys to construct our ring. In each case, we averaged our results over 100 experimental runs and computed verification time per signature by dividing the total batching time by the number of signatures batched.

4.4 Batch Verification in Practice

Prior works considered the implication of *invalid* signatures in a batch, e.g., [28, 42, 50, 51, 69]. Mainly, these works estimated raw signature verification times under various conditions. To evaluate how signature batching might work in real life, we constructed a simulation to determine the resilience of our techniques to various denial of service attacks launched by an adversary.

Basic Model. For this experiment, we simulated a server that verifies incoming signed messages read from a network connection. This might be a reasonable model for a busy server-side TLS endpoint using client authentication or for a car-to-car communications base station.

Our server is designed to process as many signatures as possible, and is limited only by its computational resources.¹⁶ Signatures are drawn off of the “wire” and grouped into batches, with each batch size representing the expected number of signatures that can be verified in one second. Initially this number is simply a guess, which is adjusted upwards or downwards based on the time required to verify each batch.¹⁷ This approach can lead to some transient errors (batches that require significantly more or less than one

¹⁶This models a server that delays, drops or redirects the signatures that it cannot handle (e.g., via load balancing).

¹⁷The adjustment is handled in a relatively naive way: the server simply computes the next batch size by extrapolating based on its time to compute the previous batch.

Process	BLS	CHP	CL	HW-diff	Waters09	Waters05	ChCh/Hess	CYH	Boyen	BBS	VRF
Batcher	103.1	90.1	295.2	126.1	578.9	1859.2	160.1	101.2	545.1	443.5	419.5
Partial-Codegen	124.3	171.7	152.2	242.3	361.6	291.2	162.0	242.8	321.2	315.1	251.2
Full-Codegen	491.7	757.8	785.9	1481.6	3405.8	1507.1	798.6	876.3	1233.5	1998.3	2748.3

Table 4: Time in milliseconds required by the Batcher and Code Generator to process a variety of signature schemes (averaged over 100 test runs). **Batcher** time includes search time for the technique ordering, generating the proof and estimating the crossover point between individual and batch verification. The **Partial-Codegen** time represents the generation of the batch verifier code from a partial SDL description and Charm implementation of the scheme in Python. The **Full-Codegen** time represents the generation of code from a full SDL description only. The running times are a product of the complexity of each scheme as well as the number of unique paths uncovered by our search algorithm. In all cases, the standard deviation in the results were within $\pm 3\%$ of the average.

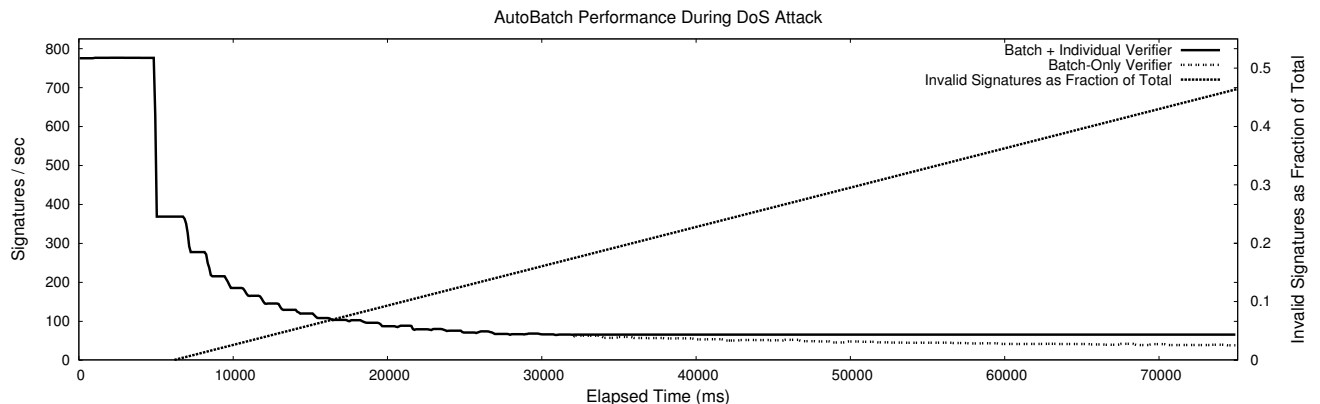


Figure 7: Simulated service denial attacks against a batch verifier (BLS signatures, single signer). The “Invalid Signatures as Fraction of Total” line (right scale) shows the fraction of invalid signatures in the stream. Batcher throughput is measured in signatures per second (left scale). The “Batch-Only Verifier” line depicts a standard batch verifier. The solid line is a batch verifier that automatically switches to *individual* verification when batching becomes suboptimal.

second to evaluate) when the initial guess is wrong, or when conditions change. In normal usage, however, this approach converges on an appropriate batch size within 1-2 seconds.

4.4.1 Basic DoS Attacks

A major concern when using a batch verifier is the possibility of *service denial* or degradation, resulting from the presence of some invalid signatures in the batch. As described in Section 3, each of our batch verifiers incorporates a recursive divide-and-conquer strategy for identifying these invalid signatures, which is borrowed from Law and Matt [42]. This recursion comes at a price; the presence of even a small number of invalid signatures can seriously degrade the performance of a batch verifier.

To measure this, we simulated an adversary who injects invalid signatures into the input stream. Under the assumption that these signatures are well-mixed with the remaining valid signatures,¹⁸ we measured the verifier’s throughput. Our adversary injects no invalid signatures for the first several seconds of the experiment, then gradually ramps up its output until the number of invalid signatures received by the verifier approaches 50%.

¹⁸In practice, this is not a strong assumption, as a server can simply randomize the order of the signatures it receives.

A switch to individual verification. Our experiments indicate that batch verification performance exceeds that of individual verification even in the presence of a relatively large fraction of invalid signatures. However, at a certain point the batch verifier inevitably begins to underperform individual verification.¹⁹ To address this, we implemented a “countermeasure” in our batch verifier to automatically switch to individual verification whenever it detects the presence of a significant fraction of invalid signatures.

Analysis of results. We tested the batch verifier on the single-signer BLS scheme with and without the individual-verification countermeasure. See Figure 7. Throughput is quite sensitive to even small numbers of invalid signatures in the input stream. Yet, when comparing batch verification to *individual* verification throughput, *even under a significant attack* batch verification dramatically outperforms individual verification (up to approximately 15% ratio of invalid signatures). Similarly, the switch to individual verification is a useful countermeasure for attacks that exceed approximately 20% invalid signatures. While these threshold switches do not thwart DoS attacks, they do provide some mitigation of the potential damage.

5 AutoBatch Toolkit

The AutoBatch source code and test cases described in this paper are publicly available in the github repository at <https://github.com/JHUISI/auto-tools>.

6 Conclusion

The batch verification of pairing-based signatures is a great fit for applications where short signatures are a design requirement and yet high verification throughput is required, such as car-to-car communications [23, 60]. This work demonstrates for the first time that the design of these batching algorithms can be efficiently and securely automated.

The next step is to tackle the automated design of more complex functionalities, where it may be infeasible to replicate a theorem like Theorem 3.1 arguing that automated design process unconditionally preserves security. In this case, one might instead focus on having the design tool also output a proof sketch that could be fed into and verified by EasyCrypt [10] or a similar proof checking tool. Indeed, what are the natural settings where the creativity of the design process can be feasibly replaced by an extensive computerized search (perhaps with smart pruning)? Can the “proof sketches” needed for verification by EasyCrypt be generated automatically for these designs? These are exciting questions which could fundamentally change cryptography.

On the implementation of AutoBatch, future work could be more resilient to DoS and related attacks by implementing alternative techniques for recognizing invalid signatures in a batch, e.g., [42, 50, 51, 69]. We are continuously on the lookout for more efficient means of computing in bilinear groups. Future versions of AutoBatch will support MIRACL’s API for computing “multipairings” (efficient products of multiple bilinear pairings). It would be interesting to understand how this and future inclusions may impact performance.

References

- [1] AKINYELE, J. A., GARMAN, C., MIERS, I., PAGANO, M. W., RUSHANAN, M., GREEN, M., AND RUBIN, A. D. Charm: a framework for rapidly prototyping cryptosystems. *Journal of Cryptographic Engineering* 3, 2 (2013), 111–128.
- [2] AKINYELE, J. A., GREEN, M., HOHENBERGER, S., AND PAGANO, M. W. Machine-generated algorithms, proofs and software for the batch verification of digital signature schemes. In *Proceedings of the 2012 ACM Conference on Computer and Communications Security* (New York, NY, USA, 2012), CCS ’12, ACM, pp. 474–487.
- [3] ALMEIDA, J. B., BANGERTER, E., BARBOSA, M., KRENN, S., SADEGHI, A.-R., AND SCHNEIDER, T. A certifying compiler for zero-knowledge proofs of knowledge based on Σ -protocols. In *Proceedings of the 15th European conference on Research in computer security* (2010), ESORICS’10, Springer-Verlag, pp. 151–167.

¹⁹The reason for this is easy to explain: since our batch verifier handles invalid signatures via a divide-and-conquer approach (cutting the signature batch into halves, and recursing on each half), at a certain point the number of “extra” operations exceeds those required for individual verification.

- [4] ARANHA, D. F., AND GOUVÊA, C. P. L. RELIC is an Efficient Library for Cryptography. <http://code.google.com/p/relic-toolkit/>.
- [5] BACELAR ALMEIDA, J., BARBOSA, M., BANGERTER, E., BARTHE, G., KRENN, S., AND ZANELLA BÉGUELIN, S. Full proof cryptography: verifiable compilation of efficient zero-knowledge protocols. In *Proceedings of the 2012 ACM conference on Computer and communications security* (2012), CCS '12, ACM, pp. 488–500.
- [6] BACKES, M., MAFFEI, M., AND UNRUH, D. Zero-knowledge in the applied pi-calculus and automated verification of the direct anonymous attestation protocol. In *Proceedings of the 2008 IEEE Symposium on Security and Privacy* (2008), SP '08, IEEE Computer Society, pp. 202–215.
- [7] BANGERTER, E., BRINER, T., HENECKA, W., KRENN, S., SADEGHI, A.-R., AND SCHNEIDER, T. Automatic generation of sigma-protocols. In *Proceedings of the 6th European conference on Public key infrastructures, services and applications* (2010), EuroPKI'09, Springer-Verlag, pp. 67–82.
- [8] BARAK, B., CANETTI, R., NIELSEN, J. B., AND PASS, R. Universally composable protocols with relaxed set-up assumptions. In *FOCS* (2004), IEEE Computer Society, pp. 186–195.
- [9] BARBOSA, M., MOSS, A., AND PAGE, D. Compiler assisted elliptic curve cryptography. In *Proceedings of the 2007 OTM confederated international conference on On the move to meaningful internet systems: CoopIS, DOA, ODBASE, GADA, and IS - Volume Part II* (2007), OTM'07, Springer-Verlag, pp. 1785–1802.
- [10] BARTHE, G., GRÉGOIRE, B., HERAUD, S., AND BÉGUELIN, S. Z. Computer-aided security proofs for the working cryptographer. In *CRYPTO* (2011), pp. 71–90.
- [11] BELLARE, M., GARAY, J. A., AND RABIN, T. Fast batch verification for modular exponentiation and digital signatures. In *EUROCRYPT '98* (1998), vol. 1403 of LNCS, Springer, pp. 236–250.
- [12] BLAZY, O., FUCHSBAUER, G., IZABACHÈNE, M., JAMBERT, A., SIBERT, H., AND VERGNAUD, D. Batch groth-sahai. In *ACNS '10* (2010), Springer, pp. 218–235.
- [13] BONEH, D., BOYEN, X., AND SHACHAM, H. Short group signatures. In *CRYPTO '04* (2004), vol. 3152 of LNCS, pp. 45–55.
- [14] BONEH, D., AND FRANKLIN, M. K. Identity-based encryption from the Weil pairing. In *CRYPTO* (2001), pp. 213–229.
- [15] BONEH, D., LYNN, B., AND SHACHAM, H. Short signatures from the Weil pairing. In *ASIACRYPT '01* (2001), vol. 2248 of LNCS, pp. 514–532.
- [16] BONEH, D., LYNN, B., AND SHACHAM, H. Short signatures from the Weil pairing. *Journal of Cryptology* 17(4) (2004), 297–319.
- [17] BOYD, C., AND PAVLOVSKI, C. Attacking and repairing batch verification schemes. In *Advances in Cryptology – ASIACRYPT '00* (2000), vol. 1976, pp. 58–71.
- [18] BOYEN, X. Mesh signatures: How to leak a secret with unwitting and unwilling participants. In *EUROCRYPT* (2007), vol. 4515, pp. 210–227.
- [19] CAMENISCH, J., HOHENBERGER, S., AND PEDERSEN, M. Ø. Batch verification of short signatures. In *EUROCRYPT '07* (2007), vol. 4515 of LNCS, Springer, pp. 246–263. Full version at <http://eprint.iacr.org/2007/172>.
- [20] CAMENISCH, J., AND LYSYANSKAYA, A. Signature schemes and anonymous credentials from bilinear maps. In *CRYPTO '04* (2004), vol. 3152 of LNCS, Springer, pp. 56–72.
- [21] CAMENISCH, J., ROHE, M., AND SADEGHI, A. Sokrates - a compiler framework for zero-knowledge protocols. In *Proceedings of the Western European Workshop on Research in Cryptology* (2005), WEWoRC 2005.
- [22] CAO, T., LIN, D., AND XUE, R. Security analysis of some batch verifying signatures from pairings. *International Journal of Network Security* 3, 2 (2006), 138–143.
- [23] CAR 2 CAR. Communication consortium. <http://car-to-car.org>.
- [24] CHA, J. C., AND CHEON, J. H. An identity-based signature from gap Diffie-Hellman groups. In *PKC '03* (2003), vol. 2567 of LNCS, Springer, pp. 18–30.
- [25] CHATTERJEE, S., AND SARKAR, P. HIBE with short public parameters without random oracle. In *ASIACRYPT '06* (2006), vol. 4284 of LNCS, pp. 145–160.
- [26] CHAUM, D., AND VAN HEYST, E. Group signatures. In *EUROCRYPT* (1991), pp. 257–265.
- [27] CHOW, S. S. M., YIU, S.-M., AND HUI, L. C. Efficient identity based ring signature. In *ACNS* (2005), vol. 3531 of LNCS, pp. 499–512.
- [28] FERRARA, A. L., GREEN, M., HOHENBERGER, S., AND PEDERSEN, M. Ø. Practical short signature batch verification. In *CT-RSA* (2009), vol. 5473 of LNCS, pp. 309–324.
- [29] FIAT, A. Batch RSA. In *Advances in Cryptology – CRYPTO '89* (1989), vol. 435, pp. 175–185.
- [30] FOURNET, C., KOHLWEISS, M., DANEZIS, G., AND LUO, Z. ZQL: A compiler for privacy-preserving data processing. In *Proceedings of the 13th conference on USENIX Security Symposium - Volume 13* (2004), SSYM'04, USENIX Association, pp. 20–20.

- [31] GOLDWASSER, S., MICALI, S., AND RIVEST, R. L. A digital signature scheme secure against adaptive chosen-message attacks. *SIAM J. Computing* 17(2) (1988).
- [32] HARN, L. Batch verifying multiple DSA digital signatures. *Electronics Letters* 34(9) (1998), 870–871.
- [33] HARN, L. Batch verifying multiple RSA digital signatures. *Electronics Letters* 34(12) (1998), 1219–1220.
- [34] HENECKA, W., K ÖGL, S., SADEGHI, A.-R., SCHNEIDER, T., AND WEHRENBURG, I. TASTY: tool for automating secure two-party computations. In *Proceedings of the 17th ACM conference on Computer and communications security* (2010), CCS '10, ACM, pp. 451–462.
- [35] HESS, F. Efficient identity based signature schemes based on pairings. In *Selected Areas in Cryptography* (2002), vol. 2595 of LNCS, Springer, pp. 310–324.
- [36] HOHENBERGER, S., AND WATERS, B. Realizing hash-and-sign signatures under standard assumptions. In *EUROCRYPT* (2009), pp. 333–350.
- [37] HOHENBERGER, S., AND WATERS, B. Constructing verifiable random functions with large input spaces. In *EUROCRYPT* (2010), pp. 656–672.
- [38] HWANG, M.-S., LEE, C.-C., AND TANG, Y.-L. Two simple batch verifying multiple digital signatures. In *3rd Information and Communications Security (ICICS)* (2001), pp. 233–237.
- [39] HWANG, M.-S., LIN, I.-C., AND HWANG, K.-F. Cryptanalysis of the batch verifying multiple RSA digital signatures. *Informatica, Lithuanian Academy of Sciences* 11, 1 (2000), 15–19.
- [40] KIYOMOTO, S., OTA, H., AND TANAKA, T. A security protocol compiler generating C source codes. In *Proceedings of the 2008 International Conference on Information Security and Assurance (isa 2008)* (2008), ISA '08, IEEE Computer Society, pp. 20–25.
- [41] LAIH, C.-S., AND YEN, S.-M. Improved digital signature suitable for batch verification. *IEEE Transactions on Computers* 44, 7 (1995), 957–959.
- [42] LAW, L., AND MATT, B. J. Finding invalid signatures in pairing-based batches. In *Cryptography and Coding* (2007), vol. 4887 of LNCS, pp. 34–53.
- [43] LEE, S., CHO, S., CHOI, J., AND CHO, Y. Efficient identification of bad signatures in RSA-type batch signature. *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences E89-A*, 1 (2006), 74–80.
- [44] LIM, C., AND LEE, P. Security of interactive DSA batch verification. In *Electronics Letters* (1994), vol. 30(19), pp. 1592–1593.
- [45] LOWE, G. Casper: a compiler for the analysis of security protocols. *J. Comput. Secur.* 6, 1-2 (Jan. 1998), 53–84.
- [46] LUCKS, S., SCHMOIGL, N., AND TATLI, E. I. Issues on designing a cryptographic compiler. In *WEWoRC* (2005), pp. 109–122.
- [47] LYNN, B. The Stanford Pairing Based Crypto Library. Available from <http://crypto.stanford.edu/abc>.
- [48] MACKENZIE, P., OPREA, A., AND REITER, M. K. Automatic generation of two-party computations. In *Proceedings of the 10th ACM conference on Computer and communications security* (2003), CCS '03, ACM, pp. 210–219.
- [49] MALKHI, D., NISAN, N., PINKAS, B., AND SELLA, Y. Fairplay – a secure two-party computation system. In *Proceedings of the 13th conference on USENIX Security Symposium - Volume 13* (2004), SSYM'04, USENIX Association, pp. 20–20.
- [50] MATT, B. J. Identification of multiple invalid signatures in pairing-based batched signatures. In *Public Key Cryptography* (2009), pp. 337–356.
- [51] MATT, B. J. Identification of multiple invalid pairing-based signatures in constrained batches. In *Pairing* (2010), pp. 78–95.
- [52] MEIKLEJOHN, S., ERWAY, C. C., KÜPÇÜ, A., HINKLE, T., AND LYSYANSKAYA, A. ZKPD: a language-based system for efficient zero-knowledge proofs and electronic cash. In *Proceedings of the 19th USENIX conference on Security* (2010), USENIX Security'10, USENIX Association, pp. 13–13.
- [53] MICALI, S., RABIN, M. O., AND VADHAN, S. P. Verifiable random functions. In *FOCS* (1999), pp. 120–130.
- [54] NACCACHE, D. Secure and *practical* identity-based encryption, 2005. Cryptology ePrint Archive: Report 2005/369.
- [55] NACCACHE, D., M'RAÏHI, D., VAUDENAY, S., AND RAPHAËLI, D. Can DSA be improved? complexity trade-offs with the digital signature standard. In *Advances in Cryptology – EUROCRYPT '94* (1994), vol. 950, pp. 77–85.
- [56] PEREZ, L. J. D., AND SCOTT, M. Designing a code generator for pairing based cryptographic functions. In *Proceedings of the 4th international conference on Pairing-based cryptography* (2010), Pairing'10, Springer-Verlag, pp. 207–224.
- [57] POZZA, D., SISTO, R., AND DURANTE, L. Spi2Java: Automatic cryptographic protocol java code generation from spi calculus. In *Proceedings of the 18th International Conference on Advanced Information Networking and Applications - Volume 2* (2004), AINA '04, IEEE Computer Society, pp. 400–.
- [58] RIVEST, R. L., SHAMIR, A., AND TAUMAN, Y. How to leak a secret. In *ASIACRYPT* (2001), pp. 552–565.
- [59] SCOTT, M. Multiprecision Integer and Rational Arithmetic C/C++ Library (MIRACL), Oct. 2007. Published by Shamus Software Ltd., <http://www.shamus.ie/>.

- [60] SEVECOM. Security on the road. <http://www.sevecom.org>.
- [61] SHACHAM, H., AND BONEH, D. Improving SSL handshake performance via batching. In *Cryptographer's Track at RSA Conference '01* (2001), vol. 2020, pp. 28–43.
- [62] SHAMIR, A. Identity-based cryptosystems and signature schemes. In *CRYPTO* (1984), pp. 47–53.
- [63] SONG, D. X., PERRIG, A., AND PHAN, D. AGVI - automatic generation, verification, and implementation of security protocols. In *Proceedings of the 13th International Conference on Computer Aided Verification* (2001), CAV '01, Springer-Verlag, pp. 241–245.
- [64] STANEK, M. Attacking LCCC batch verification of RSA signatures, 2006. Cryptology ePrint Archive: Report 2006/111.
- [65] WATERS, B. Efficient identity-based encryption without random oracles. In *EUROCRYPT '05* (2005), vol. 3494 of LNCS, Springer, pp. 320–329.
- [66] WATERS, B. Dual System Encryption: Realizing Fully Secure IBE and HIBE under Simple Assumptions. In *CRYPTO* (2009), pp. 619–636.
- [67] WATERS, B. Dual system encryption: Realizing fully secure ibe and hibe under simple assumptions. Cryptology ePrint Archive, Report 2009/385, 2009. <http://eprint.iacr.org/>.
- [68] YOON, H., CHEON, J. H., AND KIM, Y. Batch verifications with ID-based signatures. In *ICISC* (2004), Lecture Notes in Computer Science, pp. 233–248.
- [69] ZAVERUCHA, G. M., AND STINSON, D. R. Group testing and batch verification. In *Proceedings of the 4th international conference on Information theoretic security* (2010), ICITS'09, Springer-Verlag, pp. 140–157.
- [70] ZHANG, F., AND KIM, K. Efficient ID-based blind signature and proxy signature from bilinear pairings. In *8th Information Security and Privacy, Australasian Conference (ACISP)* (2003), vol. 2727, pp. 312–323.
- [71] ZHANG, F., SAFAVI-NAINI, R., AND SUSILO, W. Efficient verifiably encrypted signature and partially blind signature from bilinear pairings. In *Progress in Cryptology – INDOCRYPT '03* (2003), vol. 2904, pp. 191–204.

A Machine-Generated Batch Verification Equations

In Figure 8, we provide the final batch verification equations output by AutoBatch for each of the signature schemes tested.

B A machine-generated proof for HW

The following proof was automatically generated by the Batcher while processing the HW signature scheme [36]. This execution allows signatures on different signing keys.

B.1 Definitions

This document contains a proof that HW.BatchVerify is a valid batch verifier for the signature scheme HW. Let U, V, D, g, w, z, h be values drawn from the key and/or parameters, and $M, \sigma_1, \sigma_2, r, i$ represent a message (or message hash) and signature. The individual verification equation HW.Verify is:

$$e(\sigma_1, g) \stackrel{?}{=} U^M \cdot V^r \cdot D \cdot e(\sigma_2, w^{\lceil \lg(i) \rceil} \cdot z^i \cdot h)$$

Let η be the number of signatures in a batch, and $\delta_1, \dots, \delta_\eta \in [1, 2^\lambda - 1]$ be a set of random exponents chosen by the verifier. The batch verification equation HW.BatchVerify is:

$$e\left(\prod_{z=1}^{\eta} \sigma_{z,1}^{\delta_z}, g\right) \stackrel{?}{=} \prod_{z=1}^{\eta} U_z^{M_z \cdot \delta_z} \cdot \prod_{z=1}^{\eta} V_z^{r_z \cdot \delta_z} \cdot \prod_{z=1}^{\eta} D_z^{\delta_z} \cdot e\left(\prod_{z=1}^{\eta} \sigma_{z,2}^{\delta_z \cdot \lceil \lg(i_z) \rceil}, w\right) \cdot e\left(\prod_{z=1}^{\eta} \sigma_{z,2}^{\delta_z \cdot i_z}, z\right) \cdot e\left(\prod_{z=1}^{\eta} \sigma_{z,2}^{\delta_z}, h\right)$$

We will now formally define a batch verifier and demonstrate that HW.BatchVerify is a secure batch verifier for the HW signature scheme.

Definition B.1 (Pairing-based Batch Verifier) Let $\text{BSetup}(1^\tau) \rightarrow (q, g, \mathbb{G}, \mathbb{G}_T, e)$. For each $j \in [1, \eta]$, where $\eta \in \text{poly}(\tau)$, let $X^{(j)}$ be a generic pairing-based claim and let Verify be a pairing based verifier. We define pairing-based batch verifier for Verify a probabilistic $\text{poly}(\tau)$ -time algorithm which outputs accept if $X^{(j)}$ holds for all $j \in [1, \eta]$ whereas it outputs reject if $X^{(j)}$ does not hold for any $j \in [1, \eta]$ except with negligible probability.

Theorem B.2 HW.BatchVerify is a batch verifier for the HW signature scheme.

Scheme	Batch Verification Equation output by AutoBatch
<i>Signatures</i>	
BLS [16] (same signer)	$e(\prod_{z=1}^{\eta} h_z^{\delta_z}, pk) \stackrel{?}{=} e(\prod_{z=1}^{\eta} sig_z^{\delta_z}, g)$
CHP [19] (same time period)	$e(\prod_{z=1}^{\eta} sig_z^{\delta_z}, g) \stackrel{?}{=} e(a, \prod_{z=1}^{\eta} pk_z^{\delta_z}) \cdot e(h, \prod_{z=1}^{\eta} pk_z^{b_z \cdot \delta_z})$
HW [36] (same signer)	$e(\prod_{z=1}^{\eta} \sigma_{1,z}^{\delta_z}, g) \stackrel{?}{=} U^{\sum_{z=1}^{\eta} M_z \cdot \delta_z} \cdot V^{\sum_{z=1}^{\eta} r_z \cdot \delta_z} \cdot D^{\sum_{z=1}^{\eta} \delta_z}$ $\cdot e(\prod_{z=1}^{\eta} \sigma_{2,z}^{\lg(i_z) \cdot \delta_z}, w) \cdot e(\prod_{z=1}^{\eta} \sigma_{2,z}^{i_z \cdot \delta_z}, z) \cdot e(\prod_{z=1}^{\eta} \sigma_{2,z}^{\delta_z}, h)$
HW [36] (different signers)	$e(\prod_{z=1}^{\eta} \sigma_{z,1}^{\delta_z}, g) \stackrel{?}{=} \prod_{z=1}^{\eta} U_z^{M_z \cdot \delta_z} \cdot \prod_{z=1}^{\eta} V_z^{r_z \cdot \delta_z}$ $\cdot \prod_{z=1}^{\eta} D_z^{\delta_z} \cdot e(\prod_{z=1}^{\eta} \sigma_{z,2}^{\delta_z \cdot \lceil \lg(i) \rceil_z}, w) \cdot e(\prod_{z=1}^{\eta} \sigma_{z,2}^{\delta_z \cdot i_z}, z) \cdot e(\prod_{z=1}^{\eta} \sigma_{z,2}^{\delta_z}, h)$
Waters09 [66] (same signer)	$e(g_1^b, \prod_{z=1}^{\eta} \sigma_{z,1}^{s_{z,1} \cdot \delta_z}) \cdot e(g_1^{b \cdot a_1}, \prod_{z=1}^{\eta} \sigma_{z,2}^{s_{z,1} \cdot \delta_z})$ $\cdot e(g_1^{a_1}, \prod_{z=1}^{\eta} \sigma_{z,3}^{s_{z,1} \cdot \delta_z}) \cdot e(g_1^{b \cdot a_2}, \prod_{z=1}^{\eta} \sigma_{z,4}^{s_{z,2} \cdot \delta_z})$ $\cdot e(g_1^{a_2}, \prod_{z=1}^{\eta} \sigma_{z,5}^{s_{z,2} \cdot \delta_z}) \stackrel{?}{=} e(\prod_{z=1}^{\eta} \sigma_{z,6}^{\delta_z \cdot s_{z,1}}, \tau_1)$ $\cdot e(\prod_{z=1}^{\eta} \sigma_{z,6}^{\delta_z \cdot s_{z,2}}, \tau_2) \cdot e(\prod_{z=1}^{\eta} \sigma_{z,7}^{\delta_z \cdot s_{z,1}}, \tau_1^b)$ $\cdot e(\prod_{z=1}^{\eta} \sigma_{z,7}^{\delta_z \cdot s_{z,2}}, \tau_2^b) \cdot e(\prod_{z=1}^{\eta} \sigma_{z,7}^{(\delta_z \cdot -t_z + \theta_z \cdot \delta_z \cdot tag_{z,c} \cdot t_z)}, w)$ $\cdot e(\prod_{z=1}^{\eta} \sigma_{z,7}^{\theta_z \cdot \delta_z \cdot M_z \cdot t_z}, u) \cdot e(\prod_{z=1}^{\eta} \sigma_{z,7}^{\theta_z \cdot \delta_z \cdot t_z}, h)$ $\cdot e(g_1, \prod_{z=1}^{\eta} \sigma_{z,K}^{-t_z \cdot \theta_z \cdot \delta_z}) \cdot A^{\sum_{z=1}^{\eta} s_{z,2} \cdot \delta_z}$
CL [20] (same signer)	$e(g, \prod_{z=1}^{\eta} b_z^{\delta_{z,1}} \cdot c_z^{\delta_{z,2}}) \cdot e(Y, \prod_{z=1}^{\eta} a_z^{-\delta_{z,1}}) \stackrel{?}{=} e(X, \prod_{z=1}^{\eta} a_z^{\delta_{z,2}} \cdot b_z^{m_z \cdot \delta_{z,2}})$
<i>ID-based Signatures</i>	
Hess [35]	$e(\prod_{z=1}^{\eta} S_{2,z}^{\delta_z}, g_2) \stackrel{?}{=} e(\prod_{z=1}^{\eta} pk_z^{a_z \cdot \delta_z}, P_{pub}) \cdot \prod_{z=1}^{\eta} S_{1,z}^{\delta_z}$
ChCh [24]	$e(\prod_{z=1}^{\eta} S_{2,z}^{\delta_z}, g_2) \stackrel{?}{=} e(\prod_{z=1}^{\eta} (S_{1,z} \cdot pk_z^{a_z})^{\delta_z}, P_{pub})$
Waters05 [65]	$e(\prod_{z=1}^{\eta} S_{1,z}^{\delta_z}, g_2) \cdot e(\prod_{z=1}^{\eta} S_{2,z}^{\delta_z}, u_1^t) \cdot \prod_{i=1}^l e(\prod_{z=1}^{\eta} S_{2,z}^{\delta_z \cdot k_{i,z}} \cdot S_{3,z}^{\delta_z \cdot m_{i,z}}, u_i)$ $\cdot e(\prod_{z=1}^{\eta} S_{3,z}^{\delta_z}, u_2^t) \stackrel{?}{=} e(g_1, g_2)^{\sum_{z=1}^{\eta} \delta_z}$
<i>Group, Ring, and ID-based Ring Signatures</i>	
BBS [13]	$e(\prod_{z=1}^{\eta} T_{z,3}^{s_{z,x} \cdot \delta_z} \cdot h^{(-s_{z,\gamma_1} - s_{z,\gamma_2}) \cdot \delta_z} \cdot g_1^{-c_z \cdot \delta_z}, g_2)$ $\cdot e(h^{\sum_{z=1}^{\eta} (-s_{z,\alpha} - s_{z,\beta}) \cdot \delta_z} \cdot \prod_{z=1}^{\eta} T_{z,3}^{c_z \cdot \delta_z}, w) \stackrel{?}{=} \prod_{z=1}^{\eta} R_{z,3}^{\delta_z}$
Boyen [18] (same ring)	$\prod_{y=1}^l e(\prod_{z=1}^{\eta} S_{y,z}^{\delta_z}, \hat{A}_y) \cdot e(\prod_{z=1}^{\eta} S_{y,z}^{m_{y,z} \cdot \delta_z}, \hat{B}_y) \cdot e(\prod_{z=1}^{\eta} S_{y,z}^{t_{y,z} \cdot \delta_z}, \hat{C}_y) \stackrel{?}{=} \prod_{z=1}^{\eta} D^{\delta_z}$
CYH [27]	$e(\prod_{z=1}^{\eta} \prod_{y=1}^l u_{y,z} \cdot pk_{y,z}^{h_{y,z} \cdot \delta_z}, P) \stackrel{?}{=} e(\prod_{z=1}^{\eta} S_z^{\delta_z}, g)$
<i>VRFs</i>	
HW VRF [37] (same signer)	$e(\prod_{z=1}^{\eta} g_1^{(1-x_1) \cdot \delta_{z,2}} \cdot U_1^{x_1 \cdot \delta_{z,2}}, \hat{U}) \cdot e(\prod_{z=1}^{\eta} \pi_{z,1}^{-\delta_{z,2}} \cdot \pi_{z,2}^{\delta_{z,3}} \cdot \pi_{z,1}^{(1-x_{z,2}) \cdot -\delta_{z,3}}$ $\cdot \pi_{z,3}^{-\delta_{z,4}} \cdot \pi_{z,2}^{(1-x_{z,3}) \cdot -\delta_{z,4} - 1} \cdot \pi_{z,4}^{-\delta_{z,5}} \cdot \pi_{z,3}^{(1-x_{z,4}) \cdot -\delta_{z,5} - 1}$ $\cdot \pi_{z,5}^{-\delta_{z,6}} \cdot \pi_{z,4}^{(1-x_{z,5}) \cdot -\delta_{z,6} - 1} \cdot \pi_{z,6}^{-\delta_{z,7}} \cdot \pi_{z,5}^{(1-x_{z,6}) \cdot -\delta_{z,7} - 1} \cdot \pi_{z,7}^{-\delta_{z,8}} \cdot \pi_{z,6}^{(1-x_{z,7}) \cdot -\delta_{z,8} - 1}$ $\cdot \pi_{z,8}^{-\delta_{z,9}} \cdot \pi_{z,7}^{(1-x_{z,8}) \cdot -\delta_{z,9} - 1}, g_2) \stackrel{?}{=}$ $e(\prod_{z=1}^{\eta} \pi_{z,1}^{\delta_{z,1}}, U_0) \cdot \prod_{z=1}^{\eta} y_z^{\delta_{z,1}} \cdot e(\prod_{z=1}^{\eta} \pi_{z,0}^{-\delta_{z,1}}, g_2 \cdot h)$ $\cdot e(\prod_{z=1}^{\eta} \pi_{z,1}^{x_{z,2} \cdot \delta_{z,3}}, U_2) \cdot e(\prod_{z=1}^{\eta} \pi_{z,2}^{x_{z,3} \cdot \delta_{z,4} - 1}, U_3) \cdot e(\prod_{z=1}^{\eta} \pi_{z,3}^{x_{z,4} \cdot \delta_{z,5} - 1}, U_4)$ $\cdot e(\prod_{z=1}^{\eta} \pi_{z,4}^{x_{z,5} \cdot \delta_{z,6} - 1}, U_5) \cdot e(\prod_{z=1}^{\eta} \pi_{z,5}^{x_{z,6} \cdot \delta_{z,7} - 1}, U_6) \cdot e(\prod_{z=1}^{\eta} \pi_{z,6}^{x_{z,7} \cdot \delta_{z,8} - 1}, U_7)$ $\cdot e(\prod_{z=1}^{\eta} \pi_{z,7}^{x_{z,8} \cdot \delta_{z,9} - 1}, U_8)$ for block size of 8
<i>Combinations</i>	
ChCh + Hess	$e(\prod_{z=1}^{\eta} pk_z^{ah_z \cdot \delta_{z,1}} \cdot Sc_{z,1}^{-\delta_{z,2}} \cdot pk_z^{ac_z \cdot -\delta_{z,2}}, P_{pub}) \cdot \prod_{z=1}^{\eta} Sh_{z,1}^{\delta_{z,1}}$ $e(\prod_{z=1}^{\eta} Sh_{z,2}^{-\delta_{z,1}} \cdot Sc_{z,2}^{\delta_{z,2}}, g_2) \stackrel{?}{=} 1$

Figure 8: These are the final batch verification equations output by AutoBatch. Due to space, we do not include the full schemes or further describe the elements of the signature or our shorthand for them, such as setting $h = H(M)$ in BLS. However, a reader could retrace our steps by applying the techniques in Section 3 to the original verification equation in the order specified in Figure 2. “Combined signatures” refers to the combined batching of multiple signature verification equations that share algebraic structure.

B.2 Proof

Proof. Via a series of steps, we will show that if HW is a secure signature scheme, then BatchVerify is a secure batch verifier. Recall our batch verification software will perform a group membership test to ensure that each group element of the signature is a member of the proper subgroup, so here we will assume this fact. We begin with the original verification equation.

$$e(\sigma_1, g) \stackrel{?}{=} U^M \cdot V^r \cdot D \cdot e(\sigma_2, w^{\lceil \lg(i) \rceil} \cdot z^i \cdot h) \quad (1)$$

Step 1: Combine η signatures (technique 1):

$$\prod_{z=1}^{\eta} e(\sigma_{z,1}, g) \stackrel{?}{=} \prod_{z=1}^{\eta} U_z^{M_z} \cdot V_z^{r_z} \cdot D_z \cdot e(\sigma_{z,2}, w^{\lceil \lg(i_z) \rceil} \cdot z^{i_z} \cdot h) \quad (2)$$

Step 2: Apply the small exponents test, using exponents $\delta_1, \dots, \delta_\eta \in [1, 2^\lambda - 1]$:

$$\prod_{z=1}^{\eta} e(\sigma_{z,1}, g)^{\delta_z} \stackrel{?}{=} \prod_{z=1}^{\eta} U_z^{M_z \cdot \delta_z} \cdot \prod_{z=1}^{\eta} V_z^{r_z \cdot \delta_z} \cdot \prod_{z=1}^{\eta} D_z^{\delta_z} \cdot \prod_{z=1}^{\eta} e(\sigma_{z,2}, w^{\lceil \lg(i_z) \rceil} \cdot z^{i_z} \cdot h)^{\delta_z} \quad (3)$$

Step 3: Move exponent(s) inside the pairing (technique 2):

$$\prod_{z=1}^{\eta} e(\sigma_{z,1}^{\delta_z}, g) \stackrel{?}{=} \prod_{z=1}^{\eta} U_z^{M_z \cdot \delta_z} \cdot \prod_{z=1}^{\eta} V_z^{r_z \cdot \delta_z} \cdot \prod_{z=1}^{\eta} D_z^{\delta_z} \cdot \prod_{z=1}^{\eta} e(\sigma_{z,2}^{\delta_z}, w^{\lceil \lg(i_z) \rceil} \cdot z^{i_z} \cdot h) \quad (4)$$

Step 4: Move products inside pairings to reduce η pairings to 1 (technique 3a):

$$e(\prod_{z=1}^{\eta} \sigma_{z,1}^{\delta_z}, g) \stackrel{?}{=} \prod_{z=1}^{\eta} U_z^{M_z \cdot \delta_z} \cdot \prod_{z=1}^{\eta} V_z^{r_z \cdot \delta_z} \cdot \prod_{z=1}^{\eta} D_z^{\delta_z} \cdot \prod_{z=1}^{\eta} e(\sigma_{z,2}^{\delta_z}, w^{\lceil \lg(i_z) \rceil}) \cdot e(\sigma_{z,2}^{\delta_z}, z^{i_z}) \cdot e(\sigma_{z,2}^{\delta_z}, h) \quad (5)$$

Step 5: Distribute products (technique 5):

$$e(\prod_{z=1}^{\eta} \sigma_{z,1}^{\delta_z}, g) \stackrel{?}{=} \prod_{z=1}^{\eta} U_z^{M_z \cdot \delta_z} \cdot \prod_{z=1}^{\eta} V_z^{r_z \cdot \delta_z} \cdot \prod_{z=1}^{\eta} D_z^{\delta_z} \cdot \prod_{z=1}^{\eta} e(\sigma_{z,2}^{\delta_z}, w^{\lceil \lg(i_z) \rceil}) \cdot \prod_{z=1}^{\eta} e(\sigma_{z,2}^{\delta_z}, z^{i_z}) \cdot \prod_{z=1}^{\eta} e(\sigma_{z,2}^{\delta_z}, h) \quad (6)$$

Step 6: Move products inside pairings to reduce η pairings to 1 (technique 3a):

$$e(\prod_{z=1}^{\eta} \sigma_{z,1}^{\delta_z}, g) \stackrel{?}{=} \prod_{z=1}^{\eta} U_z^{M_z \cdot \delta_z} \cdot \prod_{z=1}^{\eta} V_z^{r_z \cdot \delta_z} \cdot \prod_{z=1}^{\eta} D_z^{\delta_z} \cdot e(\prod_{z=1}^{\eta} \sigma_{z,2}^{\delta_z \cdot \lceil \lg(i_z) \rceil}, w) \cdot e(\prod_{z=1}^{\eta} \sigma_{z,2}^{\delta_z \cdot i_z}, z) \cdot e(\prod_{z=1}^{\eta} \sigma_{z,2}^{\delta_z}, h) \quad (7)$$

Steps 1 and 2 form the Combination Step in [28], which was proven to result in a secure batch verifier in [28, Theorem 3.2]. We observe that the remaining steps are merely reorganizing terms within the same equation. Hence, the final verification equation (7) is also batch verifier for HW. $\square$

C A Machine-Generated Proof for VRF

The following proof was automatically generated by the Batcher while processing the VRF signature scheme [37]. This execution was restricted to signatures on a single signing key.

C.1 Definitions

This document contains a proof that `VRF.BatchVerify` is a valid batch verifier for the signature scheme VRF. Let $\hat{U}, U, g_1, g_2, h$ be values drawn from the key and/or parameters, and x, π, y represent a message (or message hash) and signature. The ℓ parameter represents the ℓ -bit input size of VRF and varies in practice. We have shown an example of $\ell = 8$ to simplify the proof. The individual verification equation `VRF.Verify` is:

$$e(\pi_1, g_2) \stackrel{?}{=} e(g_1^{(1-x_1)} \cdot U_1^{x_1}, \hat{U}) \text{ and } e(\pi_0, g_2) \stackrel{?}{=} e(\pi_l, U_0) \text{ and } e(\pi_0, h) \stackrel{?}{=} y \text{ and}$$

$$\text{for } t = 2 \text{ to } \ell \text{ it holds: } e(\pi_t, g_2) \stackrel{?}{=} e(\pi_{t-1}^{(1-x_t)}, g_2) \cdot e(\pi_{t-1}^{x_t}, U_t)$$

Let η be the number of signatures in a batch, and $\delta_{1,i}, \dots, \delta_{\eta,i} \in [1, 2^\lambda - 1]$ be a set of random exponents chosen by the verifier. Since the input size of $\ell = 8$, then $i = 9$. The batch verification equation for VRF is:

`VRFBatchVerify`:

$$\begin{aligned} & e\left(\prod_{z=1}^{\eta} g_1^{(1-x_1) \cdot \delta_{z,2}} \cdot U_1^{x_1 \cdot \delta_{z,2}}, \hat{U}\right) \cdot e\left(\prod_{z=1}^{\eta} \pi_{z,1}^{-\delta_{z,2}} \cdot \pi_{z,2}^{\delta_{z,3}} \cdot \pi_{z,1}^{(1-x_{z,2}) \cdot -\delta_{z,3}} \cdot \pi_{z,3}^{-\delta_{z,4}} \cdot \pi_{z,2}^{(1-x_{z,3}) \cdot -\delta_{z,4} \cdot -1} \cdot \pi_{z,4}^{-\delta_{z,5}} \cdot \pi_{z,3}^{(1-x_{z,4}) \cdot -\delta_{z,5} \cdot -1} \right. \\ & \cdot \pi_{z,5}^{-\delta_{z,6}} \cdot \pi_{z,4}^{(1-x_{z,5}) \cdot -\delta_{z,6} \cdot -1} \cdot \pi_{z,6}^{-\delta_{z,7}} \cdot \pi_{z,5}^{(1-x_{z,6}) \cdot -\delta_{z,7} \cdot -1} \cdot \pi_{z,7}^{-\delta_{z,8}} \cdot \pi_{z,6}^{(1-x_{z,7}) \cdot -\delta_{z,8} \cdot -1} \cdot \pi_{z,8}^{-\delta_{z,9}} \cdot \pi_{z,7}^{(1-x_{z,8}) \cdot -\delta_{z,9} \cdot -1} \left. , g_2\right) \stackrel{?}{=} \\ & e\left(\prod_{z=1}^{\eta} \pi_{z,l}^{\delta_{z,1}}, U_0\right) \cdot \prod_{z=1}^{\eta} y_z^{\delta_{z,1}} \cdot e\left(\prod_{z=1}^{\eta} \pi_{z,0}^{-\delta_{z,1}}, g_2 \cdot h\right) \cdot e\left(\prod_{z=1}^{\eta} \pi_{z,1}^{x_{z,2} \cdot \delta_{z,3}}, U_2\right) \cdot e\left(\prod_{z=1}^{\eta} \pi_{z,2}^{x_{z,3} \cdot \delta_{z,4} \cdot -1}, U_3\right) \cdot e\left(\prod_{z=1}^{\eta} \pi_{z,3}^{x_{z,4} \cdot \delta_{z,5} \cdot -1}, U_4\right) \\ & \cdot e\left(\prod_{z=1}^{\eta} \pi_{z,4}^{x_{z,5} \cdot \delta_{z,6} \cdot -1}, U_5\right) \cdot e\left(\prod_{z=1}^{\eta} \pi_{z,5}^{x_{z,6} \cdot \delta_{z,7} \cdot -1}, U_6\right) \cdot e\left(\prod_{z=1}^{\eta} \pi_{z,6}^{x_{z,7} \cdot \delta_{z,8} \cdot -1}, U_7\right) \cdot e\left(\prod_{z=1}^{\eta} \pi_{z,7}^{x_{z,8} \cdot \delta_{z,9} \cdot -1}, U_8\right) \end{aligned}$$

We will now formally define a batch verifier and demonstrate that `VRF.BatchVerify` is a secure batch verifier for the VRF signature scheme.

Definition C.1 (Pairing-based Batch Verifier) Let $\text{BSetup}(1^\tau) \rightarrow (q, g, \mathbb{G}, \mathbb{G}_T, e)$. For each $j \in [1, \eta]$, where $\eta \in \text{poly}(\tau)$, let $X^{(j)}$ be a generic pairing-based claim and let `Verify` be a pairing based verifier. We define pairing-based batch verifier for `Verify` a probabilistic $\text{poly}(\tau)$ -time algorithm which outputs accept if $X^{(j)}$ holds for all $j \in [1, \eta]$ whereas it outputs reject if $X^{(j)}$ does not hold for any $j \in [1, \eta]$ except with negligible probability.

Theorem C.2 VRF BatchVerify is a batch verifier for the VRF signature scheme.

C.2 Proof

Proof. Via a series of steps, we will show that if VRF is a secure signature scheme, then `BatchVerify` is a secure batch verifier. Recall our batch verification software will perform a group membership test to ensure that each group element of the signature is a member of the proper subgroup, so here will we assume this fact. We begin with the original verification equation.

$$e(\pi_1, g_2) \stackrel{?}{=} e(g_1^{(1-x_1)} \cdot U_1^{x_1}, \hat{U}) \text{ and } e(\pi_0, g_2) \stackrel{?}{=} e(\pi_l, U_0) \text{ and } e(\pi_0, h) \stackrel{?}{=} y \text{ and}$$

$$\text{for } t = 2 \text{ to } \ell \text{ it holds: } e(\pi_t, g_2) \stackrel{?}{=} e(\pi_{t-1}^{(1-x_t)}, g_2) \cdot e(\pi_{t-1}^{x_t}, U_t)$$

EQ1 Step 1: Consolidate the verification equations (technique 0a), merge pairings with common first or second argument (technique 3b), and apply the small exponents test as follows: For each of the $z = 1$ to η signatures, choose random $\delta_1, \delta_2 \in [1, 2^\lambda - 1]$ and compute for each equation:

$$e(g_1^{(1-x_1)} \cdot U_1^{x_1}, \hat{U})^{\delta_2} \cdot e(\pi_1, g_2)^{-\delta_2} \stackrel{?}{=} e(\pi_l, U_0)^{\delta_1} \cdot y^{\delta_1} \cdot e(\pi_0, g_2 \cdot h)^{-\delta_1} \quad (8)$$

EQ1 Step 2: Combine η signatures (technique 1), move exponent(s) inside pairing (technique 2):

$$\prod_{z=1}^{\eta} e(g_1^{(1-x_1) \cdot \delta_{z,2}} \cdot U_1^{x_1 \cdot \delta_{z,2}}, \hat{U}) \cdot \prod_{z=1}^{\eta} e(\pi_{z,1}^{-\delta_{z,2}}, g_2) \stackrel{?}{=} \prod_{z=1}^{\eta} e(\pi_{z,l}^{\delta_{z,1}}, U_0) \cdot \prod_{z=1}^{\eta} y_z^{\delta_{z,1}} \cdot \prod_{z=1}^{\eta} e(\pi_{z,0}^{-\delta_{z,1}}, g_2 \cdot h) \quad (9)$$

EQ1 Step 3: Move products inside pairings to reduce η pairings to 1 (technique 3a):

$$e\left(\prod_{z=1}^{\eta} g_1^{(1-x_1) \cdot \delta_{z,2}} \cdot U_1^{x_1 \cdot \delta_{z,2}}, \hat{U}\right) \cdot e\left(\prod_{z=1}^{\eta} \pi_{z,1}^{-\delta_{z,2}}, g_2\right) \stackrel{?}{=} e\left(\prod_{z=1}^{\eta} \pi_{z,l}^{\delta_{z,1}}, U_0\right) \cdot \prod_{z=1}^{\eta} y_z^{\delta_{z,1}} \cdot e\left(\prod_{z=1}^{\eta} \pi_{z,0}^{-\delta_{z,1}}, g_2 \cdot h\right) \quad (10)$$

EQ2 Step 4: Combine η signatures (technique 1):

$$\text{for } t = 2 \text{ to } \ell \text{ it holds: } \prod_{z=1}^{\eta} e(\pi_{z,t}, g_2) \stackrel{?}{=} \prod_{z=1}^{\eta} e(\pi_{z,t-1}^{(1-x_{z,t})}, g_2) \cdot e(\pi_{z,t-1}^{x_{z,t}}, U_t) \quad (11)$$

EQ2 Step 5: Apply the small exponents test, using exponents $\delta_1, \dots, \delta_{\eta} \in [1, 2^{\lambda}]$:

$$\text{for } t = 2 \text{ to } \ell \text{ it holds: } \prod_{z=1}^{\eta} e(\pi_{z,t}, g_2)^{\delta_z} \stackrel{?}{=} \prod_{z=1}^{\eta} (e(\pi_{z,t-1}^{(1-x_{z,t})}, g_2) \cdot e(\pi_{z,t-1}^{x_{z,t}}, U_t))^{\delta_z} \quad (12)$$

EQ2 Step 6: Move exponent(s) inside the pairing (technique 2):

$$\text{for } t = 2 \text{ to } \ell \text{ it holds: } \prod_{z=1}^{\eta} e(\pi_{z,t}^{\delta_z}, g_2) \stackrel{?}{=} \prod_{z=1}^{\eta} e(\pi_{z,t-1}^{(1-x_{z,t}) \cdot \delta_z}, g_2) \cdot e(\pi_{z,t-1}^{x_{z,t} \cdot \delta_z}, U_t) \quad (13)$$

EQ2 Step 7: Move products inside pairings to reduce η pairings to 1 (technique 3a):

$$\text{for } t = 2 \text{ to } \ell \text{ it holds: } e\left(\prod_{z=1}^{\eta} \pi_{z,t}^{\delta_z}, g_2\right) \stackrel{?}{=} \prod_{z=1}^{\eta} e(\pi_{z,t-1}^{(1-x_{z,t}) \cdot \delta_z}, g_2) \cdot e(\pi_{z,t-1}^{x_{z,t} \cdot \delta_z}, U_t) \quad (14)$$

EQ2 Step 8: Distribute products (technique 5):

$$\text{for } t = 2 \text{ to } \ell \text{ it holds: } e\left(\prod_{z=1}^{\eta} \pi_{z,t}^{\delta_z}, g_2\right) \stackrel{?}{=} \prod_{z=1}^{\eta} e(\pi_{z,t-1}^{(1-x_{z,t}) \cdot \delta_z}, g_2) \cdot \prod_{z=1}^{\eta} e(\pi_{z,t-1}^{x_{z,t} \cdot \delta_z}, U_t) \quad (15)$$

EQ2 Step 9: Move products inside pairings to reduce η pairings to 1 (technique 3a):

$$\text{for } t = 2 \text{ to } \ell \text{ it holds: } e\left(\prod_{z=1}^{\eta} \pi_{z,t}^{\delta_z}, g_2\right) \stackrel{?}{=} e\left(\prod_{z=1}^{\eta} \pi_{z,t-1}^{(1-x_{z,t}) \cdot \delta_z}, g_2\right) \cdot e\left(\prod_{z=1}^{\eta} \pi_{z,t-1}^{x_{z,t} \cdot \delta_z}, U_t\right) \quad (16)$$

EQ2 Step 10: Merge pairings with common first or second argument (technique 3b):

$$\text{for } t = 2 \text{ to } \ell \text{ it holds: } e\left(\prod_{z=1}^{\eta} \pi_{z,t}^{\delta_z} \cdot \pi_{z,t-1}^{(1-x_{z,t}) \cdot \delta_z}, g_2\right) \stackrel{?}{=} e\left(\prod_{z=1}^{\eta} \pi_{z,t-1}^{x_{z,t} \cdot \delta_z}, U_t\right) \quad (17)$$

EQ2 Step 11: Unroll for loop (technique 0b) and choose random $\delta_3, \dots, \delta_9 \in [1, 2^{\lambda} - 1]$ for each $z = 1$ to

η equations:

$$\begin{aligned}
& e\left(\prod_{z=1}^{\eta} \pi_{z,2}^{\delta_{z,3}} \cdot \pi_{z,1}^{(1-x_{z,2}) \cdot -\delta_{z,3}} \cdot \pi_{z,3}^{-\delta_{z,4}} \cdot \pi_{z,2}^{(1-x_{z,3}) \cdot -\delta_{z,4} - 1} \cdot \pi_{z,4}^{-\delta_{z,5}} \cdot \pi_{z,3}^{(1-x_{z,4}) \cdot -\delta_{z,5} - 1} \cdot \pi_{z,5}^{-\delta_{z,6}} \cdot \pi_{z,4}^{(1-x_{z,5}) \cdot -\delta_{z,6} - 1} \right. \\
& \quad \cdot \pi_{z,6}^{-\delta_{z,7}} \cdot \pi_{z,5}^{(1-x_{z,6}) \cdot -\delta_{z,7} - 1} \cdot \pi_{z,7}^{-\delta_{z,8}} \cdot \pi_{z,6}^{(1-x_{z,7}) \cdot -\delta_{z,8} - 1} \cdot \pi_{z,8}^{-\delta_{z,9}} \cdot \pi_{z,7}^{(1-x_{z,8}) \cdot -\delta_{z,9} - 1}, g_2) \stackrel{?}{=} \\
& e\left(\prod_{z=1}^{\eta} \pi_{z,1}^{x_{z,2} \cdot \delta_{z,3}}, U_2\right) \cdot e\left(\prod_{z=1}^{\eta} \pi_{z,2}^{x_{z,3} \cdot \delta_{z,4} - 1}, U_3\right) \cdot e\left(\prod_{z=1}^{\eta} \pi_{z,3}^{x_{z,4} \cdot \delta_{z,5} - 1}, U_4\right) \cdot e\left(\prod_{z=1}^{\eta} \pi_{z,4}^{x_{z,5} \cdot \delta_{z,6} - 1}, U_5\right) \\
& \quad \cdot e\left(\prod_{z=1}^{\eta} \pi_{z,5}^{x_{z,6} \cdot \delta_{z,7} - 1}, U_6\right) \cdot e\left(\prod_{z=1}^{\eta} \pi_{z,6}^{x_{z,7} \cdot \delta_{z,8} - 1}, U_7\right) \cdot e\left(\prod_{z=1}^{\eta} \pi_{z,7}^{x_{z,8} \cdot \delta_{z,9} - 1}, U_8\right) \quad (18)
\end{aligned}$$

Step 12: Combine equations 1 and 2, then pairings within final equation (technique 3b):

$$\begin{aligned}
& e\left(\prod_{z=1}^{\eta} g_1^{(1-x_1) \cdot \delta_{z,2}} \cdot U_1^{x_1 \cdot \delta_{z,2}} \cdot \hat{U}\right) \cdot e\left(\prod_{z=1}^{\eta} \pi_{z,1}^{-\delta_{z,2}} \cdot \pi_{z,2}^{\delta_{z,3}} \cdot \pi_{z,1}^{(1-x_{z,2}) \cdot -\delta_{z,3}} \cdot \pi_{z,3}^{-\delta_{z,4}} \cdot \pi_{z,2}^{(1-x_{z,3}) \cdot -\delta_{z,4} - 1} \cdot \pi_{z,4}^{-\delta_{z,5}} \cdot \pi_{z,3}^{(1-x_{z,4}) \cdot -\delta_{z,5} - 1} \right. \\
& \quad \cdot \pi_{z,5}^{-\delta_{z,6}} \cdot \pi_{z,4}^{(1-x_{z,5}) \cdot -\delta_{z,6} - 1} \cdot \pi_{z,6}^{-\delta_{z,7}} \cdot \pi_{z,5}^{(1-x_{z,6}) \cdot -\delta_{z,7} - 1} \cdot \pi_{z,7}^{-\delta_{z,8}} \cdot \pi_{z,6}^{(1-x_{z,7}) \cdot -\delta_{z,8} - 1} \cdot \pi_{z,8}^{-\delta_{z,9}} \cdot \pi_{z,7}^{(1-x_{z,8}) \cdot -\delta_{z,9} - 1}, g_2) \stackrel{?}{=} \\
& e\left(\prod_{z=1}^{\eta} \pi_{z,l}^{\delta_{z,1}}, U_0\right) \cdot \prod_{z=1}^{\eta} y_z^{\delta_{z,1}} \cdot e\left(\prod_{z=1}^{\eta} \pi_{z,0}^{-\delta_{z,1}}, g_2 \cdot h\right) \cdot e\left(\prod_{z=1}^{\eta} \pi_{z,1}^{x_{z,2} \cdot \delta_{z,3}}, U_2\right) \cdot e\left(\prod_{z=1}^{\eta} \pi_{z,2}^{x_{z,3} \cdot \delta_{z,4} - 1}, U_3\right) \cdot e\left(\prod_{z=1}^{\eta} \pi_{z,3}^{x_{z,4} \cdot \delta_{z,5} - 1}, U_4\right) \\
& \quad \cdot e\left(\prod_{z=1}^{\eta} \pi_{z,4}^{x_{z,5} \cdot \delta_{z,6} - 1}, U_5\right) \cdot e\left(\prod_{z=1}^{\eta} \pi_{z,5}^{x_{z,6} \cdot \delta_{z,7} - 1}, U_6\right) \cdot e\left(\prod_{z=1}^{\eta} \pi_{z,6}^{x_{z,7} \cdot \delta_{z,8} - 1}, U_7\right) \cdot e\left(\prod_{z=1}^{\eta} \pi_{z,7}^{x_{z,8} \cdot \delta_{z,9} - 1}, U_8\right) \quad (19)
\end{aligned}$$

Steps 1 and 2 form the Combination Step in [28], which was proven to result in a secure batch verifier in [28, Theorem 3.2]. We observe that the remaining steps are merely reorganizing terms within the same equation except for the application of technique 0b, which applies the small exponents test again while unrolling the loop. Hence, the final verification equation (19) is also batch verifier for VRF. $\square$

D A machine-generated candidate batch algorithm for WATERS09

The following candidate batching algorithm was automatically generated by the Batcher while processing the WATERS09 signature scheme [66, 67]. This execution was restricted to signatures on a single signing key.

D.1 Definitions

Let g_1, g_2 be values drawn from the key and/or parameters, and $M, \sigma_1, \sigma_2, \sigma_3, \sigma_4, \sigma_5, \sigma_6, \sigma_7, \sigma_K, tag_k$ represent a message (or message hash) and signature. Select s_1, s_2, t, tag_c variables at random in $\mathbb{Z}_q$ and the variables θ, A are computed as follows: $\theta = 1/(tag_c - tag_k), A = e(g, g)^{\alpha \cdot a_1 \cdot b}$. The individual verification equation WATERS09.Verify [§6.1]²⁰ is:

$$\begin{aligned}
& e(g_1^{bs}, \sigma_1) \cdot e(g_1^{b \cdot a_1 s_1}, \sigma_2) \cdot e(g_1^{a_1 s_1}, \sigma_3) \cdot e(g_1^{b \cdot a_2 s_2}, \sigma_4) \cdot e(g_1^{a_2 s_2}, \sigma_5) \stackrel{?}{=} \\
& e(\sigma_6, \tau_1^{s_1} \cdot \tau_2^{s_2}) \cdot e(\sigma_7, \tau_1^{bs_1} \cdot \tau_2^{bs_2} \cdot w^{-t}) \cdot (e(\sigma_7, u^{M \cdot t} \cdot w^{tag_c \cdot t} \cdot h^t) \cdot e(g_1^{-t}, \sigma_K))^{\theta} \cdot A^{s_2}
\end{aligned}$$

²⁰For simplicity, Waters [67] presents this verification equation as a series of calculations. We have merely combined these calculations, reorganized a few terms in the verification equation and turned division operations into multiplication.

Let η be the number of signatures in a batch, and $\delta_1, \dots, \delta_\eta \in \{1, 2^\lambda - 1\}$ be a set of random exponents chosen by the verifier. The batch verification equation WATERS09.BatchVerify is:

$$\begin{aligned}
& e(g_1^b, \prod_{z=1}^{\eta} \sigma_{z,1}^{s_z \cdot \delta_z}) \cdot e(g_1^{b \cdot a_1}, \prod_{z=1}^{\eta} \sigma_{z,2}^{s_z \cdot \delta_z}) \cdot e(g_1^{a_1}, \prod_{z=1}^{\eta} \sigma_{z,3}^{s_z \cdot \delta_z}) \cdot e(g_1^{b \cdot a_2}, \prod_{z=1}^{\eta} \sigma_{z,4}^{s_z \cdot \delta_z}) \cdot e(g_1^{a_2}, \prod_{z=1}^{\eta} \sigma_{z,5}^{s_z \cdot \delta_z}) \stackrel{?}{=} \\
& e(\prod_{z=1}^{\eta} \sigma_{z,6}^{\delta_z \cdot s_z \cdot 1}, \tau_1) \cdot e(\prod_{z=1}^{\eta} \sigma_{z,6}^{\delta_z \cdot s_z \cdot 2}, \tau_2) \cdot e(\prod_{z=1}^{\eta} \sigma_{z,7}^{\delta_z \cdot s_z \cdot 1}, \tau_1^b) \cdot e(\prod_{z=1}^{\eta} \sigma_{z,7}^{\delta_z \cdot s_z \cdot 2}, \tau_2^b) \cdot e(\prod_{z=1}^{\eta} \sigma_{z,7}^{(\delta_z \cdot t_z + \theta_z \cdot \delta_z \cdot \text{tag}_{c \cdot t_z})}, w) \\
& \cdot e(\prod_{z=1}^{\eta} \sigma_{z,7}^{\theta_z \cdot \delta_z \cdot M_z \cdot t_z}, u) \cdot e(\prod_{z=1}^{\eta} \sigma_{z,7}^{\theta_z \cdot \delta_z \cdot t_z}, h) \cdot e(g_1, \prod_{z=1}^{\eta} \sigma_{z,K}^{-t_z \cdot \theta_z \cdot \delta_z}) \cdot A^{\sum_{z=0}^{\eta} s_z \cdot 2 \cdot \delta_z}
\end{aligned}$$

We conjecture that this scheme satisfies a relaxation of Definition 2.2 to allow for two-sided negligible error; that is, where there is also a chance that a set of valid signatures will be rejected by the BatchVer.

D.2 Details on How Candidate Construction was Derived

Via a series of steps, we show how the above batching algorithm was derived. We begin with the original verification equation.

$$\begin{aligned}
& e(g_1^{bs}, \sigma_1) \cdot e(g_1^{b \cdot a_1 s_1}, \sigma_2) \cdot e(g_1^{a_1 s_1}, \sigma_3) \cdot e(g_1^{b \cdot a_2 s_2}, \sigma_4) \cdot e(g_1^{a_2 s_2}, \sigma_5) \stackrel{?}{=} \\
& e(\sigma_6, \tau_1^{s_1} \cdot \tau_2^{s_2}) \cdot e(\sigma_7, \tau_1^{bs_1} \cdot \tau_2^{bs_2} \cdot w^{-t}) \cdot (e(\sigma_7, u^{M \cdot t} \cdot w^{\text{tag}_{c \cdot t}} \cdot h^t) \cdot e(g_1^{-t}, \sigma_K))^\theta \cdot A^{s_2} \quad (20)
\end{aligned}$$

Step 1: Combine η signatures (technique 1):

$$\begin{aligned}
& \prod_{z=1}^{\eta} e(g_1^{bs_z}, \sigma_{z,1}) \cdot e(g_1^{b \cdot a_1 s_{z,1}}, \sigma_{z,2}) \cdot e(g_1^{a_1 s_{z,1}}, \sigma_{z,3}) \cdot e(g_1^{b \cdot a_2 s_{z,2}}, \sigma_{z,4}) \cdot e(g_1^{a_2 s_{z,2}}, \sigma_{z,5}) \stackrel{?}{=} \\
& \prod_{z=1}^{\eta} e(\sigma_{z,6}, \tau_1^{s_{z,1}} \cdot \tau_2^{s_{z,2}}) \cdot e(\sigma_{z,7}, \tau_1^{bs_{z,1}} \cdot \tau_2^{bs_{z,2}} \cdot w^{-t_z}) \\
& \cdot (e(\sigma_{z,7}, u^{M_z \cdot t_z} \cdot w^{\text{tag}_{c \cdot t_z}} \cdot h^{t_z}) \cdot e(g_1^{-t_z}, \sigma_{z,K}))^{\theta_z} \cdot A^{s_{z,2}} \quad (21)
\end{aligned}$$

Step 2: Apply the small exponents test, using exponents $\delta_1, \dots, \delta_\eta \in [1, 2^\lambda - 1]$:

$$\begin{aligned}
& \prod_{z=1}^{\eta} (e(g_1^{bs_z}, \sigma_{z,1}) \cdot e(g_1^{b \cdot a_1 s_{z,1}}, \sigma_{z,2}) \cdot e(g_1^{a_1 s_{z,1}}, \sigma_{z,3}) \cdot e(g_1^{b \cdot a_2 s_{z,2}}, \sigma_{z,4}) \cdot e(g_1^{a_2 s_{z,2}}, \sigma_{z,5}))^{\delta_z} \stackrel{?}{=} \\
& \prod_{z=1}^{\eta} (e(\sigma_{z,6}, \tau_1^{s_{z,1}} \cdot \tau_2^{s_{z,2}}) \cdot e(\sigma_{z,7}, \tau_1^{bs_{z,1}} \cdot \tau_2^{bs_{z,2}} \cdot w^{-t_z}) \\
& \cdot (e(\sigma_{z,7}, u^{M_z \cdot t_z} \cdot w^{\text{tag}_{c \cdot t_z}} \cdot h^{t_z}) \cdot e(g_1^{-t_z}, \sigma_{z,K}))^{\theta_z} \cdot A^{s_{z,2}})^{\delta_z} \quad (22)
\end{aligned}$$

Step 3: Move exponent(s) inside the pairing (technique 2):

$$\begin{aligned}
& \prod_{z=1}^{\eta} e(g_1^{bs_z \cdot \delta_z}, \sigma_{z,1}) \cdot e(g_1^{b \cdot a_1 s_{z,1} \cdot \delta_z}, \sigma_{z,2}) \cdot e(g_1^{a_1 s_{z,1} \cdot \delta_z}, \sigma_{z,3}) \cdot e(g_1^{b \cdot a_2 s_{z,2} \cdot \delta_z}, \sigma_{z,4}) \cdot e(g_1^{a_2 s_{z,2} \cdot \delta_z}, \sigma_{z,5}) \stackrel{?}{=} \\
& \prod_{z=1}^{\eta} e(\sigma_{z,6}^{\delta_z}, \tau_1^{s_{z,1}} \cdot \tau_2^{s_{z,2}}) \cdot e(\sigma_{z,7}^{\delta_z}, \tau_1^{bs_{z,1}} \cdot \tau_2^{bs_{z,2}} \cdot w^{-t_z}) \\
& \cdot e(\sigma_{z,7}^{\theta_z \cdot \delta_z}, u^{M_z \cdot t_z} \cdot w^{\text{tag}_{c \cdot t_z}} \cdot h^{t_z}) \cdot e(g_1^{-t_z \cdot \theta_z \cdot \delta_z}, \sigma_{z,K}) \cdot A^{s_{z,2} \cdot \delta_z} \quad (23)
\end{aligned}$$

Step 4: Split pairings (technique 8):

$$\begin{aligned}
& \prod_{z=1}^{\eta} e(g_1^{b s_z \cdot \delta_z}, \sigma_{z,1}) \cdot e(g_1^{b \cdot a_1 s_{z,1} \cdot \delta_z}, \sigma_{z,2}) \cdot e(g_1^{a_1 s_{z,1} \cdot \delta_z}, \sigma_{z,3}) \cdot e(g_1^{b \cdot a_2 s_{z,2} \cdot \delta_z}, \sigma_{z,4}) \cdot e(g_1^{a_2 s_{z,2} \cdot \delta_z}, \sigma_{z,5}) \stackrel{?}{=} \\
& \prod_{z=1}^{\eta} e(\sigma_{z,6}^{\delta_z}, \tau_1^{s_{z,1}}) \cdot e(\sigma_{z,6}^{\delta_z}, \tau_2^{s_{z,2}}) \cdot e(\sigma_{z,7}^{\delta_z}, \tau_1^{b s_{z,1}}) \cdot e(\sigma_{z,7}^{\delta_z}, \tau_2^{b s_{z,2}}) \cdot e(\sigma_{z,7}^{\delta_z}, w^{-t_z}) \cdot e(\sigma_{z,7}^{\theta_z \cdot \delta_z}, u^{M_z \cdot t_z}) \\
& \cdot e(\sigma_{z,7}^{\theta_z \cdot \delta_z}, w^{tag_{z,c} \cdot t_z}) \cdot e(\sigma_{z,7}^{\theta_z \cdot \delta_z}, h^{t_z}) \cdot e(g_1^{-t_z \cdot \theta_z \cdot \delta_z}, \sigma_{z,K}) \cdot A^{s_{z,2} \cdot \delta_z} \quad (24)
\end{aligned}$$

Step 5: Distribute products (technique 5):

$$\begin{aligned}
& \prod_{z=1}^{\eta} e(g_1^{b s_z \cdot \delta_z}, \sigma_{z,1}) \cdot \prod_{z=1}^{\eta} e(g_1^{b \cdot a_1 s_{z,1} \cdot \delta_z}, \sigma_{z,2}) \cdot \prod_{z=1}^{\eta} e(g_1^{a_1 s_{z,1} \cdot \delta_z}, \sigma_{z,3}) \cdot \prod_{z=1}^{\eta} e(g_1^{b \cdot a_2 s_{z,2} \cdot \delta_z}, \sigma_{z,4}) \cdot \prod_{z=1}^{\eta} e(g_1^{a_2 s_{z,2} \cdot \delta_z}, \sigma_{z,5}) \stackrel{?}{=} \\
& \prod_{z=1}^{\eta} e(\sigma_{z,6}^{\delta_z}, \tau_1^{s_{z,1}}) \cdot \prod_{z=1}^{\eta} e(\sigma_{z,6}^{\delta_z}, \tau_2^{s_{z,2}}) \cdot \prod_{z=1}^{\eta} e(\sigma_{z,7}^{\delta_z}, \tau_1^{b s_{z,1}}) \cdot \prod_{z=1}^{\eta} e(\sigma_{z,7}^{\delta_z}, \tau_2^{b s_{z,2}}) \cdot \prod_{z=1}^{\eta} e(\sigma_{z,7}^{\delta_z}, w^{-t_z}) \cdot \prod_{z=1}^{\eta} e(\sigma_{z,7}^{\theta_z \cdot \delta_z}, u^{M_z \cdot t_z}) \\
& \cdot \prod_{z=1}^{\eta} e(\sigma_{z,7}^{\theta_z \cdot \delta_z}, w^{tag_{z,c} \cdot t_z}) \cdot \prod_{z=1}^{\eta} e(\sigma_{z,7}^{\theta_z \cdot \delta_z}, h^{t_z}) \cdot \prod_{z=1}^{\eta} e(g_1^{-t_z \cdot \theta_z \cdot \delta_z}, \sigma_{z,K}) \cdot \prod_{z=1}^{\eta} A^{s_{z,2} \cdot \delta_z} \quad (25)
\end{aligned}$$

Step 6: Move products inside pairings to reduce η pairings to 1 (technique 3a) and move product to summation on precomputed pairing (technique 6):

$$\begin{aligned}
& e(g_1^b, \prod_{z=1}^{\eta} \sigma_{z,1}^{s_z \cdot \delta_z}) \cdot e(g_1^{b \cdot a_1}, \prod_{z=1}^{\eta} \sigma_{z,2}^{s_{z,1} \cdot \delta_z}) \cdot e(g_1^{a_1}, \prod_{z=1}^{\eta} \sigma_{z,3}^{s_{z,1} \cdot \delta_z}) \cdot e(g_1^{b \cdot a_2}, \prod_{z=1}^{\eta} \sigma_{z,4}^{s_{z,2} \cdot \delta_z}) \cdot e(g_1^{a_2}, \prod_{z=1}^{\eta} \sigma_{z,5}^{s_{z,2} \cdot \delta_z}) \stackrel{?}{=} \\
& e(\prod_{z=1}^{\eta} \sigma_{z,6}^{\delta_z \cdot s_{z,1}}, \tau_1) \cdot e(\prod_{z=1}^{\eta} \sigma_{z,6}^{\delta_z \cdot s_{z,2}}, \tau_2) \cdot e(\prod_{z=1}^{\eta} \sigma_{z,7}^{\delta_z \cdot s_{z,1}}, \tau_1^b) \cdot e(\prod_{z=1}^{\eta} \sigma_{z,7}^{\delta_z \cdot s_{z,2}}, \tau_2^b) \cdot e(\prod_{z=1}^{\eta} \sigma_{z,7}^{\delta_z \cdot -t_z}, w) \cdot e(\prod_{z=1}^{\eta} \sigma_{z,7}^{\theta_z \cdot \delta_z \cdot M_z \cdot t_z}, u) \\
& \cdot e(\prod_{z=1}^{\eta} \sigma_{z,7}^{\theta_z \cdot \delta_z \cdot tag_{z,c} \cdot t_z}, w) \cdot e(\prod_{z=1}^{\eta} \sigma_{z,7}^{\theta_z \cdot \delta_z \cdot t_z}, h) \cdot e(g_1, \prod_{z=1}^{\eta} \sigma_{z,K}^{-t_z \cdot \theta_z \cdot \delta_z}) \cdot A^{\sum_{z=0}^{\eta} s_{z,2} \cdot \delta_z} \quad (26)
\end{aligned}$$

Step 7: Merge pairings with common first or second argument (technique 3b):

$$\begin{aligned}
& e(g_1^b, \prod_{z=1}^{\eta} \sigma_{z,1}^{s_z \cdot \delta_z}) \cdot e(g_1^{b \cdot a_1}, \prod_{z=1}^{\eta} \sigma_{z,2}^{s_{z,1} \cdot \delta_z}) \cdot e(g_1^{a_1}, \prod_{z=1}^{\eta} \sigma_{z,3}^{s_{z,1} \cdot \delta_z}) \cdot e(g_1^{b \cdot a_2}, \prod_{z=1}^{\eta} \sigma_{z,4}^{s_{z,2} \cdot \delta_z}) \cdot e(g_1^{a_2}, \prod_{z=1}^{\eta} \sigma_{z,5}^{s_{z,2} \cdot \delta_z}) \stackrel{?}{=} \\
& e(\prod_{z=1}^{\eta} \sigma_{z,6}^{\delta_z \cdot s_{z,1}}, \tau_1) \cdot e(\prod_{z=1}^{\eta} \sigma_{z,6}^{\delta_z \cdot s_{z,2}}, \tau_2) \cdot e(\prod_{z=1}^{\eta} \sigma_{z,7}^{\delta_z \cdot s_{z,1}}, \tau_1^b) \cdot e(\prod_{z=1}^{\eta} \sigma_{z,7}^{\delta_z \cdot s_{z,2}}, \tau_2^b) \cdot e(\prod_{z=1}^{\eta} \sigma_{z,7}^{(\delta_z \cdot -t_z + \theta_z \cdot \delta_z \cdot tag_{z,c} \cdot t_z)}, w) \\
& \cdot e(\prod_{z=1}^{\eta} \sigma_{z,7}^{\theta_z \cdot \delta_z \cdot M_z \cdot t_z}, u) \cdot e(\prod_{z=1}^{\eta} \sigma_{z,7}^{\theta_z \cdot \delta_z \cdot t_z}, h) \cdot e(g_1, \prod_{z=1}^{\eta} \sigma_{z,K}^{-t_z \cdot \theta_z \cdot \delta_z}) \cdot A^{\sum_{z=0}^{\eta} s_{z,2} \cdot \delta_z} \quad (27)
\end{aligned}$$

E SDL Grammar

We provide a full description of our SDL grammar below:

$$\begin{aligned}
\langle \text{assign-statement} \rangle &::= \langle \text{single-assignment} \rangle \mid \langle \text{func-call-statement} \rangle \\
&\mid \langle \text{group-assign-statement} \rangle \mid \langle \text{dict-statement} \rangle \mid \langle \text{random-statement} \rangle \mid \langle \text{hash-statement} \rangle \mid \langle \text{pair-statement} \rangle.
\end{aligned}$$

$\langle \text{single-assignment} \rangle ::= \langle \text{variable-target} \rangle \langle \text{assign-op} \rangle \langle \text{expr-statement} \rangle \mid \langle \text{variable-target} \rangle.$
 $\langle \text{variable-target} \rangle ::= \langle \text{keywords} \rangle \mid \langle \text{variable-name} \rangle.$
 $\langle \text{group-assign-statement} \rangle ::= \langle \text{variable-name} \rangle \langle \text{assign-op} \rangle \langle \text{type} \rangle.$
 $\langle \text{func-call-statement} \rangle ::= \langle \text{variable-name} \rangle \langle \text{assign-op} \rangle \langle \text{variable-name} \rangle \text{'('} \langle \text{arg-list} \rangle \text{'}' }.$
 $\langle \text{arg-list} \rangle ::= \langle \text{arg-name} \rangle \mid \langle \text{arg-name} \rangle \text{' ,' } \langle \text{arg-list} \rangle.$
 $\langle \text{arg-name} \rangle ::= \langle \text{variable-name} \rangle.$
 $\langle \text{random-statement} \rangle ::= \langle \text{variable-name} \rangle \langle \text{assign-op} \rangle \langle \text{random-func-name} \rangle \text{'('} \langle \text{group-type} \rangle \text{'}' }.$
 $\langle \text{hash-statement} \rangle ::= \langle \text{variable-name} \rangle \langle \text{assign-op} \rangle \langle \text{hash-func-name} \rangle \text{'('} \langle \text{arg-list} \rangle \text{' ,' } \langle \text{group-type} \rangle \text{'}' }.$
 $\langle \text{dict-statement} \rangle ::= \langle \text{variable-name} \rangle \langle \text{assign-op} \rangle \text{'list{' } } \langle \text{arg-list} \rangle \text{'}' } \mid \text{'expand{' } } \langle \text{arg-list} \rangle \text{'}' }.$
 $\langle \text{dot-prod-statement} \rangle ::= \text{'prod{' } } \langle \text{single-assignment} \rangle \text{' ,' } \langle \text{single-assignment} \rangle \text{'}' } \text{ on ' } \langle \text{expr-statement} \rangle.$
 $\langle \text{sum-of-statement} \rangle ::= \text{'sum{' } } \langle \text{single-assignment} \rangle \text{' ,' } \langle \text{single-assignment} \rangle \text{'}' } \text{ of ' } \langle \text{expr-statement} \rangle$
 $\langle \text{for-statement} \rangle ::= \langle \text{proc-token} \rangle \langle \text{block-sep} \rangle \text{'for'}$
 $\mid \text{'for{' } } \langle \text{assign-statement} \rangle \text{ , } \langle \text{assign-statement} \rangle \text{'}' } [\langle \text{new-line} \rangle \langle \text{expr-statement} \rangle]^*$
 $\mid \text{forall{' } } \langle \text{assign-statement} \rangle \text{ , } \langle \text{assign-statement} \rangle \text{'}' } [\langle \text{new-line} \rangle \langle \text{expr-statement} \rangle]^*.$
 $\langle \text{conditional-statement} \rangle ::= \langle \text{proc-token} \rangle \langle \text{block-sep} \rangle \text{'if'}$
 $\mid \text{'if{' } } \langle \text{cond-statement} \rangle \text{'}' } [\langle \text{new-line} \rangle \langle \text{expr-statement} \rangle]^+$
 $\mid \text{'else' } [\langle \text{new-line} \rangle \langle \text{expr-statement} \rangle]^+.$
 $\langle \text{expr-statement} \rangle ::= \langle \text{pair-statement} \rangle \mid \langle \text{expr0-statement} \rangle.$
 $\langle \text{expr0-statement} \rangle ::= \langle \text{expr0-statement} \rangle \langle \text{group-op} \rangle \langle \text{expr0-statement} \rangle$
 $\mid \langle \text{exp1-statement} \rangle$
 $\mid \langle \text{variable-name} \rangle.$
 $\langle \text{cond-statement} \rangle ::= \langle \text{cond-statement} \rangle [\langle \text{bool-op} \rangle \langle \text{cond-statement} \rangle]^* \mid \langle \text{expr-statement} \rangle.$
 $\langle \text{pair-statement} \rangle ::= \text{'e(' } \langle \text{expr-statement} \rangle \text{' ,' } \langle \text{expr-statement} \rangle \text{'}' }$
 $\mid \langle \text{pair-statement} \rangle \langle \text{group-op} \rangle \langle \text{pair-statement} \rangle$
 $\mid \langle \text{pair-statement} \rangle \langle \text{exp} \rangle \langle \text{exp1-statement} \rangle.$
 $\langle \text{exp1-statement} \rangle ::= \langle \text{exp1-statement} \rangle \langle \text{exp} \rangle \langle \text{exp2-statement} \rangle \mid \langle \text{exp2-statement} \rangle$
 $\langle \text{exp2-statement} \rangle ::= \langle \text{expr-statement} \rangle \langle \text{exp-ops} \rangle \langle \text{expr-statement} \rangle$
 $\mid \langle \text{expr-statement} \rangle \langle \text{group-op} \rangle \langle \text{expr-statement} \rangle$
 $\mid \langle \text{negate-op} \rangle [\langle \text{exp2-statement} \rangle \mid \langle \text{integer} \rangle].$
 $\langle \text{element-type} \rangle ::= \text{None} \mid \text{int} \mid \text{str} \mid \text{ZR} \mid \text{G1} \mid \text{G2} \mid \text{GT}$
 $\langle \text{type} \rangle ::= \langle \text{element-type} \rangle \mid \text{'list{' } } \langle \text{element-type} \rangle \text{'}' } [\langle \text{element-type} \rangle]^* \text{'}' }$
 $\mid \text{'expand{' } } \langle \text{element-type} \rangle \text{'}' } [\langle \text{element-type} \rangle]^* \text{'}' }$
 $\mid \langle \text{type-list} \rangle.$

$\langle type-list \rangle ::= \langle type \rangle \text{' ; ' } \langle type-list \rangle \mid \langle type \rangle.$
 $\langle procedure \rangle ::= \langle proc-token \rangle \langle block-sep \rangle \langle procedure-name \rangle.$
 $\langle procedure-name \rangle ::= \langle variable-name \rangle \mid \text{' func: ' } \langle procedure-name \rangle.$
 $\langle variable-name \rangle ::= [0-9, a-z, A-Z, \langle symbols \rangle]^*$
 $\langle symbols \rangle ::= \text{' _ ' } \mid \text{' \# ' } \mid \text{' ? ' } \mid \text{' \$ '}$
 $\langle proc-token \rangle ::= \text{' BEGIN ' } \mid \text{' END '}$
 $\langle block-sep \rangle ::= \text{' :: '}$
 $\langle random-func-name \rangle ::= \text{' random '}$
 $\langle hash-func-name \rangle ::= \text{' H '}$
 $\langle keywords \rangle ::= \text{' N ' } \mid \text{' verify ' } \mid \text{' constant ' } \mid \text{' public ' } \mid \text{' signature ' } \mid \text{' message ' } \mid \text{' public\_count ' } \mid \text{' signature\_count '}$
 $\quad \mid \text{' message\_count ' } \mid \text{' latex ' } \mid \text{' precompute ' } \mid \text{' types ' } \mid \text{' name ' } \mid \text{' setting ' } \mid \text{' symmetric ' } \mid \text{' asymmetric '}$
 $\langle assign-op \rangle ::= \text{' := '}$
 $\langle exp \rangle ::= \text{' ^ '}$
 $\langle exp-ops \rangle ::= \text{' + ' } \mid \text{' - ' } \mid \langle group-op \rangle$
 $\langle group-op \rangle ::= \text{' * ' } \mid \text{' / '}$
 $\langle bool-op \rangle ::= \text{' | ' } \mid \text{' and ' } \mid \text{' or ' } \mid \text{' == ' } \mid \text{' != ' } \mid \text{' < ' } \mid \text{' <= ' } \mid \text{' > ' } \mid \text{' >= '}$
 $\langle negate-op \rangle ::= \text{' - '}$

F Semantics of SDL

We provide a brief overview of our domain specific language and examples of how schemes are written in it. SDL can accommodate a full description of pairing schemes in situations where an existing implementation of a signature scheme does not exist or a developer prefers to code their scheme directly in SDL. This information is used to inform AutoBatch on details needed to generate the scheme implementation and the batch algorithm. The SDL file consists of two parts.

The first part is a full representation of the signature scheme which consists of the descriptions of each algorithm such as **keygen**, **sign**, **verify** and a **types** section. This information is used to generate executable code for the scheme either in Python or C++.

The second part is a broken down version of the verification algorithm in a form for AutoBatch to derive the desired batch verification algorithm. To this end, there are several keywords used to provide context for AutoBatch. **Public**, **signature** and **message** keywords are used to identify the public key variables and the signature and message variables. Additionally, the **public_count** keyword is used to determine whether public keys belong to the same or different signers. The **signature_count** and **message_count** keywords describe the number of signatures and messages expected per batch. The **constants** keyword describes variables in the scheme shared by signers such as the generators of a group. **Precompute** section represents computation steps necessary before each verification check. The **verify** keyword is used to describe the verification equation as a mathematical expression. Finally, we include a block for LaTeX to assist the proof generator map variables in SDL to equivalent LaTeX representation.

Our abstract language is capable of representing a variety of programming constructs such as dot products, for loops, summation, and boolean operators. Thus, very complex schemes can be described using our SDL and to reflect this we provide full SDL descriptions below for BLS [16] and CL04 [20]. See our github repository for other full SDL examples.

```
#####
##              BLS signature scheme              ##
#####
name := bls
# expected batch size per some time period
N := 100
setting := asymmetric

# types for variables used in verification.
# all other variable types are inferred by SDL parser
BEGIN :: types
  M := Str
END :: types

# description of key generation, signing, and verification algorithms
BEGIN :: func:keygen
input := list{None}
  # choose random generator in a prime order group G2
  g := random(G2)
  # choose random integer modulo prime r
  x := random(ZR)
  pk := g^x
  sk := x
  # keygen returns a tuple consisting of three elements
output := list{pk, sk, g}
END :: func:keygen

BEGIN :: func:sign
input := list{sk, M}
  # H is a general purpose hash function that maps its inputs
  # (consisting of strings, group elements, etc)
  # to a particular target group (either ZR, G1 or G2)
  sig := (H(M, G1)^sk)
output := sig
END :: func:sign

BEGIN :: func:verify
input := list{pk, M, sig, g}
  h := H(M, G1)
  BEGIN :: if
    if {e(h, pk) == e(sig, g)}
      output := True
    else
      output := False
  END :: if
END :: func:verify
```

```

# Batchers SDL input
constant := g
public := pk
signature := sig
message := h

# same signer
BEGIN :: count
  message_count := N
  public_count := one
  signature_count := N
END :: count

# variables computed before each signature verification
BEGIN :: precompute
  h := H(M, G1)
END :: precompute

# verification equation
verify := {e(h, pk) == e(sig, g)}

#####
##              CL signature scheme              ##
#####
name := cl04
N := 100
setting := asymmetric

BEGIN :: types
  M := Str
  # represents a list of elements in group G2
  sig := list{G2}
END :: types

BEGIN :: func:setup
input := list{None}
  g := random(G1)
output := g
END :: func:setup

BEGIN :: func:keygen
input := list{g}
  x := random(ZR)
  y := random(ZR)
  X := g^x
  Y := g^y
  sk := list{x, y}
  pk := list{X, Y}
output := list{pk, sk}
END :: func:keygen

```

```

BEGIN :: func:sign
input := list{sk, M}
# expand macro is shorthand for extracting the variables contained
# in the list or tuple to make them accessible within the function
sk := expand{x, y}
a := random(G2)
m := H(M, ZR)
b := a^y
c := a^(x + (m * x * y))
sig := list{a, b, c}
output := sig
END :: func:sign

BEGIN :: func:verify
input := list{pk, g, M, sig}
pk := expand{X, Y}
sig := expand{a, b, c}
m := H(M, ZR)
BEGIN :: if
if {{ e(Y, a) == e(g, b) } and { (e(X, a) * (e(X, b)^m)) == e(g, c) }}
output := True
else
output := False
END :: if
END :: func:verify

# Batch input
BEGIN :: precompute
m := H(M, ZR)
END :: precompute

constant := g
public := pk
signature := sig
message := m

# same signer
BEGIN :: count
message_count := N
public_count := one
signature_count := N
END :: count

verify := { e(Y, a) == e(g, b) } and { (e(X, a) * (e(X, b)^m)) == e(g, c) }

```

Zerocash: Decentralized Anonymous Payments from Bitcoin

Eli Ben-Sasson*, Alessandro Chiesa[†], Christina Garman[‡], Matthew Green[‡], Ian Miers[‡], Eran Tromer[§], Madars Virza[†]

*Technion, eli@cs.technion.ac.il

[†]MIT, {alexch, madars}@mit.edu

[‡]Johns Hopkins University, {cgarman, imiers, mgreen}@cs.jhu.edu

[§]Tel Aviv University, tromer@cs.tau.ac.il

Abstract—Bitcoin is the first digital currency to see widespread adoption. While payments are conducted between pseudonyms, Bitcoin cannot offer strong privacy guarantees: payment transactions are recorded in a public decentralized ledger, from which much information can be deduced. Zerocoin (Miers et al., IEEE S&P 2013) tackles some of these privacy issues by unlinking transactions from the payment’s origin. Yet, it still reveals payments’ destinations and amounts, and is limited in functionality.

In this paper, we construct a full-fledged ledger-based digital currency with strong privacy guarantees. Our results leverage recent advances in zero-knowledge Succinct Non-interactive Arguments of Knowledge (zk-SNARKs).

First, we formulate and construct *decentralized anonymous payment schemes* (DAP schemes). A DAP scheme enables users to directly pay each other privately: the corresponding transaction hides the payment’s origin, destination, and transferred amount. We provide formal definitions and proofs of the construction’s security.

Second, we build Zerocash, a practical instantiation of our DAP scheme construction. In Zerocash, transactions are less than 1kB and take under 6ms to verify — orders of magnitude more efficient than the less-anonymous Zerocoin and competitive with plain Bitcoin.

Keywords: Bitcoin, decentralized electronic cash, zero knowledge

I. INTRODUCTION

Bitcoin is the first digital currency to achieve widespread adoption. The currency owes its rise in part to the fact that, unlike traditional e-cash schemes [1, 2, 3], it requires no trusted parties. Instead of appointing a central bank, Bitcoin leverages a distributed ledger known as the *block chain* to store transactions made between users. Because the block chain is massively replicated by mutually-distrustful peers, the information it contains is public.

While users may employ many identities (or *pseudonyms*) to enhance their privacy, an increasing body of research shows that anyone can *de-anonymize* Bitcoin by using information in the block chain [4, 5, 6], such as the structure of the transaction graph as well as the value and dates of transactions. As a result, Bitcoin fails to offer even a modicum of the privacy provided by traditional payment systems, let alone the robust privacy of anonymous e-cash schemes.

While Bitcoin is not anonymous itself, those with sufficient motivation can obfuscate their transaction history with the help of *mixes* (also known as *laundries* or *tumblers*). A mix allows users to entrust a set of coins to a pool operated by a central

party and then, after some interval, retrieve different coins (with the same total value) from the pool. Yet, mixes suffer from three limitations: (i) the delay to reclaim coins must be large to allow enough coins to be mixed in; (ii) the mix can trace coins; and (iii) the mix may steal coins.¹ For users with “something to hide,” these risks may be acceptable. But typical legitimate users (1) wish to keep their spending habits private from their peers, (2) are risk-averse and do not wish to expend continual effort in protecting their privacy, and (3) are often not sufficiently aware of their compromised privacy.

To protect their *privacy*, users thus need an instant, risk-free, and, most importantly, automatic guarantee that data revealing their spending habits and account balances is not publicly accessible by their neighbors, co-workers, and merchants. Anonymous transactions also guarantee that the market value of a coin is independent of its history, thus ensuring legitimate users’ coins remain *fungible*.²

Zerocoin: a decentralized mix. Miers et al. [8] proposed Zerocoin, which extends Bitcoin to provide strong anonymity guarantees. Like many e-cash protocols (e.g., [2]), Zerocoin employs zero-knowledge proofs to prevent transaction graph analyses. Unlike earlier practical e-cash protocols, however, Zerocoin does not rely on digital signatures to validate coins, nor does it require a central bank to prevent double spending. Instead, Zerocoin authenticates coins by proving, in zero-knowledge, that they belong to a public list of valid coins (which can be maintained on the block chain). Yet, rather than a full-fledged anonymous currency, Zerocoin is a *decentralized mix*, where users may periodically “wash” their bitcoins via the Zerocoin protocol. Routine day-to-day transactions must be conducted via Bitcoin, due to reasons that we now review.

The first reason is performance. Redeeming zerocoins requires double-discrete-logarithm proofs of knowledge, which have size that exceeds 45kB and require 450ms to verify (at the 128-bit security level).³ These proofs must be broadcast

¹CoinJoin [7], an alternative proposal, replaces the central party of a mix with multi-signature transactions that involve many collaborating Bitcoin users. CoinJoin can thus only mix small volumes of coins amongst users who are currently online, is prone to denial-of-service attacks by third parties, and requires effort to find mixing partners.

²While the methods we detail in this paper accomplish this, the same techniques open the door for privacy preserving accountability and oversight (see Section X).

³These published numbers [8] actually use a mix of parameters at both 128-bit and 80-bit security for different components of the construction. The cost is higher if all parameters are instantiated at the 128-bit security level.

through the network, verified by every node, and permanently stored in the ledger. The entailed costs are higher, by orders of magnitude, than those in Bitcoin and can seriously tax a Bitcoin network operating at normal scale.

The second reason is functionality. While Zerocoin constitutes a basic e-cash scheme, it lacks critical features required of full-fledged anonymous payments. First, Zerocoin uses coins of fixed denomination: it does not support payments of exact values, nor does it provide a means to make change following a transaction (i.e., divide coins). Second, Zerocoin has no mechanism for one user to pay another one directly in “zerocoins.” And third, while Zerocoin provides anonymity by unlinking a payment transaction from its origin address, it does not hide the amount or other metadata about transactions occurring on the network.

Our contribution. In this work we address the aforementioned issues via two main contributions.

(1) We introduce the notion of a *decentralized anonymous payment scheme*, which formally captures the functionality and security guarantees of a full-fledged decentralized electronic currency with strong anonymity guarantees. We provide a construction of this primitive and prove its security under specific cryptographic assumptions. The construction leverages recent advances in the area of zero-knowledge proofs. Specifically, it uses *zero-knowledge Succinct Non-interactive ARguments of Knowledge* (zk-SNARKs) [9, 10, 11, 12, 13, 14, 15, 16].

(2) We achieve an implementation of the above primitive, via a system that we call **Zerocash**. Compared to Zerocoin, our system (at 128 bits of security):

- Reduces the size of transactions spending a coin by 97.7%.
- Reduces the spend-transaction verification time by 98.6%.
- Allows for anonymous transactions of variable amounts.
- Hides transaction amounts and the values of coins held by users.
- Allows for payments to be made directly to a user’s fixed address (without user interaction).

To validate our system, we measured its performance and established feasibility by conducting experiments in a test network of 1000 nodes (approximately $\frac{1}{16}$ of the unique IPs in the Bitcoin network and $\frac{1}{3}$ of the nodes reachable at any given time [17]). This inspires confidence that Zerocash can be deployed as a fork of Bitcoin and operate at the same scale. Thus, due to its significantly improved functionality and performance, Zerocash makes it possible to entirely replace traditional Bitcoin payments with anonymous alternatives.

Concurrent work. The idea of using zk-SNARKs in the setting of Bitcoin was first presented by one of the authors at Bitcoin 2013 [18]. In concurrent work, Danezis et al. [19] suggest using zk-SNARKs to reduce proof size and verification time in Zerocoin; see Section IX for a comparison.

A. zk-SNARKs

We now sketch in more technical terms the definition of a zk-SNARK; see Section II for more details. A zk-SNARK is a non-interactive zero-knowledge proof of knowledge that

is *succinct*, i.e., for which proofs are very short and easy to verify. More precisely, let $\mathcal{L}$ be an NP language, and let C be a nondeterministic decision circuit for $\mathcal{L}$ on a given instance size n . A zk-SNARK can be used to prove and verify membership in $\mathcal{L}$, for instances of size n , as follows. After taking C as input, a trusted party conducts a one-time setup phase that results in two public keys: a proving key pk and a verification key vk . The proving key pk enables any (untrusted) prover to produce a proof π attesting to the fact that $x \in \mathcal{L}$, for an instance x (of size n) of his choice. The non-interactive proof π is *zero knowledge* and a *proof of knowledge*. Anyone can use the verification key vk to verify the proof π ; in particular zk-SNARK proofs are publicly verifiable: anyone can verify π , without ever having to interact with the prover that generated π . Succinctness requires that (for a given security level) π has *constant size* and can be verified in time that is linear in $|x|$ (rather than linear in $|C|$).

B. Decentralized anonymous payment schemes

We construct a *decentralized anonymous payment (DAP) scheme*, which is a decentralized e-cash scheme that allows direct anonymous payments of any amount. See Section III for a formal definition. Here, we outline our construction in six incremental steps; the construction details are in Section IV.

Our construction functions on top of any ledger-based base currency, such as Bitcoin. At any given time, a unique valid snapshot of the currency’s *ledger* is available to all users. The ledger is a sequence of *transactions* and is append-only. Transactions include both the underlying currency’s transactions, as well as new transactions introduced by our construction. For concreteness, we focus the discussion below on Bitcoin (though later definitions and constructions are stated abstractly). We assume familiarity with Bitcoin [20] and Zerocoin [8].

Step 1: user anonymity with fixed-value coins. We first describe a simplified construction, in which all coins have the same value of, e.g., 1 BTC. This construction, similar to the Zerocoin protocol, shows how to hide a payment’s origin. In terms of tools, we make use of zk-SNARKs (recalled above) and a commitment scheme. Let COMM denote a statistically-hiding non-interactive commitment scheme (i.e., given randomness r and message m , the commitment is $c := \text{COMM}_r(m)$; subsequently, c is opened by revealing r and m , and one can verify that $\text{COMM}_r(m)$ equals c).

In the simplified construction, a new coin c is minted as follows: a user u samples a random *serial number* sn and a *trapdoor* r , computes a *coin commitment* $cm := \text{COMM}_r(sn)$, and sets $c := (r, sn, cm)$. A corresponding mint transaction tx_{Mint} , containing cm (but not sn or r), is sent to the ledger; tx_{Mint} is appended to the ledger only if u has paid 1 BTC to a backing escrow pool (e.g., the 1 BTC may be paid via plaintext information encoded in tx_{Mint}). Mint transactions are thus certificates of deposit, deriving their value from the backing pool.

Subsequently, letting CMList denote the list of all coin commitments on the ledger, u may spend c by posting a spend

transaction tx_{Spend} that contains (i) the coin's serial number sn ; and (ii) a zk-SNARK proof π of the NP statement "*I know r such that $\text{COMM}_r(\text{sn})$ appears in the list CMList of coin commitments*". Assuming that sn does not already appear on the ledger (as part of a past spend transaction), u can redeem the deposited amount of 1 BTC, which u can either keep for himself, transfer to someone else, or immediately deposit into a new coin. (If sn does already appear on the ledger, this is considered double spending, and the transaction is discarded.)

User anonymity is achieved because the proof π is zero-knowledge: while sn is revealed, no information about r is, and finding which of the numerous commitments in CMList corresponds to a particular spend transaction tx_{Spend} is equivalent to inverting $f(x) := \text{COMM}_x(\text{sn})$, which is assumed to be infeasible. Thus, the origin of the payment is anonymous.

Step 2: compressing the list of coin commitments. In the above NP statement, CMList is specified explicitly as a list of coin commitments. This naive representation severely limits scalability because the time and space complexity of most protocol algorithms (e.g., the proof verification algorithm) grows linearly with CMList. Moreover, coin commitments corresponding to already spent coins cannot be dropped from CMList to reduce costs, since they cannot be identified (due to the same zero-knowledge property that provides anonymity).

As in [3], we rely on a collision-resistant hash function CRH to avoid an explicit representation of CMList. We maintain an efficiently updatable append-only CRH-based Merkle tree $\text{Tree}(\text{CMList})$ over the (growing) list CMList. Letting rt denote the root of $\text{Tree}(\text{CMList})$, it is well-known that updating rt to account for insertion of new leaves can be done with time and space proportional to the tree depth. Hence, the time and space complexity is reduced from linear in the size of CMList to logarithmic. With this in mind, we modify the NP statement to the following one: "*I know r such that $\text{COMM}_r(\text{sn})$ appears as a leaf in a CRH-based Merkle tree whose root is rt* ". Compared with the naive data structure for CMList, this modification increases exponentially the size of CMList which a given zk-SNARK implementation can support (concretely, using trees of depth 64, Zerocash supports 2^{64} coins).

Step 3: extending coins for direct anonymous payments. So far, the coin commitment cm of a coin c is a commitment to the coin's serial number sn . However, this creates a problem when transferring c to another user. Indeed, suppose that a user u_A created c , and u_A sends c to another user u_B . First, since u_A knows sn , the spending of c by u_B is both not anonymous (since u_A sees when c is spent, by recognizing sn) and risky (since u_A could still spend c first). Thus, u_B must immediately spend c and mint a new coin c' to protect himself. Second, if u_A in fact wants to transfer to u_B , e.g., 100 BTC, then doing so is both unwieldy (since it requires 100 transfers) and not anonymous (since the amount of the transfer is leaked). And third, transfers in amounts that are not multiples of 1 BTC (the fixed value of a coin) are not supported. Thus, the simplified construction described is inadequate as a payment scheme.

We address this by modifying the derivation of a coin

commitment, and using pseudorandom functions to target payments and to derive serial numbers, as follows. We use three pseudorandom functions (derived from a single one). For a seed x these are denoted $\text{PRF}_x^{\text{addr}}(\cdot)$, $\text{PRF}_x^{\text{sn}}(\cdot)$, and $\text{PRF}_x^{\text{pk}}(\cdot)$. We assume that PRF^{sn} is moreover collision-resistant.

To provide targets for payments, we use *addresses*: each user u generates an address key pair $(a_{\text{pk}}, a_{\text{sk}})$. The coins of u contain the value a_{pk} and can be spent only with knowledge of a_{sk} . A key pair $(a_{\text{pk}}, a_{\text{sk}})$ is sampled by selecting a random seed a_{sk} and setting $a_{\text{pk}} := \text{PRF}_{a_{\text{sk}}}^{\text{addr}}(0)$. A user can generate and use any number of address key pairs.

Next, we re-design minting to allow for greater functionality. To mint a coin c of a desired value v , the user u first samples ρ , which is a secret value that determines the coin's serial number as $\text{sn} := \text{PRF}_{a_{\text{sk}}}^{\text{sn}}(\rho)$. Then, u commits to the tuple (a_{pk}, v, ρ) in two phases: (a) u computes $k := \text{COMM}_r(a_{\text{pk}} \parallel \rho)$ for a random r ; and then (b) u computes $\text{cm} := \text{COMM}_s(v \parallel k)$ for a random s . The minting results in a coin $c := (a_{\text{pk}}, v, \rho, r, s, \text{cm})$ and a mint transaction $\text{tx}_{\text{Mint}} := (v, k, s, \text{cm})$. Crucially, due to the nested commitment, anyone can verify that cm in tx_{Mint} is a coin commitment of a coin of value v (by checking that $\text{COMM}_s(v \parallel k)$ equals cm) but cannot discern the owner (by learning the address key a_{pk}) or serial number (derived from ρ) because these are hidden in k . As before, tx_{Mint} is accepted by the ledger only if u deposits the correct amount, in this case v BTC.

Coins are spent using the *pour* operation, which takes a set of input coins, to be consumed, and "pours" their value into a set of fresh output coins — such that the total value of output coins equals the total value of the input coins. Suppose that u , with address key pair $(a_{\text{pk}}^{\text{old}}, a_{\text{sk}}^{\text{old}})$, wishes to consume his coin $c^{\text{old}} = (a_{\text{pk}}^{\text{old}}, v^{\text{old}}, \rho^{\text{old}}, r^{\text{old}}, s^{\text{old}}, \text{cm}^{\text{old}})$ and produce two new coins c_1^{new} and c_2^{new} , with total value $v_1^{\text{new}} + v_2^{\text{new}} = v^{\text{old}}$, respectively targeted at address public keys $a_{\text{pk},1}^{\text{new}}$ and $a_{\text{pk},2}^{\text{new}}$. (The addresses $a_{\text{pk},1}^{\text{new}}$ and $a_{\text{pk},2}^{\text{new}}$ may belong to u or to some other user.) The user u , for each $i \in \{1, 2\}$, proceeds as follows: (i) u samples serial number randomness ρ_i^{new} ; (ii) u computes $k_i^{\text{new}} := \text{COMM}_{r_i^{\text{new}}}(a_{\text{pk},i}^{\text{new}} \parallel \rho_i^{\text{new}})$ for a random r_i^{new} ; and (iii) u computes $\text{cm}_i^{\text{new}} := \text{COMM}_{s_i^{\text{new}}}(v_i^{\text{new}} \parallel k_i^{\text{new}})$ for a random s_i^{new} .

This yields the coins $c_1^{\text{new}} := (a_{\text{pk},1}^{\text{new}}, v_1^{\text{new}}, \rho_1^{\text{new}}, r_1^{\text{new}}, s_1^{\text{new}}, \text{cm}_1^{\text{new}})$ and $c_2^{\text{new}} := (a_{\text{pk},2}^{\text{new}}, v_2^{\text{new}}, \rho_2^{\text{new}}, r_2^{\text{new}}, s_2^{\text{new}}, \text{cm}_2^{\text{new}})$. Next, u produces a zk-SNARK proof π_{POUR} for the following NP statement, which we call **POUR**:

"Given the Merkle-tree root rt , serial number sn^{old} , and coin commitments $\text{cm}_1^{\text{new}}, \text{cm}_2^{\text{new}}$, I know coins $c^{\text{old}}, c_1^{\text{new}}, c_2^{\text{new}}$, and address secret key $a_{\text{sk}}^{\text{old}}$ such that:

- The coins are well-formed: for c^{old} it holds that $k^{\text{old}} = \text{COMM}_{r^{\text{old}}}(a_{\text{pk}}^{\text{old}} \parallel \rho^{\text{old}})$ and $\text{cm}^{\text{old}} = \text{COMM}_{s^{\text{old}}}(v^{\text{old}} \parallel k^{\text{old}})$; and similarly for c_1^{new} and c_2^{new} .
- The address secret key matches the public key: $a_{\text{pk}}^{\text{old}} = \text{PRF}_{a_{\text{sk}}^{\text{old}}}^{\text{addr}}(0)$.
- The serial number is computed correctly: $\text{sn}^{\text{old}} := \text{PRF}_{a_{\text{sk}}^{\text{old}}}^{\text{sn}}(\rho^{\text{old}})$.
- The coin commitment cm^{old} appears as a leaf of a Merkle-

tree with root rt .

- The values add up: $v_1^{\text{new}} + v_2^{\text{new}} = v^{\text{old}}$.

A resulting pour transaction $\text{tx}_{\text{Pour}} := (rt, \text{sn}^{\text{old}}, \text{cm}_1^{\text{new}}, \text{cm}_2^{\text{new}}, \pi_{\text{Pour}})$ is appended to the ledger. (As before, the transaction is rejected if the serial number sn appears in a previous transaction.)

Now suppose that u does not know, say, the address secret key $a_{\text{sk},1}^{\text{new}}$ that is associated with the public key $a_{\text{pk},1}^{\text{new}}$. Then, u cannot spend c_1^{new} because he cannot provide $a_{\text{sk},1}^{\text{new}}$ as part of the witness of a subsequent pour operation. Furthermore, when a user that knows $a_{\text{sk},1}^{\text{new}}$ does spend c_1^{new} , the user u cannot track it, because he knows no information about its revealed serial number, which is $\text{sn}_1^{\text{new}} := \text{PRF}_{a_{\text{sk},1}^{\text{new}}}^{\text{sn}}(\rho_1^{\text{new}})$.

Also observe that tx_{Pour} reveals no information about how the value of the consumed coin was divided among the two new fresh coins, nor which coin commitment corresponds to the consumed coin, nor the address public keys to which the two new fresh coins are targeted. The payment was conducted in full anonymity.

More generally, a user may pour $N^{\text{old}} \geq 0$ coins into $N^{\text{new}} \geq 0$ coins. For simplicity we consider the case $N^{\text{old}} = N^{\text{new}} = 2$, without loss of generality. Indeed, for $N^{\text{old}} < 2$, the user can mint a coin with value 0 and then provide it as a “null” input, and for $N^{\text{new}} < 2$, the user can create (and discard) a new coin with value 0. For $N^{\text{old}} > 2$ or $N^{\text{new}} > 2$, the user can compose $\log N^{\text{old}} + \log N^{\text{new}}$ of the 2-input/2-output pours.

Step 4: sending coins. Suppose that $a_{\text{pk},1}^{\text{new}}$ is the address public key of u_1 . In order to allow u_1 to actually spend the new coin c_1^{new} produced above, u must somehow send the secret values in c_1^{new} to u_1 . One way is for u to send u_1 a private message, but the requisite private communication channel necessitates additional infrastructure or assumptions. We avoid this “out-of-band” channel and instead build this capability directly into our construction by leveraging the ledger as follows.

We modify the structure of an address key pair. Each user now has a key pair $(\text{addr}_{\text{pk}}, \text{addr}_{\text{sk}})$, where $\text{addr}_{\text{pk}} = (a_{\text{pk}}, \text{pk}_{\text{enc}})$ and $\text{addr}_{\text{sk}} = (a_{\text{sk}}, \text{sk}_{\text{enc}})$. The values $(a_{\text{pk}}, a_{\text{sk}})$ are generated as before. In addition, $(\text{pk}_{\text{enc}}, \text{sk}_{\text{enc}})$ is a key pair for a *key-private encryption scheme* [21].

Then, u computes the ciphertext C_1 that is the encryption of the plaintext $(v_1^{\text{new}}, \rho_1^{\text{new}}, r_1^{\text{new}}, s_1^{\text{new}})$, under $\text{pk}_{\text{enc},1}^{\text{new}}$ (which is part of u_1 ’s address public key $\text{addr}_{\text{pk},1}^{\text{new}}$), and includes C_1 in the pour transaction tx_{Pour} . The user u_1 can then find and decrypt this message (using his $\text{sk}_{\text{enc},1}^{\text{new}}$) by scanning the pour transactions on the public ledger. Again, note that adding C_1 to tx_{Pour} leaks neither paid amounts, nor target addresses due to the key-private property of the encryption scheme. (The user u does the same with c_2^{new} and includes a corresponding ciphertext C_2 in tx_{Pour} .)

Step 5: public outputs. The construction so far allows users to mint, merge, and split coins. But how can a user redeem one of his coins, i.e., convert it back to the base currency (Bitcoin)? For this, we modify the pour operation to include a *public output*. When spending a coin, the user u also specifies a nonnegative v_{pub} and an arbitrary string info . The balance

equation in the NP statement POUR is changed accordingly: “ $v_1^{\text{new}} + v_2^{\text{new}} + v_{\text{pub}} = v^{\text{old}}$ ”. Thus, of the input value v^{old} , a part v_{pub} is publicly declared, and its target is specified, somehow, by the string info . The string info can be used to specify the destination of these redeemed funds (e.g., a Bitcoin wallet public key).⁴ Both v_{pub} and info are now included in the resulting pour transaction tx_{Pour} . (The public output is optional, as the user u can set $v_{\text{pub}} = 0$.)

Step 6: non-malleability. To prevent malleability attacks on a pour transaction tx_{Pour} (e.g., embezzlement by re-targeting the public output of the pour by modifying info), we further modify the NP statement POUR and use digital signatures. Specifically, during the pour operation, the user u (i) samples a key pair $(\text{pk}_{\text{sig}}, \text{sk}_{\text{sig}})$ for a one-time signature scheme; (ii) computes $h_{\text{sig}} := \text{CRH}(\text{pk}_{\text{sig}})$; (iii) computes the two values $h_1 := \text{PRF}_{a_{\text{sk},1}^{\text{old}}}^{\text{pk}}(h_{\text{sig}})$ and $h_2 := \text{PRF}_{a_{\text{sk},2}^{\text{old}}}^{\text{pk}}(h_{\text{sig}})$, which act as MACs to “tie” h_{sig} to both address secret keys; (iv) modifies POUR to include the three values h_{sig}, h_1, h_2 and prove that the latter two are computed correctly; and (v) uses sk_{sig} to sign every value associated with the pour operation, thus obtaining a signature σ , which is included, along with pk_{sig} , in tx_{Pour} . Since the $a_{\text{sk},i}^{\text{old}}$ are secret, and with high probability h_{sig} changes for each pour transaction, the values h_1, h_2 are unpredictable. Moreover, the signature on the NP statement (and other values) binds all of these together.

This ends the outline of the construction, which is summarized in part in Figure 1. We conclude by noting that, due to the zk-SNARK, our construction requires a one-time trusted setup of public parameters. The trust affects soundness of the proofs, though anonymity continues to hold even if the setup is corrupted by a malicious party.

C. Zerocash

We outline Zerocash, a concrete implementation, at 128 bits of security, of our DAP scheme construction; see Section V for details. Zerocash entails carefully instantiating the cryptographic ingredients of the construction to ensure that the zk-SNARK, the “heaviest” component, is efficient enough in practice. In the construction, the zk-SNARK is used to prove/verify a specific NP statement: POUR . While zk-SNARKs are asymptotically efficient, their concrete efficiency depends on the arithmetic circuit C that is used to decide the NP statement. Thus, we seek instantiations for which we can design a relatively-small arithmetic circuit C_{Pour} for verifying the NP statement POUR .

Our approach is to instantiate all of the necessary cryptographic ingredients (commitment schemes, pseudorandom functions, and collision-resistant hashing) based on SHA256. We first design a hand-optimized circuit for verifying SHA256 computations (or, more precisely, its compression function,

⁴These public outputs can be considered as an “input” to a Bitcoin-style transaction, where the info string contains the Bitcoin output scripts. This mechanism also allows us to support Bitcoin’s public transaction fees.

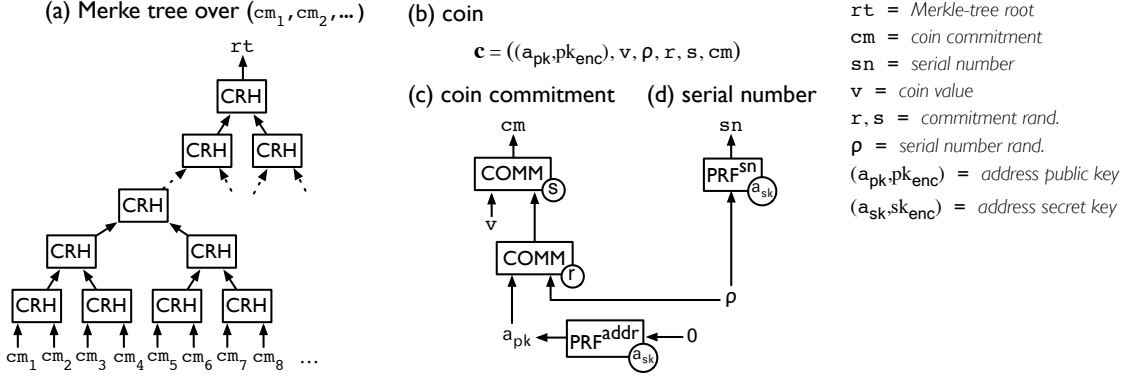


Fig. 1: (a) Illustration of the CRH-based Merkle tree over the list CMList of coin commitments. (b) A coin c . (c) Illustration of the structure of a coin commitment cm . (d) Illustration of the structure of a coin serial number sn .

which suffices for our purposes).⁵ Then, we use this circuit in constructing C_{POUR} , which verifies all the necessary checks for satisfying the NP statement C_{POUR} .

This, along with judicious parameter choices, and a state-of-the-art implementation of a zk-SNARK for arithmetic circuits [16] (see Section II-C), results in a zk-SNARK prover running time of few minutes and zk-SNARK verifier running time of few milliseconds. This allows the DAP scheme implementation to be practical for deployment, as our experiments show.

Zerocash can be integrated into Bitcoin or forks of it (commonly referred to as “altcoins”); we later describe how this is done.

D. Paper organization

The remainder of this paper is organized as follows. Section II provides background on zk-SNARKs. We define DAP schemes in Section III, and our construction thereof in Section IV. Section V discusses the concrete instantiation in Zerocash. Section VI describes the integration of Zerocash into existing ledger-based currencies. Section VII provides microbenchmarks for our prototype implementation, as well as results based on full-network simulations. Section VIII describes optimizations. We discuss concurrent work in Section IX and summarize our contributions and future directions in Section X.

II. BACKGROUND ON ZK-SNARKS

The main cryptographic primitive used in this paper is a special kind of *Succinct Non-interactive Argument of Knowledge* (SNARK). Concretely, we use a *publicly-verifiable preprocessing zero-knowledge SNARK*, or zk-SNARK for short. In this section we provide basic background on zk-SNARKs, provide an informal definition, and recall known constructions and implementations.

⁵Alternatively, we could have opted to rely on the circuit generators [13, 14, 16], which support various classes of C programs, by writing C code expressing the POUR checks. However, as discussed later, these generic approaches are more expensive than our hand-optimized construction.

A. Informal definition

We informally define zk-SNARKs for arithmetic circuit satisfiability. We refer the reader to, e.g., [11] for a formal definition.

For a field $\mathbb{F}$, an $\mathbb{F}$ -arithmetic circuit takes inputs that are elements in $\mathbb{F}$, and its gates output elements in $\mathbb{F}$. We naturally associate a circuit with the function it computes. To model nondeterminism we consider circuits that have an *input* $x \in \mathbb{F}^n$ and an auxiliary input $a \in \mathbb{F}^h$, called a *witness*. The circuits we consider only have *bilinear gates*.⁶ Arithmetic circuit satisfiability is defined analogously to the boolean case, as follows.

Definition II.1. The *arithmetic circuit satisfiability problem* of an $\mathbb{F}$ -arithmetic circuit $C: \mathbb{F}^n \times \mathbb{F}^h \rightarrow \mathbb{F}^l$ is captured by the relation $\mathcal{R}_C = \{(x, a) \in \mathbb{F}^n \times \mathbb{F}^h : C(x, a) = 0^l\}$; its language is $\mathcal{L}_C = \{x \in \mathbb{F}^n : \exists a \in \mathbb{F}^h \text{ s.t. } C(x, a) = 0^l\}$.

Given a field $\mathbb{F}$, a (publicly-verifiable preprocessing) **zk-SNARK** for $\mathbb{F}$ -arithmetic circuit satisfiability is a triple of polynomial-time algorithms (KeyGen, Prove, Verify):

- **KeyGen**($1^\lambda, C$) $\rightarrow$ (pk, vk). On input a security parameter λ (presented in unary) and an $\mathbb{F}$ -arithmetic circuit C , the *key generator* KeyGen probabilistically samples a *proving key* pk and a *verification key* vk. Both keys are published as public parameters and can be used, any number of times, to prove/verify membership in $\mathcal{L}_C$.
- **Prove**(pk, x, a) $\rightarrow$ π . On input a proving key pk and any $(x, a) \in \mathcal{R}_C$, the *prover* Prove outputs a non-interactive proof π for the statement $x \in \mathcal{L}_C$.
- **Verify**(vk, x, π) $\rightarrow$ b . On input a verification key vk, an input x , and a proof π , the *verifier* Verify outputs $b = 1$ if he is convinced that $x \in \mathcal{L}_C$.

A zk-SNARK satisfies the following properties.

Completeness. For every security parameter λ , any $\mathbb{F}$ -arithmetic circuit C , and any $(x, a) \in \mathcal{R}_C$, the honest prover

⁶A gate with inputs $y_1, \dots, y_m \in \mathbb{F}$ is *bilinear* if the output is $\langle \vec{a}, (1, y_1, \dots, y_m) \rangle \cdot \langle \vec{b}, (1, y_1, \dots, y_m) \rangle$ for some $\vec{a}, \vec{b} \in \mathbb{F}^{m+1}$. These include addition, multiplication, negation, and constant gates.

can convince the verifier. Namely, $b = 1$ with probability $1 - \text{negl}(\lambda)$ in the following experiment: $(\text{pk}, \text{vk}) \leftarrow \text{KeyGen}(1^\lambda, C)$; $\pi \leftarrow \text{Prove}(\text{pk}, x, a)$; $b \leftarrow \text{Verify}(\text{vk}, x, \pi)$.

Succinctness. An honestly-generated proof π has $O_\lambda(1)$ bits and $\text{Verify}(\text{vk}, x, \pi)$ runs in time $O_\lambda(|x|)$. (Here, O_λ hides a fixed polynomial factor in λ .)

Proof of knowledge (and soundness). If the verifier accepts a proof output by a bounded prover, then the prover “knows” a witness for the given instance. (In particular, soundness holds against bounded provers.) Namely, for every $\text{poly}(\lambda)$ -size adversary $\mathcal{A}$, there is a $\text{poly}(\lambda)$ -size extractor $\mathcal{E}$ such that $\text{Verify}(\text{vk}, x, \pi) = 1$ and $(x, a) \notin \mathcal{R}_C$ with probability $\text{negl}(\lambda)$ in the following experiment: $(\text{pk}, \text{vk}) \leftarrow \text{KeyGen}(1^\lambda, C)$; $(x, \pi) \leftarrow \mathcal{A}(\text{pk}, \text{vk})$; $a \leftarrow \mathcal{E}(\text{pk}, \text{vk})$.

Perfect zero knowledge. An honestly-generated proof is perfect zero knowledge.⁷ Namely, there is a $\text{poly}(\lambda)$ -size simulator Sim such that for all stateful $\text{poly}(\lambda)$ -size distinguishers $\mathcal{D}$ the following two probabilities are equal:

- The probability that $\mathcal{D}(\pi) = 1$ on an honest proof.

$$\Pr \left[\begin{array}{c} (x, a) \in \mathcal{R}_C \\ \mathcal{D}(\pi) = 1 \end{array} \middle| \begin{array}{c} (\text{pk}, \text{vk}) \leftarrow \text{KeyGen}(C) \\ (x, a) \leftarrow \mathcal{D}(\text{pk}, \text{vk}) \\ \pi \leftarrow \text{Prove}(\text{pk}, x, a) \end{array} \right]$$

- The probability that $\mathcal{D}(\pi) = 1$ on a simulated proof.

$$\Pr \left[\begin{array}{c} (x, a) \in \mathcal{R}_C \\ \mathcal{D}(\pi) = 1 \end{array} \middle| \begin{array}{c} (\text{pk}, \text{vk}, \text{trap}) \leftarrow \text{Sim}(C) \\ (x, a) \leftarrow \mathcal{D}(\text{pk}, \text{vk}) \\ \pi \leftarrow \text{Sim}(\text{pk}, x, \text{trap}) \end{array} \right]$$

B. Known constructions and security

There are many zk-SNARK constructions in the literature [9, 10, 11, 12, 13, 14, 15, 16]. We are interested in zk-SNARKs for arithmetic circuit satisfiability, and the most efficient ones for this language are based on *quadratic arithmetic programs* [12, 11, 13, 14, 16]; such constructions provide a linear-time KeyGen , quasilinear-time Prove , and linear-time Verify .

Security of zk-SNARKs is based on knowledge-of-exponent assumptions and variants of Diffie–Hellman assumptions in bilinear groups [9, 22, 23]. While knowledge-of-exponent assumptions are fairly strong, there is evidence that such assumptions may be inherent for constructing zk-SNARKs [24, 25].

C. zk-SNARK implementations

There are three published implementations of zk-SNARKs: (i) Parno et al. [13] present an implementation of zk-SNARKs for programs having no data dependencies;⁸ (ii) Ben-Sasson et al. [14] present an implementation of zk-SNARKs for arbitrary programs (with data dependencies); and (iii) Ben-Sasson et al. [16] present an implementation of zk-SNARKs

⁷While most zk-SNARK descriptions in the literature only mention statistical zero knowledge, all zk-SNARK constructions can be made perfect zero knowledge by allowing for a negligible error probability in completeness.

⁸They only support programs where array indices are restricted to be known compile-time constants; similarly, loop iteration counts (or at least upper bounds to these) must be known at compile time.

that supports programs that modify their own code (e.g., for runtime code generation); their implementation also reduces costs for programs of larger size and allows for universal key pairs.

Each of the works above also achieves zk-SNARKs for arithmetic circuit satisfiability as a stepping stone towards their respective higher-level efforts. In this paper we are only interested in a zk-SNARK for arithmetic circuit satisfiability, and we rely on the implementation of [16] for such a zk-SNARK.⁹ The implementation in [16] is itself based on the protocol of Parno et al. [13]. We thus refer the interested reader to [13] for details of the protocol, its intuition, and its proof of security; and to [16] for the implementation and its performance. In terms of concrete parameters, the implementation of [16] provides 128 bits of security, and the field $\mathbb{F}$ is of a 256-bit prime order p .

III. DEFINITION OF A DECENTRALIZED ANONYMOUS PAYMENT SCHEME

We introduce the notion of a *decentralized anonymous payment scheme* (DAP scheme), extending the notion of *decentralized e-cash* [8]. Later, in Section IV, we provide a construction.

A. Data structures

We begin by describing, and giving intuition about, the data structures used by a DAP scheme. The algorithms that use and produce these data structures are introduced in Section III-B.

Basecoin ledger. Our protocol is applied on top of a ledger-based base currency such as Bitcoin; for generality we refer to this base currency as *Basecoin*. At any given time T , all users have access to L_T , the *ledger* at time T , which is a sequence of *transactions*. The ledger is append-only (i.e., $T < T'$ implies that L_T is a prefix of $L_{T'}$).¹⁰ The transactions in the ledger include both Basecoin transactions as well as two new transaction types described below.

Public parameters. A list of *public parameters* pp is available to all users in the system. These are generated by a trusted party at the “start of time” and are used by the system’s algorithms.

Addresses. Each user generates at least one *address key pair* $(\text{addr}_{\text{pk}}, \text{addr}_{\text{sk}})$. The public key addr_{pk} is published and enables others to direct payments to the user. The secret key addr_{sk} is used to receive payments sent to addr_{pk} . A user may generate any number of address key pairs.

Coins. A *coin* is a data object c , to which we associate the following:

- A *coin commitment*, denoted $\text{cm}(\text{c})$: a string that appears on the ledger once c is *minted*.

⁹In [16], one optimization to the verifier’s runtime requires preprocessing the verification key vk ; for simplicity, we do not use this optimization.

¹⁰In reality, the Basecoin ledger (such as the one of Bitcoin) is not perfect and may incur temporary inconsistencies. In this respect our construction is as good as the underlying ledger. We discuss the effects of this on anonymity and mitigations in Section VI-C.

- A *coin value*, denoted $v(c)$: the denomination of c , as measured in basecoins, as an integer between 0 and a maximum value $v_{\max}$ (which is a system parameter).
- A *coin serial number*, denoted $sn(c)$: a unique string associated with the c , used to prevent double spending.
- A *coin address*, denoted $addr_{pk}(c)$: an address public key, representing who owns c .

Any other quantities associated with a coin c (e.g., various trapdoors) are implementation details.

New transactions. Besides Basecoin transactions, there are two new types of transactions.

- *Mint transactions.* A mint transaction tx_{Mint} is a tuple $(cm, v, *)$, where cm is a coin commitment, v is a coin value, and $*$ denotes other (implementation-dependent) information. The transaction tx_{Mint} records that a coin c with coin commitment cm and value v has been minted.
- *Pour transactions.* A pour transaction tx_{Pour} is a tuple $(rt, sn_1^{old}, sn_2^{old}, cm_1^{new}, cm_2^{new}, v_{pub}, info, *)$, where rt is a root of a Merkle tree, sn_1^{old}, sn_2^{old} are two coin serial numbers, cm_1^{new}, cm_2^{new} are two coin commitments, v_{pub} is a coin value, $info$ is an arbitrary string, and $*$ denotes other (implementation-dependent) information. The transaction tx_{Pour} records the pouring of two input (and now consumed) coins c_1^{old}, c_2^{old} , with respective serial numbers sn_1^{old}, sn_2^{old} , into two new output coins c_1^{new}, c_2^{new} , with respective coin commitments cm_1^{new}, cm_2^{new} , as well as a public output v_{pub} (which may be zero). Furthermore, tx_{Pour} also records an information string $info$ (perhaps containing information on who is the recipient of v_{pub} basecoins) and that, when this transaction was made, the root of the Merkle tree over coin commitments was rt (see below).

Commitments of minted coins and serial numbers of spent coins. For any given time T ,

- $CMList_T$ denotes the list of all coin commitments appearing in mint and pour transactions in L_T ;
- $SNList_T$ denotes the list of all serial numbers appearing in pour transactions in L_T .

While both of these lists can be deduced from L_T , it will be convenient to think about them as separate (as, in practice, these may be separately maintained due to efficiency reasons).

Merkle tree over commitments. For any given time T , $Tree_T$ denotes a Merkle tree over $CMList_T$ and rt_T its root. Moreover, the function $Path_T(cm)$ gives the authentication path from a coin commitment cm appearing in $CMList_T$ to the root of $Tree_T$.¹¹ For convenience, we assume that L_T also stores $rt_{T'}$ for all $T' \leq T$ (i.e., it stores all past Merkle tree roots).

B. Algorithms

A DAP scheme Π is a tuple of polynomial-time algorithms (Setup, CreateAddress, Mint, Pour, VerifyTransaction, Receive)

¹¹While we refer to Merkle trees for simplicity, it is straightforward to extend the definition to allow other data structures representing sets with fast insertion and short proofs of membership.

with the following syntax and semantics.

System setup. The algorithm Setup generates a list of public parameters:

- Setup
- INPUTS: security parameter λ
 - OUTPUTS: public parameters pp

The algorithm Setup is executed by a trusted party. The resulting public parameters pp are published and made available to all parties (e.g., by embedding them into the protocol's implementation). The setup is done *only once*; afterwards, no trusted party is needed, and no global secrets or trapdoors are kept.

Creating payment addresses. The algorithm CreateAddress generates a new address key pair:

- CreateAddress
- INPUTS: public parameters pp
 - OUTPUTS: address key pair $(addr_{pk}, addr_{sk})$

Each user generates at least one address key pair $(addr_{pk}, addr_{sk})$ in order to receive coins. The public key $addr_{pk}$ is published, while the secret key $addr_{sk}$ is used to redeem coins sent to $addr_{pk}$. A user may generate any number of address key pairs; doing so does not require any interaction.

Minting coins. The algorithm Mint generates a coin (of a given value) and a mint transaction:

- Mint
- INPUTS:
 - public parameters pp
 - coin value $v \in \{0, 1, \dots, v_{\max}\}$
 - destination address public key $addr_{pk}$
 - OUTPUTS: coin c and mint transaction tx_{Mint}

A system parameter, $v_{\max}$, caps the value of any single coin. The output coin c has value v and coin address $addr_{pk}$; the output mint transaction tx_{Mint} equals $(cm, v, *)$, where cm is the coin commitment of c .

Pouring coins. The Pour algorithm transfers value from input coins into new output coins, marking the input coins as consumed. Moreover, a fraction of the input value may be publicly revealed. Pouring allows users to subdivide coins into smaller denominations, merge coins, and transfer ownership of anonymous coins, or make public payments.¹²

- Pour
- INPUTS:
 - public parameters pp
 - the Merkle root rt
 - old coins c_1^{old}, c_2^{old}
 - old addresses secret keys $addr_{sk,1}^{old}, addr_{sk,2}^{old}$
 - authentication path $path_1$ from commitment $cm(c_1^{old})$ to root rt ,

¹²We consider pours with 2 inputs and 2 outputs, for simplicity and (as discussed in Section I-B) without loss of generality.

- authentication path path_2 from commitment $\text{cm}(c_2^{\text{old}})$ to root rt
- new values $v_1^{\text{new}}, v_2^{\text{new}}$
- new addresses public keys $\text{addr}_{\text{pk},1}^{\text{new}}, \text{addr}_{\text{pk},2}^{\text{new}}$
- public value v_{pub}
- transaction string info
- OUTPUTS: new coins $c_1^{\text{new}}, c_2^{\text{new}}$ and pour transaction tx_{Pour}

Thus, the Pour algorithm takes as input two distinct input coins $c_1^{\text{old}}, c_2^{\text{old}}$, along with corresponding address secret keys $\text{addr}_{\text{sk},1}^{\text{old}}, \text{addr}_{\text{sk},2}^{\text{old}}$ (required to redeem the two input coins). To ensure that $c_1^{\text{old}}, c_2^{\text{old}}$ have been previously minted, the Pour algorithm also takes as input the Merkle root rt (allegedly, equal to the root of Merkle tree over all coin commitments so far), along with two authentication paths $\text{path}_1, \text{path}_2$ for the two coin commitments $\text{cm}(c_1^{\text{old}}), \text{cm}(c_2^{\text{old}})$. Two input values $v_1^{\text{new}}, v_2^{\text{new}}$ specify the values of two new anonymous coins $c_1^{\text{new}}, c_2^{\text{new}}$ to be generated, and two input address public keys $\text{addr}_{\text{pk},1}^{\text{new}}, \text{addr}_{\text{pk},2}^{\text{new}}$ specify the recipients of $c_1^{\text{new}}, c_2^{\text{new}}$. A third value, v_{pub} , specifies the amount to be publicly spent (e.g., to redeem coins or pay transaction fees). The sum of output values $v_1 + v_2 + v_{\text{pub}}$ must be equal to the sum of the values of the input coins (and cannot exceed v_{max}). Finally, the Pour algorithm also receives an arbitrary string info, which is bound into the output pour transaction tx_{Pour} .

The Pour algorithm outputs two new coins $c_1^{\text{new}}, c_2^{\text{new}}$ and a pour transaction tx_{Pour} . The transaction tx_{Pour} equals $(\text{rt}, \text{sn}_1^{\text{old}}, \text{sn}_2^{\text{old}}, \text{cm}_1^{\text{new}}, \text{cm}_2^{\text{new}}, v_{\text{pub}}, \text{info}, *)$, where $\text{cm}_1^{\text{new}}, \text{cm}_2^{\text{new}}$ are the two coin commitments of the two output coins, and $*$ denotes other (implementation-dependent) information. Crucially, tx_{Pour} reveals only one currency value, the public value v_{pub} (which may be zero); it does not reveal the payment addresses or values of the old or new coins.

Verifying transactions. The algorithm VerifyTransaction checks the validity of a transaction:

- VerifyTransaction
- INPUTS:
 - public parameters pp
 - a (mint or pour) transaction tx
 - the current ledger L
- OUTPUTS: bit b , equals 1 iff the transaction is valid

Both mint and pour transactions must be verified before being considered well-formed. In practice, transactions can be verified by the nodes in the distributed system maintaining the ledger, as well as by users who rely on these transactions.

Receiving coins. The algorithm Receive scans the ledger and retrieves unspent coins paid to a particular user address:

- Receive
- INPUTS:
 - recipient address key pair $(\text{addr}_{\text{pk}}, \text{addr}_{\text{sk}})$
 - the current ledger L
- OUTPUTS: set of (unspent) received coins

When a user with address key pair $(\text{addr}_{\text{pk}}, \text{addr}_{\text{sk}})$ wishes to receive payments sent to addr_{pk} , he uses the Receive algorithm to scan the ledger. For each payment to addr_{pk} appearing in the ledger, Receive outputs the corresponding coins whose serial numbers do not appear on the ledger L . Coins received in this way may be spent, just like minted coins, using the Pour algorithm. (We only require Receive to detect coins paid to addr_{pk} via the Pour algorithm and not also detect coins minted by the user himself.)

Next, we describe completeness (Section III-C) and security (Section III-D).

C. Completeness

Completeness of a DAP scheme requires that unspent coins can be spent. More precisely, consider a *ledger sampler* $\mathcal{S}$ outputting a ledger L . If c_1 and c_2 are two coins whose coin commitments appear in (valid) transactions on L , but their serial numbers do not appear in L , then c_1 and c_2 can be spent using Pour. Namely, running Pour results in a pour transaction tx_{Pour} that VerifyTransaction accepts, and the new coins can be received by the intended recipients (by using Receive); moreover, tx_{Pour} correctly records the intended v_{pub} and transaction string info. This property is formalized via an *incompleteness experiment* **INCOMP**.

Definition III.1. A DAP scheme $\Pi = (\text{Setup}, \text{CreateAddress}, \text{Mint}, \text{Pour}, \text{VerifyTransaction}, \text{Receive})$ is **complete** if no polynomial-size ledger sampler $\mathcal{S}$ wins **INCOMP** with more than negligible probability.

D. Security

Security of a DAP scheme is characterized by three properties, which we call *ledger indistinguishability*, *transaction non-malleability*, and *balance*.

Definition III.2. A DAP scheme $\Pi = (\text{Setup}, \text{CreateAddress}, \text{Mint}, \text{Pour}, \text{VerifyTransaction}, \text{Receive})$ is **secure** if it satisfies ledger indistinguishability, transaction non-malleability, and balance.

Below, we provide an informal overview of each property, and defer formal definitions to the extended version of this paper [26].

Each property is formalized as a game between an adversary $\mathcal{A}$ and a challenger $\mathcal{C}$. In each game, the behavior of honest parties is realized via a DAP scheme oracle $\mathcal{O}^{\text{DAP}}$, which maintains a ledger L and provides an interface for executing CreateAddress , Mint , Pour and Receive algorithms for honest parties. To elicit behavior from honest parties, $\mathcal{A}$ passes a query to $\mathcal{C}$, which (after sanity checks) proxies the query to $\mathcal{O}^{\text{DAP}}$. For each query that requests an honest party to perform an action, $\mathcal{A}$ specifies identities of previous transactions and the input values, and learns the resulting transaction, but not any of the secrets or trapdoors involved in producing that transaction. The oracle $\mathcal{O}^{\text{DAP}}$ also provides an **Insert** query that allows $\mathcal{A}$ to directly add arbitrary transactions to the ledger L .

Ledger indistinguishability. This property captures the requirement that the ledger reveals no new information to the adversary beyond the publicly-revealed information (values of minted coins, public values, information strings, total number of transactions, etc.), even when the adversary can adaptively induce honest parties to perform DAP operations of his choice. That is, no bounded adversary $\mathcal{A}$ can distinguish between two ledgers L_0 and L_1 , constructed by $\mathcal{A}$ using queries to two DAP scheme oracles, when the queries to the two oracles are *publicly consistent*: they have matching type and are identical in terms of publicly-revealed information and the information related to addresses controlled by $\mathcal{A}$.

Ledger indistinguishability is formalized by an experiment L-IND that proceeds as follows. First, a challenger samples a random bit b and initializes two DAP scheme oracles $\mathcal{O}_0^{\text{DAP}}$ and $\mathcal{O}_1^{\text{DAP}}$, maintaining ledgers L_0 and L_1 . Throughout, the challenger allows $\mathcal{A}$ to issue queries to $\mathcal{O}_0^{\text{DAP}}$ and $\mathcal{O}_1^{\text{DAP}}$, thus controlling the behavior of honest parties on L_0 and L_1 . The challenger provides the adversary with the view of both ledgers, but in randomized order: $L_{\text{Left}} := L_b$ and $L_{\text{Right}} := L_{1-b}$. The adversary's goal is to distinguish whether the view he sees corresponds to $(L_{\text{Left}}, L_{\text{Right}}) = (L_0, L_1)$, i.e. $b = 0$, or to $(L_{\text{Left}}, L_{\text{Right}}) = (L_1, L_0)$, i.e. $b = 1$.

At each round of the experiment, the adversary issues queries in pairs Q, Q' of matching query type. If the query type is **CreateAddress**, then the same address is generated at both oracles. If it is **Mint**, **Pour** or **Receive**, then Q is forwarded to L_0 and Q' to L_1 ; for **Insert** queries, query Q is forwarded to L_{Left} and Q' is forwarded to L_{Right} . The adversary's queries are restricted in the sense that they must maintain the *public consistency* of the two ledgers. For example, the public values for **Pour** queries must be the same, as well as minted amounts for **Mint** queries.

At the conclusion of the experiment, $\mathcal{A}$ outputs a guess b' , and wins when $b = b'$. Ledger indistinguishability requires that $\mathcal{A}$ wins L-IND with probability at most negligibly greater than $1/2$.

Transaction non-malleability. This property requires that no bounded adversary $\mathcal{A}$ can alter any of the data stored within a (valid) pour transaction tx_{Pour} . This *transaction non-malleability* prevents malicious attackers from modifying others' transactions before they are added to the ledger (e.g., by re-targeting the Basecoin public output of a pour transaction).

Transaction non-malleability is formalized by an experiment TR-NM, in which $\mathcal{A}$ adaptively interacts with a DAP scheme oracle $\mathcal{O}^{\text{DAP}}$ and then outputs a pour transaction tx^* . Letting $\mathcal{T}$ denote the set of pour transactions returned by $\mathcal{O}^{\text{DAP}}$, and L denote the final ledger, $\mathcal{A}$ wins the game if there exists $\text{tx} \in \mathcal{T}$, such that (i) $\text{tx}^* \neq \text{tx}$; (ii) tx^* reveals a serial number contained in tx ; and (iii) both tx and tx^* are valid with respect to the ledger L' containing all transactions preceding tx on L . In other words, $\mathcal{A}$ wins the game if tx^* manages to modify some previous pour transaction to spend the same coin in a different way.

Transaction non-malleability requires that $\mathcal{A}$ wins TR-NM with only negligible probability. (Note that $\mathcal{A}$ can of course

produce valid pour transactions that are unrelated to those in $\mathcal{T}$; the condition that tx^* reveals a serial number of a previously-spent coin captures non-malleability.)

Balance. This property requires that no bounded adversary $\mathcal{A}$ can own more money than what he minted or received via payments from others.

Balance is formalized by an experiment BAL, in which $\mathcal{A}$ adaptively interacts with a DAP scheme oracle $\mathcal{O}^{\text{DAP}}$ and then outputs a set of coins S_{coin} . Letting S_{addr} be set of addresses returned by **CreateAddress** queries (i.e., addresses of “honest” users), $\mathcal{A}$ wins the game if the total value he can spend or has spent (either as coins or *Basecoin* public outputs) is greater than the value he has received or mined. That is, $\mathcal{A}$ wins if $v_{\text{Unspent}} + v_{\text{Basecoin}} + v_{\mathcal{A} \rightarrow \text{ADDR}} > v_{\text{Mint}} + v_{\text{ADDR} \rightarrow \mathcal{A}}$ where: (i) v_{Unspent} is the total value of unspent coins in S_{coin} ; (ii) v_{Basecoin} is the total value of public outputs placed by $\mathcal{A}$ on the ledger; (iii) v_{Mint} is the total value of $\mathcal{A}$'s mint transactions; (iv) $v_{\text{ADDR} \rightarrow \mathcal{A}}$ is the total value of payments received by $\mathcal{A}$ from addresses in S_{addr} ; (v) $v_{\mathcal{A} \rightarrow \text{ADDR}}$ is the total value of payments sent by $\mathcal{A}$ to addresses in S_{addr} .

Balance requires that $\mathcal{A}$ wins BAL with only negligible probability.

IV. CONSTRUCTION OF A DECENTRALIZED ANONYMOUS PAYMENT SCHEME

We show how to construct a DAP scheme (introduced in Section III) using zk-SNARKs and other building blocks. Later, in Section V, we give a concrete instantiation of this construction.

A. Cryptographic building blocks

We first introduce notation for the standard cryptographic building blocks that we use. We assume familiarity with the definitions of these building blocks; for more details, see, e.g., [27]. Throughout, λ denotes the security parameter.

Collision-resistant hashing. We use a collision-resistant hash function $\text{CRH}: \{0, 1\}^* \rightarrow \{0, 1\}^{O(\lambda)}$.

Pseudorandom functions. We use a pseudorandom function family $\text{PRF} = \{\text{PRF}_x: \{0, 1\}^* \rightarrow \{0, 1\}^{O(\lambda)}\}_x$ where x denotes the seed. From PRF_x , we derive three “non-overlapping” pseudorandom functions, chosen arbitrarily as $\text{PRF}_x^{\text{addr}}(z) := \text{PRF}_x(00\|z)$, $\text{PRF}_x^{\text{sn}}(z) := \text{PRF}_x(01\|z)$, $\text{PRF}_x^{\text{pk}}(z) := \text{PRF}_x(10\|z)$. Furthermore, we assume that PRF^{sn} is also collision resistant, in the sense that it is infeasible to find $(x, z) \neq (x', z')$ such that $\text{PRF}_x^{\text{sn}}(z) = \text{PRF}_{x'}^{\text{sn}}(z')$.

Statistically-hiding commitments. We use a commitment scheme COMM where the binding property holds computationally, while the hiding property holds statistically. It is denoted $\{\text{COMM}_x: \{0, 1\}^* \rightarrow \{0, 1\}^{O(\lambda)}\}_x$ where x denotes the commitment trapdoor. Namely, to reveal a commitment cm to a value z , it suffices to provide z and the trapdoor x ; then one can check that $\text{cm} = \text{COMM}_x(z)$.

One-time strongly-unforgeable digital signatures. We use a digital signature scheme $\text{Sig} = (\mathcal{G}_{\text{sig}}, \mathcal{K}_{\text{sig}}, \mathcal{S}_{\text{sig}}, \mathcal{V}_{\text{sig}})$ that works as follows.

- $\mathcal{G}_{\text{sig}}(1^\lambda) \rightarrow \text{pp}_{\text{sig}}$. Given a security parameter λ (presented in unary), $\mathcal{G}_{\text{sig}}$ samples public parameters pp_{enc} for the encryption scheme.
- $\mathcal{K}_{\text{sig}}(\text{pp}_{\text{sig}}) \rightarrow (\text{pk}_{\text{sig}}, \text{sk}_{\text{sig}})$. Given public parameters pp_{sig} , $\mathcal{K}_{\text{sig}}$ samples a public key and a secret key for a single user.
- $\mathcal{S}_{\text{sig}}(\text{sk}_{\text{sig}}, m) \rightarrow \sigma$. Given a secret key sk_{sig} and a message m , $\mathcal{S}_{\text{sig}}$ signs m to obtain a signature σ .
- $\mathcal{V}_{\text{sig}}(\text{pk}_{\text{sig}}, m, \sigma) \rightarrow b$. Given a public key pk_{sig} , message m , and signature σ , $\mathcal{V}_{\text{sig}}$ outputs $b = 1$ if the signature σ is valid for message m ; else it outputs $b = 0$.

The signature scheme Sig satisfies the security property of *one-time strong unforgeability against chosen-message attacks* (SUF-1CMA security).

Key-private public-key encryption. We use a public-key encryption scheme $\text{Enc} = (\mathcal{G}_{\text{enc}}, \mathcal{K}_{\text{enc}}, \mathcal{E}_{\text{enc}}, \mathcal{D}_{\text{enc}})$ that works as follows.

- $\mathcal{G}_{\text{enc}}(1^\lambda) \rightarrow \text{pp}_{\text{enc}}$. Given a security parameter λ (presented in unary), $\mathcal{G}_{\text{enc}}$ samples public parameters pp_{enc} for the encryption scheme.
- $\mathcal{K}_{\text{enc}}(\text{pp}_{\text{enc}}) \rightarrow (\text{pk}_{\text{enc}}, \text{sk}_{\text{enc}})$. Given public parameters pp_{enc} , $\mathcal{K}_{\text{enc}}$ samples a public key and a secret key for a single user.
- $\mathcal{E}_{\text{enc}}(\text{pk}_{\text{enc}}, m) \rightarrow c$. Given a public key pk_{enc} and a message m , $\mathcal{E}_{\text{enc}}$ encrypts m to obtain a ciphertext c .
- $\mathcal{D}_{\text{enc}}(\text{sk}_{\text{enc}}, c) \rightarrow m$. Given a secret key sk_{enc} and a ciphertext c , $\mathcal{D}_{\text{enc}}$ decrypts c to produce a message m (or $\perp$ if decryption fails).

The encryption scheme Enc satisfies two security properties: (i) *ciphertext indistinguishability under chosen-ciphertext attack* (IND-CCA security); and (ii) *key indistinguishability under chosen-ciphertext attack* (IK-CCA security). While the first property is standard, the second is less known; informally, IK-CCA requires that ciphertexts cannot be linked to the public key used to encrypt them, or to other ciphertexts encrypted with the same public key. For definitions, we refer the reader to [21].

B. zk-SNARKs for pouring coins

As outlined in Section I-B, our construction invokes a zk-SNARK for a specific NP statement, POUR , which we now define. We first recall the context motivating POUR . When a user u pours “old” coins $c_1^{\text{old}}, c_2^{\text{old}}$ into new coins $c_1^{\text{new}}, c_2^{\text{new}}$, a corresponding pour transaction

$$\text{tx}_{\text{Pour}} = (\text{rt}, \text{sn}_1^{\text{old}}, \text{sn}_2^{\text{old}}, \text{cm}_1^{\text{new}}, \text{cm}_2^{\text{new}}, v_{\text{pub}}, \text{info}, *)$$

is generated. In our construction, we need to provide evidence in “*” that various conditions were respected by the pour operation. Concretely, tx_{Pour} should demonstrate that (i) u owns $c_1^{\text{old}}, c_2^{\text{old}}$; (ii) coin commitments for $c_1^{\text{old}}, c_2^{\text{old}}$ appear somewhere on the ledger; (iii) the revealed serial numbers $\text{sn}_1^{\text{old}}, \text{sn}_2^{\text{old}}$ are of $c_1^{\text{old}}, c_2^{\text{old}}$; (iv) the revealed coin commitments $\text{cm}_1^{\text{new}}, \text{cm}_2^{\text{new}}$ are of $c_1^{\text{new}}, c_2^{\text{new}}$; (v) balance is preserved. Our construction achieves this by including a zk-SNARK proof π_{Pour} for the statement POUR which checks the above invariants (as well as others needed for non-malleability).

The statement POUR . Concretely, the NP statement POUR is defined as follows.

- Instances are of the form $\vec{x} = (\text{rt}, \text{sn}_1^{\text{old}}, \text{sn}_2^{\text{old}}, \text{cm}_1^{\text{new}}, \text{cm}_2^{\text{new}}, v_{\text{pub}}, h_{\text{Sig}}, h_1, h_2)$. Thus, an instance $\vec{x}$ specifies a root rt for a CRH-based Merkle tree (over the list of commitments so far), the two serial numbers of the consumed coins, two coin commitments for the two new coins, a public value, and fields h_{Sig}, h_1, h_2 used for non-malleability.
- Witnesses are of the form $\vec{a} = (\text{path}_1, \text{path}_2, c_1^{\text{old}}, c_2^{\text{old}}, \text{addr}_{\text{sk},1}^{\text{old}}, \text{addr}_{\text{sk},2}^{\text{old}}, c_1^{\text{new}}, c_2^{\text{new}})$ where, for each $i \in \{1, 2\}$:

$$c_i^{\text{old}} = (\text{addr}_{\text{pk},i}^{\text{old}}, v_i^{\text{old}}, \rho_i^{\text{old}}, r_i^{\text{old}}, s_i^{\text{old}}, \text{cm}_i^{\text{old}}),$$

$$c_i^{\text{new}} = (\text{addr}_{\text{pk},i}^{\text{new}}, v_i^{\text{new}}, \rho_i^{\text{new}}, r_i^{\text{new}}, s_i^{\text{new}}, \text{cm}_i^{\text{new}})$$

for the same cm_i^{new} as in $\vec{x}$,

$$\text{addr}_{\text{pk},i}^{\text{old}} = (a_{\text{pk},i}^{\text{old}}, \text{pk}_{\text{enc},i}^{\text{old}}),$$

$$\text{addr}_{\text{pk},i}^{\text{new}} = (a_{\text{pk},i}^{\text{new}}, \text{pk}_{\text{enc},i}^{\text{new}}),$$

$$\text{addr}_{\text{sk},i}^{\text{old}} = (a_{\text{sk},i}^{\text{old}}, \text{sk}_{\text{enc},i}^{\text{old}}).$$

Thus, a witness $\vec{a}$ specifies authentication paths for the two new coin commitments, the entirety of coin information about both the old and new coins, and address secret keys for the old coins.

Given a POUR instance $\vec{x}$, a witness $\vec{a}$ is valid for $\vec{x}$ if the following holds:

- 1) For each $i \in \{1, 2\}$:
 - a) The coin commitment cm_i^{old} of c_i^{old} appears on the ledger, i.e., path_i is a valid authentication path for leaf cm_i^{old} with respect to root rt , in a CRH-based Merkle tree.
 - b) The address secret key $a_{\text{sk},i}^{\text{old}}$ matches the address public key of c_i^{old} , i.e., $a_{\text{pk},i}^{\text{old}} = \text{PRF}_{a_{\text{sk},i}^{\text{old}}}^{\text{addr}}(0)$.
 - c) The serial number sn_i^{old} of c_i^{old} is computed correctly, i.e., $\text{sn}_i^{\text{old}} = \text{PRF}_{a_{\text{sk},i}^{\text{old}}}^{\text{sn}}(\rho_i^{\text{old}})$.
 - d) The coin c_i^{old} is well-formed, i.e., $\text{cm}_i^{\text{old}} = \text{COMM}_{s_i^{\text{old}}}(\text{COMM}_{r_i^{\text{old}}}(a_{\text{pk},i}^{\text{old}} \parallel \rho_i^{\text{old}}) \parallel v_i^{\text{old}})$.
 - e) The coin c_i^{new} is well-formed, i.e., $\text{cm}_i^{\text{new}} = \text{COMM}_{s_i^{\text{new}}}(\text{COMM}_{r_i^{\text{new}}}(a_{\text{pk},i}^{\text{new}} \parallel \rho_i^{\text{new}}) \parallel v_i^{\text{new}})$.
 - f) The address secret key $a_{\text{sk},i}^{\text{old}}$ ties h_{Sig} to h_i , i.e., $h_i = \text{PRF}_{a_{\text{sk},i}^{\text{old}}}^{\text{pk}}(h_{\text{Sig}})$.
- 2) Balance is preserved: $v_1^{\text{new}} + v_2^{\text{new}} + v_{\text{pub}} = v_1^{\text{old}} + v_2^{\text{old}}$ (with $v_1^{\text{old}}, v_2^{\text{old}} \geq 0$ and $v_1^{\text{old}} + v_2^{\text{old}} \leq v_{\text{max}}$).

Recall that in this paper zk-SNARKs are relative to the language of arithmetic circuit satisfiability (see Section II); thus, we express the checks in POUR via an arithmetic circuit, denoted C_{POUR} . In particular, the depth d_{tree} of the Merkle tree needs to be hardcoded in C_{POUR} , and we thus make it a parameter of our construction (see below); the maximum number of supported coins is then $2^{d_{\text{tree}}}$.

C. Algorithm constructions

We proceed to describe the construction of the DAP scheme $\Pi = (\text{Setup}, \text{CreateAddress}, \text{Mint}, \text{Pour}, \text{VerifyTransaction}, \text{Receive})$ whose intuition was given in Section I-B. Figure 2 gives the pseudocode for each one of the six algorithms in Π , in terms of the building blocks introduced in Section IV-A and Section IV-B. In the construction, we hardcode two quantities:

Setup

- INPUTS: security parameter λ
- OUTPUTS: public parameters pp
- 1) Construct C_{POUR} for POUR at security λ .
- 2) Compute $(\text{pk}_{\text{POUR}}, \text{vk}_{\text{POUR}}) := \text{KeyGen}(1^\lambda, C_{\text{POUR}})$.
- 3) Compute $\text{pp}_{\text{enc}} := \mathcal{G}_{\text{enc}}(1^\lambda)$.
- 4) Compute $\text{pp}_{\text{sig}} := \mathcal{G}_{\text{sig}}(1^\lambda)$.
- 5) Set $\text{pp} := (\text{pk}_{\text{POUR}}, \text{vk}_{\text{POUR}}, \text{pp}_{\text{enc}}, \text{pp}_{\text{sig}})$.
- 6) Output pp .

CreateAddress

- INPUTS: public parameters pp
- OUTPUTS: address key pair $(\text{addr}_{\text{pk}}, \text{addr}_{\text{sk}})$
- 1) Compute $(\text{pk}_{\text{enc}}, \text{sk}_{\text{enc}}) := \mathcal{K}_{\text{enc}}(\text{pp}_{\text{enc}})$.
- 2) Randomly sample a PRF^{addr} seed a_{sk} .
- 3) Compute $a_{\text{pk}} = \text{PRF}^{\text{addr}}_{a_{\text{sk}}}(0)$.
- 4) Set $\text{addr}_{\text{pk}} := (a_{\text{pk}}, \text{pk}_{\text{enc}})$.
- 5) Set $\text{addr}_{\text{sk}} := (a_{\text{sk}}, \text{sk}_{\text{enc}})$.
- 6) Output $(\text{addr}_{\text{pk}}, \text{addr}_{\text{sk}})$.

Mint

- INPUTS:
 - public parameters pp
 - coin value $v \in \{0, 1, \dots, v_{\text{max}}\}$
 - destination address public key addr_{pk}
- OUTPUTS: coin c and mint transaction tx_{Mint}
- 1) Parse addr_{pk} as $(a_{\text{pk}}, \text{pk}_{\text{enc}})$.
- 2) Randomly sample a PRF^{sn} seed ρ .
- 3) Randomly sample two COMM trapdoors r, s .
- 4) Compute $k := \text{COMM}_r(a_{\text{pk}} \parallel \rho)$.
- 5) Compute $\text{cm} := \text{COMM}_s(v \parallel k)$.
- 6) Set $\text{c} := (\text{addr}_{\text{pk}}, v, \rho, r, s, \text{cm})$.
- 7) Set $\text{tx}_{\text{Mint}} := (\text{cm}, v, *)$, where $*$:= (k, s) .
- 8) Output c and tx_{Mint} .

VerifyTransaction

- INPUTS:
 - public parameters pp
 - a (mint or pour) transaction tx
 - the current ledger L
- OUTPUTS: bit b , equals 1 iff the transaction is valid
- 1) If given a mint transaction $\text{tx} = \text{tx}_{\text{Mint}}$:
 - a) Parse tx_{Mint} as $(\text{cm}, v, *)$, and $*$ as (k, s) .
 - b) Set $\text{cm}' := \text{COMM}_s(v \parallel k)$.
 - c) Output $b := 1$ if $\text{cm} = \text{cm}'$, else output $b := 0$.
- 2) If given a pour transaction $\text{tx} = \text{tx}_{\text{POUR}}$:
 - a) Parse tx_{POUR} as $(\text{rt}, \text{sn}_1^{\text{old}}, \text{sn}_2^{\text{old}}, \text{cm}_1^{\text{new}}, \text{cm}_2^{\text{new}}, v_{\text{pub}}, \text{info}, *)$, and $*$ as $(\text{pk}_{\text{sig}}, h_1, h_2, \pi_{\text{POUR}}, \text{C}_1, \text{C}_2, \sigma)$.
 - b) If sn_1^{old} or sn_2^{old} appears on L (or $\text{sn}_1^{\text{old}} = \text{sn}_2^{\text{old}}$), output $b := 0$.
 - c) If the Merkle root rt does not appear on L , output $b := 0$.
 - d) Compute $h_{\text{Sig}} := \text{CRH}(\text{pk}_{\text{sig}})$.
 - e) Set $\vec{x} := (\text{rt}, \text{sn}_1^{\text{old}}, \text{sn}_2^{\text{old}}, \text{cm}_1^{\text{new}}, \text{cm}_2^{\text{new}}, v_{\text{pub}}, h_{\text{Sig}}, h_1, h_2)$.
 - f) Set $m := (\vec{x}, \pi_{\text{POUR}}, \text{info}, \text{C}_1, \text{C}_2)$.
 - g) Compute $b := \mathcal{V}_{\text{sig}}(\text{pk}_{\text{sig}}, m, \sigma)$.
 - h) Compute $b' := \text{Verify}(\text{vk}_{\text{POUR}}, \vec{x}, \pi_{\text{POUR}})$, and output $b \wedge b'$.

Pour

- INPUTS:
 - public parameters pp
 - the Merkle root rt
 - old coins $\text{c}_1^{\text{old}}, \text{c}_2^{\text{old}}$
 - old addresses secret keys $\text{addr}_{\text{sk},1}^{\text{old}}, \text{addr}_{\text{sk},2}^{\text{old}}$
 - path path_1 from commitment $\text{cm}(\text{c}_1^{\text{old}})$ to root rt , path path_2 from commitment $\text{cm}(\text{c}_2^{\text{old}})$ to root rt
 - new values $v_1^{\text{new}}, v_2^{\text{new}}$
 - new addresses public keys $\text{addr}_{\text{pk},1}^{\text{new}}, \text{addr}_{\text{pk},2}^{\text{new}}$
 - public value v_{pub}
 - transaction string info
- OUTPUTS: new coins $\text{c}_1^{\text{new}}, \text{c}_2^{\text{new}}$ and pour transaction tx_{POUR}
- 1) For each $i \in \{1, 2\}$:
 - a) Parse c_i^{old} as $(\text{addr}_{\text{pk},i}^{\text{old}}, v_i^{\text{old}}, \rho_i^{\text{old}}, r_i^{\text{old}}, s_i^{\text{old}}, \text{cm}_i^{\text{old}})$.
 - b) Parse $\text{addr}_{\text{sk},i}^{\text{old}}$ as $(a_{\text{sk},i}^{\text{old}}, \text{sk}_{\text{enc},i}^{\text{old}})$.
 - c) Compute $\text{sn}_i^{\text{old}} := \text{PRF}_{a_{\text{sk},i}^{\text{old}}}^{\text{sn}}(\rho_i^{\text{old}})$.
 - d) Parse $\text{addr}_{\text{pk},i}^{\text{new}}$ as $(a_{\text{pk},i}^{\text{new}}, \text{pk}_{\text{enc},i}^{\text{new}})$.
 - e) Randomly sample a PRF^{sn} seed ρ_i^{new} .
 - f) Randomly sample two COMM trapdoors $r_i^{\text{new}}, s_i^{\text{new}}$.
 - g) Compute $k_i^{\text{new}} := \text{COMM}_{r_i^{\text{new}}}(a_{\text{pk},i}^{\text{new}} \parallel \rho_i^{\text{new}})$.
 - h) Compute $\text{cm}_i^{\text{new}} := \text{COMM}_{s_i^{\text{new}}}(v_i^{\text{new}} \parallel k_i^{\text{new}})$.
 - i) Set $\text{c}_i^{\text{new}} := (\text{addr}_{\text{pk},i}^{\text{new}}, v_i^{\text{new}}, \rho_i^{\text{new}}, r_i^{\text{new}}, s_i^{\text{new}}, \text{cm}_i^{\text{new}})$.
 - j) Set $\text{C}_i := \mathcal{E}_{\text{enc}}(\text{pk}_{\text{enc},i}^{\text{new}}, (v_i^{\text{new}}, \rho_i^{\text{new}}, r_i^{\text{new}}, s_i^{\text{new}}))$.
- 2) Generate $(\text{pk}_{\text{sig}}, \text{sk}_{\text{sig}}) := \mathcal{K}_{\text{sig}}(\text{pp}_{\text{sig}})$.
- 3) Compute $h_{\text{Sig}} := \text{CRH}(\text{pk}_{\text{sig}})$.
- 4) Compute $h_1 := \text{PRF}_{a_{\text{sk},1}^{\text{old}}}^{\text{pk}}(h_{\text{Sig}})$ and $h_2 := \text{PRF}_{a_{\text{sk},2}^{\text{old}}}^{\text{pk}}(h_{\text{Sig}})$.
- 5) Set $\vec{x} := (\text{rt}, \text{sn}_1^{\text{old}}, \text{sn}_2^{\text{old}}, \text{cm}_1^{\text{new}}, \text{cm}_2^{\text{new}}, v_{\text{pub}}, h_{\text{Sig}}, h_1, h_2)$.
- 6) Set $\vec{a} := (\text{path}_1, \text{path}_2, \text{c}_1^{\text{old}}, \text{c}_2^{\text{old}}, \text{addr}_{\text{sk},1}^{\text{old}}, \text{addr}_{\text{sk},2}^{\text{old}}, \text{c}_1^{\text{new}}, \text{c}_2^{\text{new}})$.
- 7) Compute $\pi_{\text{POUR}} := \text{Prove}(\text{pk}_{\text{POUR}}, \vec{x}, \vec{a})$.
- 8) Set $m := (\vec{x}, \pi_{\text{POUR}}, \text{info}, \text{C}_1, \text{C}_2)$.
- 9) Compute $\sigma := \mathcal{S}_{\text{sig}}(\text{sk}_{\text{sig}}, m)$.
- 10) Set $\text{tx}_{\text{POUR}} := (\text{rt}, \text{sn}_1^{\text{old}}, \text{sn}_2^{\text{old}}, \text{cm}_1^{\text{new}}, \text{cm}_2^{\text{new}}, v_{\text{pub}}, \text{info}, *)$, where $*$:= $(\text{pk}_{\text{sig}}, h_1, h_2, \pi_{\text{POUR}}, \text{C}_1, \text{C}_2, \sigma)$.
- 11) Output $\text{c}_1^{\text{new}}, \text{c}_2^{\text{new}}$ and tx_{POUR} .

Receive

- INPUTS:
 - public parameters pp
 - recipient address key pair $(\text{addr}_{\text{pk}}, \text{addr}_{\text{sk}})$
 - the current ledger L
- OUTPUTS: set of received coins
- 1) Parse addr_{pk} as $(a_{\text{pk}}, \text{pk}_{\text{enc}})$.
- 2) Parse addr_{sk} as $(a_{\text{sk}}, \text{sk}_{\text{enc}})$.
- 3) For each Pour transaction tx_{POUR} on the ledger:
 - a) Parse tx_{POUR} as $(\text{rt}, \text{sn}_1^{\text{old}}, \text{sn}_2^{\text{old}}, \text{cm}_1^{\text{new}}, \text{cm}_2^{\text{new}}, v_{\text{pub}}, \text{info}, *)$, and $*$ as $(\text{pk}_{\text{sig}}, h_1, h_2, \pi_{\text{POUR}}, \text{C}_1, \text{C}_2, \sigma)$.
 - b) For each $i \in \{1, 2\}$:
 - i) Compute $(v_i, \rho_i, r_i, s_i) := \mathcal{D}_{\text{enc}}(\text{sk}_{\text{enc}}, \text{C}_i)$.
 - ii) If $\mathcal{D}_{\text{enc}}$'s output is not $\perp$, verify that:
 - cm_i^{new} equals $\text{COMM}_{s_i}(v_i \parallel \text{COMM}_{r_i}(a_{\text{pk}} \parallel \rho_i))$;
 - $\text{sn}_i := \text{PRF}_{a_{\text{sk}}}^{\text{sn}}(\rho_i)$ does not appear on L .
 - iii) If both checks succeed, output $\text{c}_i := (\text{addr}_{\text{pk}}, v_i, \rho_i, r_i, s_i, \text{cm}_i^{\text{new}})$.

Fig. 2: Construction of a DAP scheme using zk-SNARKs and other ingredients.

the maximum value of a coin, v_{max} , and the depth of the Merkle tree, d_{tree} .

D. Completeness and security

Our main theorem states that the above construction is indeed a DAP scheme.

Theorem IV.1. The tuple $\Pi = (\text{Setup}, \text{CreateAddress}, \text{Mint}, \text{Pour}, \text{VerifyTransaction}, \text{Receive})$, as defined in Section IV-C,

is a complete (cf. Definition III.1) and secure (cf. Definition III.2) DAP scheme.

We provide a proof of Theorem IV.1 in the extended version of this paper [26]. We note that our construction can be modified to yield statistical (i.e., everlasting) anonymity; see the discussion in the extension section of the full version of this paper.

Remark (trusted setup). Security of Π relies on a trusted party

running Setup to generate the public parameters (once and for all). This trust is needed for the transaction non-malleability and balance properties but not for ledger indistinguishability. Thus, even if a powerful espionage agency were to corrupt the setup, anonymity will *still* be maintained. Moreover, if one wishes to mitigate the trust requirements of this step, one can conduct the computation of Setup using secure multiparty computation techniques; we leave this to future work.

V. ZEROCASH

We describe a concrete instantiation of a DAP scheme; this instantiation forms the basis of Zerocash. Later, in Section VI, we discuss how Zerocash can be integrated with existing ledger-based currencies.

A. Instantiation of building blocks

We instantiate the DAP scheme construction from Section IV (see Figure 2), aiming at a level of security of 128 bits. Doing so requires concrete choices, described next.

CRH, PRF, COMM from SHA256. Let $\mathcal{H}$ be the SHA256 compression function, which maps a 512-bit input to a 256-bit output. We mostly rely on $\mathcal{H}$, rather than the “full” hash, since this suffices for our fixed-size single-block inputs, and it simplifies the construction of C_{POUR} . We instantiate CRH, PRF, COMM via $\mathcal{H}$ (under suitable assumptions on $\mathcal{H}$).

First, we instantiate the collision-resistant hash function CRH as $\mathcal{H}(z)$ for $z \in \{0, 1\}^{512}$; this function compresses “two-to-one”, so it can be used to construct binary Merkle trees.¹³

Next, we instantiate the pseudorandom function $\text{PRF}_x(z)$ as $\mathcal{H}(x\|z)$, with $x \in \{0, 1\}^{256}$ as the seed, and $z \in \{0, 1\}^{256}$ as the input.¹⁴ Thus, the derived functions are $\text{PRF}_x^{\text{addr}}(z) := \mathcal{H}(x\|00\|z)$, $\text{PRF}_x^{\text{sn}}(z) := \mathcal{H}(x\|01\|z)$ and $\text{PRF}_x^{\text{pk}}(z) := \mathcal{H}(x\|10\|z)$, with $x \in \{0, 1\}^{256}$ and $z \in \{0, 1\}^{254}$.

As for the commitment scheme COMM, we only use it in the following pattern:

$$\begin{aligned} k &:= \text{COMM}_r(a_{\text{pk}}\|\rho) \\ \text{cm} &:= \text{COMM}_s(v\|k) \end{aligned}$$

Due to our instantiation of PRF, a_{pk} is 256 bits. So we can set ρ also to 256 bits and r to $256 + 128 = 384$ bits; then we can compute $k := \text{COMM}_r(a_{\text{pk}}\|\rho)$ as $\mathcal{H}(r\|\mathcal{H}(a_{\text{pk}}\|\rho)_{128})$. Above, $[\cdot]_{128}$ denotes that we are truncating the 256-bit string to 128 bits (say, by dropping least-significant bits, as in our implementation). Heuristically, for any string $x \in \{0, 1\}^{128}$, the distribution induced by $\mathcal{H}(r\|x)$ is 2^{-128} -close to uniform, and this forms the basis of the statistically-hiding property. For computing cm, we set coin values to be 64-bit integers (so that, in particular, $v_{\text{max}} = 2^{64} - 1$ in our implementation), and then compute $\text{cm} := \text{COMM}_s(v\|k)$ as $\mathcal{H}(k\|0^{192}\|v)$. Noticeably,

¹³A single exception: we still compute h_{sig} according to the full hash SHA256, rather than its compression function, because there is no need for this computation to be verified by C_{POUR} .

¹⁴This assumption is reminiscent of previous works analyzing the security of hash-based constructions (e.g., [28]). However in this work we assume that a portion of the compression function is the *seed* for the pseudorandom function, rather than using the chaining variable as in [28].

above we are *ignoring* the commitment randomness s . The reason is that we already know that k , being the output of a statistically-hiding commitment, can serve as randomness for the next commitment scheme.

Instantiating the NP statement POUR. The above choices imply a concrete instantiation of the NP statement POUR (see Section IV-B). Specifically, in our implementation, POUR checks that the following holds, for each $i \in \{1, 2\}$:

- path_i is an authentication path for leaf cm_i^{old} with respect to root rt , in a CRH-based Merkle tree;
- $a_{\text{pk},i}^{\text{old}} = \mathcal{H}(a_{\text{sk},i}^{\text{old}}\|0^{256})$;
- $\text{sn}_i^{\text{old}} = \mathcal{H}(a_{\text{sk},i}^{\text{old}}\|01\|[\rho_i^{\text{old}}]_{254})$;
- $\text{cm}_i^{\text{old}} = \mathcal{H}(\mathcal{H}(r_i^{\text{old}}\|\mathcal{H}(a_{\text{pk},i}^{\text{old}}\|\rho_i^{\text{old}})_{128})\|0^{192}\|v_i^{\text{old}})$;
- $\text{cm}_i^{\text{new}} = \mathcal{H}(\mathcal{H}(r_i^{\text{new}}\|\mathcal{H}(a_{\text{pk},i}^{\text{new}}\|\rho_i^{\text{new}})_{128})\|0^{192}\|v_i^{\text{new}})$; and
- $h_i = \mathcal{H}(a_{\text{sk},i}^{\text{old}}\|10\|h_{\text{sig}})_{254}$.

Moreover, POUR checks that $v_1^{\text{new}} + v_2^{\text{new}} + v_{\text{pub}} = v_1^{\text{old}} + v_2^{\text{old}}$, with $v_1^{\text{old}}, v_2^{\text{old}} \geq 0$ and $v_1^{\text{old}} + v_2^{\text{old}} < 2^{64}$.

Finally, as mentioned, in order for C_{POUR} to be well-defined, we need to fix a Merkle tree depth d_{tree} . In our implementation, we fix $d_{\text{tree}} = 64$, and thus support up to 2^{64} coins.

Instantiating Sig. For the signature scheme Sig, we use ECDSA to retain consistency and compatibility with the existing `bitcoind` source code. However, standard ECDSA is malleable: both (r, s) and $(r, -s)$ verify as valid signatures. We use a non-malleable variant, where s is restricted to the “lower half” of field elements. While we are not aware of a formal SUF-CMA proof for this variant, its use is consistent with proposals to resolve Bitcoin transaction malleability [29].¹⁵

Instantiating Enc. For the encryption scheme Enc, we use the key-private Elliptic-Curve Integrated Encryption Scheme (ECIES) [30, 31]; it is one of the few standardized key-private encryption schemes with available implementations.

For further details about efficiently realizing these in the arithmetic circuit for POUR, see the full version of this paper.

VI. INTEGRATION WITH EXISTING LEDGER-BASED CURRENCIES

Zerocash can be deployed atop any ledger (even one maintained by a central bank.) Here, we briefly detail integration with the Bitcoin protocol. Unless explicitly stated otherwise, in the following section when referring to *Bitcoin*, and its unit of account *bitcoin* (plural bitcoins), we mean the underlying protocol and software, not the currency system. (The discussion holds, with little or no modification, for many forks of Bitcoin, a.k.a. “altcoins”, such as Litecoin.)

By introducing new transaction types and payment semantics, Zerocash breaks compatibility with the Bitcoin network. While Zerocash could be integrated into Bitcoin (the actual currency and its supporting software) via a “flag day” where a supermajority of Bitcoin miners simultaneously adopt the new software, we neither expect nor advise such integration in the near future and suggest using Zerocash in a separate altcoin.

¹⁵In practice, one might replace this ECDSA variant with an EC-Schnorr signature satisfying SUF-CMA security with proper encoding of EC group elements; the performance would be similar.

Integrating Zerocash into Bitcoin consists of adding a new transaction type, Zerocash transactions, and modifying the protocol and software to invoke Zerocash’s DAP interface to create and verify these transactions. Two approaches to doing so are described next, followed by a discussion of anonymizing the network layer.

A. Integration by replacing the base currency

One approach is to alter the underlying system so that all monetary transactions are done using Zerocash, i.e., by invoking the DAP interface and writing/reading the associated transactions in the distributed ledger.

As seen in Section III, this suffices to offer the core functionality of payments, minting, merging, splitting, etc., while assuring users that all transactions using this currency are anonymous. However, this has several drawbacks: all transactions incur the cost of generating a zk-SNARK proof; the scripting feature of Bitcoin is lost; and Bitcoin’s ability to spend unconfirmed transactions is lost.

B. Integration by hybrid currency

A different approach is to extend Bitcoin with a parallel, anonymized currency of “zerocoins,” existing alongside bitcoins, using the same ledger, and with the ability to convert freely between the two. The behavior and functionality of regular bitcoins is unaltered; in particular, they may support functionality such as scripting.

In this approach, the Bitcoin ledger consists of Bitcoin-style transactions, containing inputs and outputs [20]. Each input is either a pointer to an output of a previous transaction (as in plain Bitcoin), or a Zerocash pour transaction (which contributes its *public* value, v_{pub} , of bitcoins to this transaction). Outputs are either an amount and destination public address/script (as in plain Bitcoin), or a Zerocash mint transaction (which consumes the input bitcoins to produce zerocoins). The usual invariant over bitcoins is maintained and checked in plain view: the sum of bitcoin inputs (including pours’ v_{pub}) must be at least the sum of bitcoin outputs (including mints’ v), and any difference is offered as a transaction fee. However, the accounting for zerocoins consumed and produced is done separately and implicitly by the DAP scheme.

C. Additional anonymity considerations

Zerocash only anonymizes the transaction ledger. Network traffic used to announce transactions, retrieve blocks, and contact merchants will still leak identifying information (e.g., IP addresses). Thus users need some anonymity network to safely use Zerocash. The most obvious way to do this is via Tor [32]. Given that Zerocash transactions are not low latency themselves, Mixnets (e.g., Mixminion [33]) are also a viable way to add anonymity (and one that is not as vulnerable to traffic analysis as Tor). Using mixnets that provide email-like functionality has the added benefit of providing an out-of-band notification mechanism as a replacement to Receive.

Additionally, although in theory all users have a single view of the block chain, a powerful attacker could potentially

fabricate an additional block *solely* for a targeted user. Spending any coins with respect to the updated Merkle tree in this “poison-pill” block will uniquely identify the targeted user. To mitigate such attacks, users should check with trusted peers their view of the block chain and, for sensitive transactions, only spend coins relative to blocks further back in the ledger (since creating the illusion for multiple blocks is far harder).

VII. EXPERIMENTS

To measure the performance of Zerocash, we ran several experiments. First, we benchmarked the performance of the zk-SNARK for the NP statement `POUR` (Section VII-A) and of the six DAP scheme algorithms (Section VII-B). Second, we studied the impact of a higher block verification time via a simulation of a Bitcoin network (Section VII-C).

A. Performance of zk-SNARKs for pouring coins

Our zk-SNARK for the NP statement `POUR` is obtained by constructing an arithmetic circuit C_{POUR} for verifying `POUR`, and then invoking the generic implementation of zk-SNARK for arithmetic circuit satisfiability of [16] (see Section II-C). The arithmetic circuit C_{POUR} is built from scratch and hand-optimized to exploit nondeterministic verification and the large field characteristic.

Figure 3 reports performance characteristics of the resulting zk-SNARK for `POUR`. This includes three settings: single-thread performance on a laptop machine; and single-thread and multi-thread performance on a desktop machine. (The time measurements are the average of 10 runs, with standard deviation under 2.5%.)

B. Performance of Zerocash algorithms

In Figure 4 we report performance characteristics for each of the six DAP scheme algorithms in our implementation. Note that these numbers do not include the costs of maintaining the Merkle tree because doing so is not the responsibility of these algorithms. Moreover, for `VerifyTransaction`, we separately report the cost of verifying mint and pour transactions and, in the latter case, we exclude the cost of scanning L (as this cost depends on L). Finally, for the case of `Receive`, we report the cost to process a given pour transaction in L .

C. Large-scale network simulation

Because Bitcoin mining typically takes place on dedicated GPUs or ASICs, the CPU resources to execute the DAP scheme algorithms are often of minimal consequence to network performance. There is one potential exception to this rule: the `VerifyTransaction` algorithm must be run by all of the network nodes in the course of routine transaction validation. The time it takes to perform this verification can have significant impact on network performance.

In the Zerocash implementation (as in Bitcoin), every Zerocash transaction is verified at each hop as it is forwarded through the network and, potentially, again when blocks containing the transaction are verified. Verifying a block consists of checking the proof of work and validating the contained transactions.

		Intel Core i7-2620M @ 2.70GHz 12GB of RAM	Intel Core i7-4770 @ 3.40GHz 16GB of RAM	
		1 thread	1 thread	8 threads
KeyGen	Time	7 min 48 s	5 min 17 s	4 min 11 s
	Proving key	896 MiB		
	Verification key	749 B		
Prove	Time	2 min 55 s	2 min 2 s	1 min 3 s
	Proof	288 B		
Verify	Time	8.5 ms	5.4 ms	

Fig. 3: Performance of our zk-SNARK for the NP statement POUR.
($N = 10$, $\sigma \leq 2.5\%$)

Thus Zerocash transactions may take longer to spread though the network and blocks containing Zerocash transactions may take longer to verify. While we are concerned with the first issue, the potential impact of the second issue is cause for greater concern. This is because Zerocash transactions cannot be spent until they make it onto the ledger.

Because blocks are also verified at each hop before they are forwarded through the network, delays in block verification slow down the propagation of new blocks through the network. This causes nodes to waste CPU-cycles mining on out-of-date blocks, reducing the computational power of the network and making it easier to mount a “51% attack” (dishonest majority of miners) on the distributed ledger.

It is a priori unclear whether this potential issue is a real concern. Bitcoin caches transaction verifications, so a transaction that was already verified when it propagated through the network need not be verified again when it is seen in a block. The unknown is what percentage of transactions in a block are actually in any given node’s cache. We thus conduct a simulation of the Bitcoin network to investigate both the time it takes Zerocash transactions to make it onto the ledger and establish the effects of Zerocash transactions on block verification and propagation. We find that Zerocash transactions can be spent reasonably quickly and that the effects of increased block validation time are minimal.

Simulation design. Because Zerocash requires breaking changes to the Bitcoin protocol, we cannot test our protocol in the live Bitcoin network or even in the dedicated testnet. We must run our own private testnet. For efficiency and cost reasons, we would like to run as many Bitcoin nodes as possible on the least amount of hardware. This raises two issues. First, reducing the proof of work to practical levels while still preserving a realistic rate of new blocks is difficult (especially on virtualized hardware with variable performance). Second, the overhead of zk-SNARK verification prevents us from running many Bitcoin

¹⁶346 B of this are due to the ciphertexts C_1, C_2 . Future implementations may significantly reduce this overhead or discard these (cf. Section VI-C).

¹⁷We note that σ for both Mint and VerifyTransaction (mint) is higher than 2.5% due to the variability at such short timescales. Respectively, it is 3.3 μ s and 1.9 μ s.

Intel Core i7-4770 @ 3.40GHz with 16GB of RAM (1 thread)		
Setup	Time	5 min 17 s
	pp	896 MiB
CreateAddress	Time	326.0 ms
	addr _{pk}	343 B
	addr _{sk}	319 B
Mint	Time	23 μ s
	Coin c	463 B
	tx _{Mint}	72 B
Pour	Time	2 min 2.01 s
	tx _{Pour}	996 B ¹⁶
VerifyTransaction	mint	8.3 μ s
	pour (excludes L scan)	5.7 ms
Receive	Time (per pour tx)	1.6 ms

Fig. 4: Performance of Zerocash algorithms.
($N = 10$, $\sigma \leq 2.5\%$ ¹⁷)

nodes on one virtualized server.

The frequency of new blocks can be modeled as a Poisson process with a mean of Λ_{block} seconds. To generate blocks stochastically, we modify `bitcoind` to fix its block difficulty at a trivial level and run a Poisson process, on the simulation control server, which trivially mines a block on a randomly selected node. This preserves the distribution of blocks, without the computational overhead of a real proof of work. Another Poisson process triggering mechanism, with a different mean Λ_{tx} , introduces new transactions at random network nodes.

To differentiate which transactions represent normal Bitcoin expenditures vs. which contain Zerocash pour transactions, simulated Zerocash transactions pay a unique amount of bitcoins (we set this value arbitrarily at 7 BTC). If a transaction’s output matches this preset value, and it is not in verification cache, then our modified Bitcoin client inserts a 10 ms delay simulating the runtime of `VerifyTransaction`.¹⁸ Otherwise transactions are processed as specified by the Bitcoin protocol. We vary the amount of simulated Zerocash traffic by varying the number of transactions with this particular output amount. This minimizes code changes and estimates only the generic impact of verification delays and not of any specific implementation choice.

Methodology. Recent research [17] suggests that the Bitcoin network contains 16,000 distinct nodes though most are likely no longer participating: approximately 3,500 are reachable at any given time. Each node has an average of 32 open connections to randomly selected peers. As of November 2013, the peak observed transaction rate for Bitcoin is slightly under one transaction per second [34].

In our simulation, we use a 1000-node network in which each node has an average of 32 peers, transactions are generated with a mean of $\Lambda_{tx} = 1$ s, a duration of 1 hour, and a variable percentage ϵ of Zerocash traffic. To allow for faster experiments, instead of generating a block every 10 minutes as in Bitcoin, we create blocks at an average of every $\Lambda_{block} = 150$ s (as in Litecoin, a popular altcoin).

¹⁸Subsequent optimizations lowered the cost of `VerifyTransaction` below this, after our experiments.

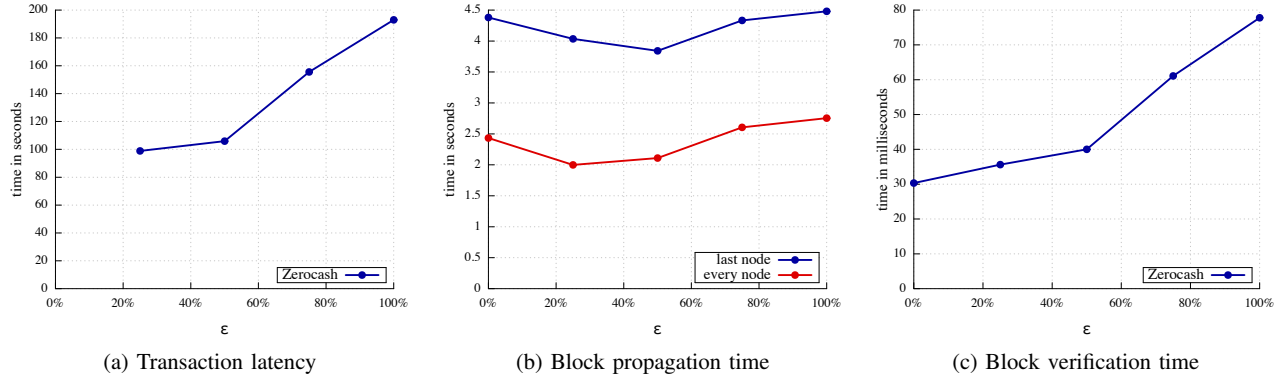


Fig. 5: The average values of the three metrics we study, as a function of ϵ , the percentage of transactions that are Zerocash transactions. Note that, in (a), latency is undefined when $\epsilon = 0$ and hence omitted.

We run our simulation for different traffic mixes, where ϵ indicates the percentage of Zerocash transactions and $\epsilon \in \{0\%, 25\%, 50\%, 75\%, 100\%\}$. Each simulation is run on 200 Amazon EC2 general-purpose `m1.medium` instances, in one region on a `10.10./16` private network. On each instance, we deploy 5 instances of `bitcoind`.

Results. Transactions are triggered by a blocking function call on the simulation control node that must connect to a random node and wait for it to complete sending a transaction. Because the Poisson process modeling transactions generates delays between such calls and not between the exact points when the node actually sends the transactions, the actual transaction rate is skewed. In our experiments the real transaction rate shifts away from our target of one per second to an average of one every 1.4 seconds.

In Figure 5 we plot three metrics for $\epsilon \in \{0\%, 25\%, 50\%, 75\%, 100\%\}$. Each is the average defined over the data from the entire run of the simulation for a given ϵ (i.e., they include multiple transactions and blocks).¹⁹ *Transaction latency* is the interval between a transaction’s creation and its inclusion in a block. *Block propagation time* comes in two flavors: 1) the average time for a new block to reach a node computed over the times for all nodes, and 2) the same average computed over only the last node to see the block.

Block verification time is the average time, over all nodes, required to verify a block. If verification caching was not effective, we would expect to see a marked increase in both block verification time and propagation time. Since blocks occur on average every 150s, and we expect approximately one transaction each second, we should see $150 \times 10 \text{ ms} = 1500 \text{ ms}$ of delay if all transactions were non-cached Zerocash transactions. Instead, we see worst case 80ms and conclude caching is effective. This results in a negligible effect on block propagation (likely because network operations dominate).

The time needed for a transaction to be confirmed, and hence

¹⁹Because our simulated Bitcoin nodes ran on shared EC2 instances, they were subject to variable external load, limiting the benchmark precision. Still, it clearly demonstrates that the mild additional delay does not cause catastrophic network effects.

spendable, is roughly 190s. For slower block generation rates (e.g., Bitcoin’s block every 10 minutes) this should mean users must wait only one block before spending received transactions.

VIII. OPTIMIZATIONS AND EXTENSIONS

See the extended version of this paper [26] for extensions on everlasting anonymity, batched Merkle tree updates, faster block propagation, and scaling to 2^{64} serial numbers.

IX. CONCURRENT WORK

Danezis et al. [19] suggest using zk-SNARKs to reduce proof size and verification time in Zerocoin. Our work differs from [19] in both supported functionality and scalability.

First, [19]’s protocol, like Zerocoin, only supports fixed-value coins, and is best viewed as a decentralized mix. Instead, we define, construct, and implement a full-fledged decentralized electronic currency, which provides anonymous payments of any amount.

Second, in [19], the complexity of the zk-SNARK generator, prover, and verifier all scale superlinearly in the number of coins, because their arithmetic circuit computes, *explicitly*, a product over all coins. In particular, the number of coins “mixed together” for anonymity cannot be large. Instead, in our construction, the respective complexities are polylogarithmic, polylogarithmic, and constant in the number of coins; our approach supports a practically-unbounded number of coins.

X. CONCLUSION

Decentralized currencies should ensure a user’s privacy from his peers when conducting legitimate financial transactions. Zerocash provides such privacy protection, by hiding user identities, transaction amounts, and account balances from public view. This, however, may be criticized for hampering accountability, regulation, and oversight. Yet, Zerocash need not be limited to enforcing the basic monetary invariants of a currency system. The underlying zk-SNARK cryptographic proof machinery is flexible enough to support a wide range of policies. It can, for example, let a user prove that he paid his due taxes on all transactions *without* revealing those transactions, their amounts, or even the amount of taxes paid. As long

as the policy can be specified by efficient nondeterministic computation using NP statements, it can (in principle) be enforced using zk-SNARKs, and added to Zerocash. This can enable privacy-preserving verification and enforcement of a wide range of compliance and regulatory policies that would otherwise be invasive to check directly or might be bypassed by corrupt authorities. This raises research, policy, and engineering questions over what policies are desirable and practically realizable.

Another research question is what new functionality can be realized by augmenting the capabilities already present in Bitcoin's scripting language with zk-SNARKs that allow fast verification of expressive statements.

ACKNOWLEDGMENTS

We thank Amazon for their assistance and kind donation of EC2 resources, and Gregory Maxwell for his advice regarding the Bitcoin codebase. We thank Iddo Ben-Tov and the SCIPR Lab members — Daniel Genkin, Lior Greenblat, Shaul Kfir, Gil Timnat and Michael Riabzev — for inspiring discussions.

This work was supported by: Amazon.com through an AWS in Education research grant; the Broadcom Foundation and Tel Aviv University Authentication Initiative; the Center for Science of Information (CSoI), an NSF Science and Technology Center, under grant agreement CCF-0939370; the Check Point Institute for Information Security; the U.S. Defense Advanced Research Projects Agency (DARPA) and the Air Force Research Laboratory (AFRL) under contract FA8750-11-2-0211; the European Community's Seventh Framework Programme (FP7/2007-2013) under grant agreement number 240258; the Israeli Centers of Research Excellence I-CORE program (center 4/11); the Israeli Ministry of Science and Technology; the Office of Naval Research under contract N00014-11-1-0470; the Simons Foundation, with a Simons Award for Graduate Students in Theoretical Computer Science; and the Skolkovo Foundation under grant agreement 6926059.

The views expressed are those of the authors and do not reflect the official policy or position of the Department of Defense or the U.S. Government.

REFERENCES

- [1] D. Chaum, "Blind signatures for untraceable payments," in *CRYPTO* '82.
- [2] J. Camenisch, S. Hohenberger, and A. Lysyanskaya, "Compact e-cash," in *EUROCRYPT* '05.
- [3] T. Sander and A. Ta-Shma, "Auditable, anonymous electronic cash," in *CRYPTO* '99.
- [4] F. Reid and H. Martin, "An analysis of anonymity in the Bitcoin system," in *SocialCom/PASSAT* '11.
- [5] S. Barber, X. Boyen, E. Shi, and E. Uzun, "Bitter to better - how to make Bitcoin a better currency," in *FC* '12.
- [6] D. Ron and A. Shamir, "Quantitative analysis of the full Bitcoin transaction graph," ePrint 2012/584, 2012.
- [7] G. Maxwell, "CoinJoin: Bitcoin privacy for the real world," August 2013, bitcoin Forum. [Online]. Available: <https://bitcointalk.org/index.php?topic=279249.0>
- [8] I. Miers, C. Garman, M. Green, and A. D. Rubin, "Zerocoin: Anonymous distributed e-cash from bitcoin," in *SP* '13.
- [9] J. Groth, "Short pairing-based non-interactive zero-knowledge arguments," in *ASIACRYPT* '10.
- [10] H. Lipmaa, "Progression-free sets and sublinear pairing-based non-interactive zero-knowledge arguments," in *TCC* '12.
- [11] N. Bitansky, A. Chiesa, Y. Ishai, R. Ostrovsky, and O. Paneth, "Succinct non-interactive arguments via linear interactive proofs," in *TCC* '13.
- [12] R. Gennaro, C. Gentry, B. Parno, and M. Raykova, "Quadratic span programs and succinct NIZKs without PCPs," in *EUROCRYPT* '13.
- [13] B. Parno, C. Gentry, J. Howell, and M. Raykova, "Pinocchio: nearly practical verifiable computation," in *Oakland* '13.
- [14] E. Ben-Sasson, A. Chiesa, D. Genkin, E. Tromer, and M. Virza, "SNARKs for C: verifying program executions succinctly and in zero knowledge," in *CRYPTO* '13.
- [15] H. Lipmaa, "Succinct non-interactive zero knowledge arguments from span programs and linear error-correcting codes," in *ASIACRYPT* '13.
- [16] E. Ben-Sasson, A. Chiesa, E. Tromer, and M. Virza, "Succinct non-interactive arguments for a von Neumann architecture," ePrint 2013/879.
- [17] C. Decker and R. Wattenhofer, "Information propagation in the Bitcoin network," in *P2P* '13.
- [18] E. Ben-Sasson, "Universal and affordable computational integrity," May 2013, bitcoin 2013: The Future of Payments. [Online]. Available: <http://www.youtube.com/watch?v=YRcPRcUpkcU>
- [19] G. Danezis, C. Fournet, M. Kohlweiss, and B. Parno, "Pinocchio Coin: building Zerocoin from a succinct pairing-based proof system," in *PETShop* '13. [Online]. Available: <http://www0.cs.ucl.ac.uk/staff/G.Danezis/papers/DanezisFournetKohlweissParno13.pdf>
- [20] S. Nakamoto, "Bitcoin: a peer-to-peer electronic cash system," 2009. [Online]. Available: <http://www.bitcoin.org/bitcoin.pdf>
- [21] M. Bellare, A. Boldyreva, A. Desai, and D. Pointcheval, "Key-privacy in public-key encryption," in *ASIACRYPT* '01.
- [22] D. Boneh and X. Boyen, "Secure identity based encryption without random oracles," in *CRYPTO* '04.
- [23] R. Gennaro, "Multi-trapdoor commitments and their applications to proofs of knowledge secure under concurrent man-in-the-middle attacks," in *CRYPTO* '04.
- [24] C. Gentry and D. Wichs, "Separating succinct non-interactive arguments from all falsifiable assumptions," in *STOC* '11.
- [25] N. Bitansky, R. Canetti, A. Chiesa, and E. Tromer, "From extractable collision resistance to succinct non-interactive arguments of knowledge, and back again," in *ITCS* '12.
- [26] E. Ben-Sasson, A. Chiesa, C. Garman, M. Green, I. Miers, E. Tromer, and M. Virza, "Zerocash: Decentralized anonymous payments from Bitcoin (extended version)," Cryptology ePrint Archive, 2014.
- [27] J. Katz and Y. Lindell, *Introduction to Modern Cryptography*. Chapman & Hall/CRC, 2007.
- [28] M. Bellare, "New proofs for NMAC and HMAC: security without collision-resistance," in *CRYPTO* '06.
- [29] P. Wuille, "Proposed BIP for dealing with malleability," Available at <https://gist.github.com/sipa/8907691>, 2014.
- [30] V. Shoup, "A proposal for an ISO standard for public key encryption (version 2.1)," *IACR E-Print Archive*, 2001.
- [31] Certicom Research, "SEC 1: Elliptic curve cryptography," 2000. [Online]. Available: http://www.secg.org/collateral/sec1_final.pdf
- [32] R. Dingledine, N. Mathewson, and P. Syverson, "Tor: the second-generation onion router," in *Security* '04.
- [33] G. Danezis, R. Dingledine, and N. Mathewson, "Mixminion: design of a type III anonymous remailer protocol," in *SP* '03.
- [34] T. B. Lee, "Bitcoin needs to scale by a factor of 1000 to compete with Visa. here's how to do it," The Washington Post (<http://www.washingtonpost.com>), November 2013.

Rational Zero: Economic Security for Zerocoin with Everlasting Anonymity

Christina Garman, Matthew Green, Ian Miers, and Aviel D. Rubin

The Johns Hopkins University Department of Computer Science, Baltimore, USA
{cgarman, imiers, mgreen, rubin}@cs.jhu.edu

Abstract. Zerocoin proposed adding decentralized cryptographically anonymous e-cash to Bitcoin. Given the increasing popularity of Bitcoin and its reliance on a distributed pseudonymous public ledger, this anonymity is important if only to provide the same minimal privacy protections from nosy neighbors offered by conventional banking. Unfortunately, at 25KB, the non-interactive zero-knowledge proofs for spending a zerocoin are nearly prohibitively large. In this paper, we consider several improvements. First, we strengthen Zerocoin’s anonymity guarantees, making them independent of the size of these proofs. Given this freedom, we explore several techniques for drastically reducing proof size while ensuring that forging a single zerocoin is more difficult than the block mining process used to maintain Bitcoin’s distributed ledger. Provided a zerocoin is worth less than the reward for a Bitcoin block, forging a coin is not an economically rational action. Hence we preserve Zerocoin’s absolute anonymity guarantees while achieving drastic reductions in proof size by limiting ourselves to security against rational attackers.

Keywords: Privacy, e-cash

1 Introduction

Bitcoin is an electronic currency built atop a distributed transaction ledger. While Bitcoin has achieved widespread success, it has significant weaknesses related to transaction privacy [16, 21]. Zerocoin [17] attempts to address these issues by extending Bitcoin with a new form of anonymous electronic cash. To add privacy while retaining Bitcoin’s decentralized nature, Zerocoin uses a novel construction based on digital commitments and efficient zero-knowledge proofs that a commitment is in a list of commitments. While this construction achieves strong anonymity and prevents double spending, it can incur significant costs. In particular, to achieve cryptographically strong protection against double spending, Zerocoin uses large “spend proofs” that grow rapidly as λ , the resistance of the proofs to forgery, increases. Even for the modest $\lambda = 80$ security level (ensuring forgery effort of 2^{80} operations), Zerocoin spend proofs exceed 25KB. Since these proofs must be stored in the block chain, the large size of these proofs makes it challenging to deploy Zerocoin in practice.

In this work we explore extensions to Zerocoin that may substantially decrease the size of these proofs. Our key observation is a need for revised assumptions.

Zerocoin was designed on the assumption that all proofs must be computationally infeasible to forge. We observe that this requirement is, in a certain sense, an anachronism of cryptographic formalism. For example, in the real world we do not require that physical money be impossible to forge, merely that it be impossible to forge while making a profit. Indeed this is already true of Zerocoin: the Bitcoin block chain, upon which Zerocoin's integrity depends, does not itself provide strong cryptographic guarantees against powerful attackers. Instead, the Bitcoin protocol depends on the weaker assumption that an attacker cannot amass more than 50% of the Bitcoin network's computational power.¹ Thus in some sense, cryptographically unforgeable zerocoins are simply impossible: even if the Zerocoin primitives resist forgery, Bitcoin's block chain can be manipulated to provide the same effect. However, the standard game-based approaches of the type used in the original security analysis of Zerocoin do not provide us any insight into safely reducing the Zerocoin security parameter. Given that this would offer a substantial performance improvements, it is interesting to consider new methods of analysis.

A primary contribution of this paper is a new methodology for examining the computational cost of forging non-interactive zero-knowledge proofs relative to the computational costs of Bitcoin mining. Our main result is as follows: by using the payout from mining a new block as a baseline, we can actually quantify the cost of forging a non-interactive zero-knowledge proof. As a result, we are able to construct game theoretic arguments for Zerocoin's resistance to forgery assuming a rational actor who wishes to profit from forging such a coin.

In and of itself, unfortunately, this new perspective does not allow us to lower the security parameter λ as far as we would like nor, consequently, realize the full reduction in proof size and increase in proof performance. To fully realize these savings, we examine two different techniques for increasing the cost of coin forgery without raising Zerocoin's proof sizes. In our new model, the security parameters are chosen based on economic considerations — such as the value of a zerocoin.

An immediate concern with our new approach is that there exist other factors that cannot be priced as easily as coin forgery. One such factor is the user's *anonymity*. There are no known techniques for pricing the value of a user's long-term transaction privacy, since this price is subjective and may vary from user to user. Moreover, we cannot easily predict the future cost of de-anonymization attacks. Indeed, since Zerocoin transcripts may be retained for long periods of time, the cost of executing an offline attack on a user's anonymity may decrease enormously over time as new computational techniques (e.g., quantum computers) become available. We must be careful in our protocol changes, since even a minor weakening of the zero-knowledge characteristics of Zerocoin's proofs could have significant long-term impact on the anonymity of users. Thus a necessary prerequisite of our above analysis is an explicit separation of Zerocoin's security as a real-world currency from its anonymity as a “pure” cryptographic protocol.

¹ Some recent results raise questions about this 50% number [9].

Fortunately we are able to address this concern in our work. In fact, through some simple enhancements to the Zerocoin protocol, we are able to provide an even stronger guarantee than what is provided by the original Zerocoin paper. Specifically, our new construction ensures that proofs will provide long-term statistical zero-knowledge even when the hash function they are instantiated with proves to be non-ideal, i.e., it behaves very differently from a random oracle.² Our analysis is somewhat unusual in that it applies only to the zero-knowledge property of the proofs; we continue to analyze the soundness of the proofs under the assumption of an ideal hash. The key benefit of our approach is that we are able to retain the efficiency of the original Fiat-Shamir proofs while ensuring that user anonymity is protected over long periods of time. This gives us everlasting anonymity in the common reference string model.

Finally, as an independent contribution, we outline a construction for divisible Zerocoin. The original Zerocoin protocol proposes a new form of electronic cash in which individual coins all have the same value. While the Bitcoin-equivalent value of each zerocoin can be adjusted by protocol convention (and multiple denominations of Zerocoin can be instantiated simultaneously), this property can still be quite restrictive. In this work we show how to modify the Zerocoin protocol to create *divisible* coins, such that every zerocoin can contain an arbitrary individual denomination which may subsequently be “subdivided” into new coins of arbitrary value.

2 Background

2.1 Bitcoin

Bitcoin is a distributed e-cash system that operates without trusted parties or signing authorities. Indeed, the only cryptographic keys necessary for the system to operate are held by individual users and used to authenticate fund transfers.

At a high level, Bitcoin is a set of transaction semantics built on top of a distributed ledger which is known as the block chain. The exact semantics of the transactions are irrelevant here, so for a more detailed discussion of them and the modifications necessary for Zerocoin, we direct the reader to the original Zerocoin paper by Miers et al. [17] or the original Bitcoin paper [19]. Of extreme importance to our proposed modifications to Zerocoin, however, is the mechanism by which Bitcoin’s ledger is maintained. We detail it here.

Consider a version of Bitcoin where there were a fixed number of network nodes. In this case, we could simply have the nodes vote on the correct version of the ledger. Under the assumption that the majority of the nodes are honest, this results in a correct ledger and hence a valid currency system. Effectively, this is the consensus technique used in Byzantine systems. However, Bitcoin is not such

² Specifically, we are concerned with future vulnerabilities in hash functions such as SHA256 that might allow for practical attacks on the zero-knowledge property of Fiat-Shamir proofs. While this concern seems rarified, existing analyses do not allow us to rule out such attacks.

a closed network: anyone can download the software, fire up an instance, and join the network. In particular, one individual can fire up numerous instances and mount a Sybil attack, effectively stuffing the ballot box.

Bitcoin's approach to solving this issue is perhaps most intuitively described as the one-CPU-cycle-one-vote approach. Instead of having each node vote, consider a version of Bitcoin that places a computational requirement on voting and updating consensus. Mounting a Sybil attack would be costly. Bitcoin takes this one step further and instead of voting, actually requires a computationally intensive process to propose an update and has updates accepted only if they add on to the maximally difficult set of updates. Under the assumption that the majority of the computational power of the network is held by honest nodes and the requirements that honest nodes only build updates on valid updates, the longest chain of updates will be the correct consensus value of the ledger. Bitcoin calls this process mining, and we describe it below.

In Bitcoin, each node competes to produce an update to the block chain, known as a block, containing new transactions. The block contains a partial hash collision over 1) the previous block hash (hence block chain), 2) the hash of the transactions, and 3) a nonce. This proof of work is $\mathcal{H}_b(data||nonce) < t$ where t is the difficulty target. The target is picked by the network every two weeks in order to cause the rate at which blocks are created to average 10 minutes given the network's current computational power. As of November 2013, the current difficulty is $609,482,679.89 \approx 2^{29}$. The number of expected hash calculations required to generate a block is given as $difficulty * 2^{32}$. As a result, it takes 2^{61} expected hash calls to generate a single Bitcoin block. Bitcoin uses the double application of SHA256 as its hash function $\mathcal{H}_b$.

Bitcoin, however, goes yet one step further to ensure block chain integrity: a block is not fully trusted until it has a certain number of confirmations (typically six), meaning that there are six blocks on top of it. As a result, the effort required to manipulate a block and completely ensure it stays on the block chain is at least $2^{61} * 6 \approx 2^{63}$ hash calls.

2.2 Zero-Knowledge Proofs

In a zero-knowledge protocol [11] a user (the prover) proves a statement to another party (the verifier) without revealing anything about the statement other than that it is true.

A three-round example of a zero-knowledge protocol is often referred to as a Sigma protocol because Σ represents the flow of the protocol. The three steps can be described in the following manner: 1) commitment, 2) challenge, and 3) response. A popular and well-known example of this is the technique of Schnorr [22], used to prove knowledge of a discrete logarithm. The protocol works as follows (Figure 1):

Given a cyclic group G of order q with generator g and $y = g^x$, prove knowledge of x .

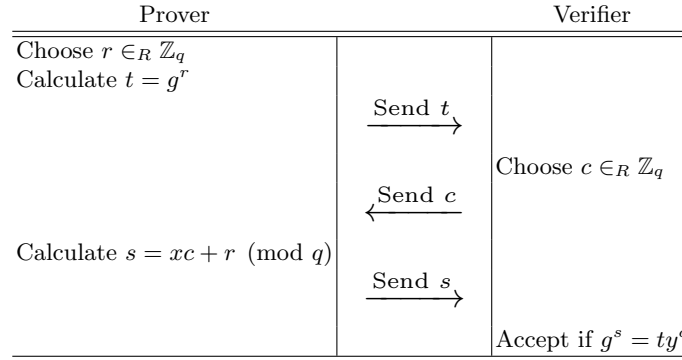


Figure 1: Schnorr protocol for proving knowledge of a discrete logarithm.

While zero-knowledge protocols are normally viewed in the “general cheating verifier” setting, where no matter the strategy of the verifier he learns no additional information, we can also consider the “honest verifier” (or semi-honest verifier) setting. An honest verifier must follow the protocol specifications exactly but maintains the ability to keep a record of the entire interaction [12]. This is of use to us because the Fiat-Shamir heuristic [10] allows us to transform any three-round (Sigma) honest-verifier zero-knowledge protocol into a non-interactive (one-round) zero-knowledge proof of knowledge with the use of a hash function modeled as a random oracle. We demonstrate an example of the application of the Fiat-Shamir heuristic using the Schnorr protocol in Figure 2 below:

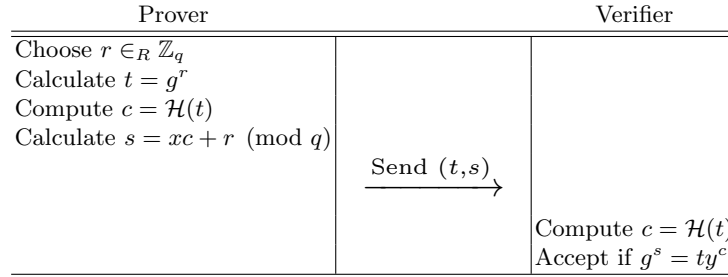


Figure 2: The Fiat-Shamir heuristic as applied to the Schnorr protocol.

When referring to the aforementioned proofs we will use the notation of Camenisch and Stadler [7]. For instance, $\text{NIZKPoK}\{(x, y) : h = g^x \wedge c = g^y\}$ denotes a non-interactive zero-knowledge proof of knowledge of the elements x and y that satisfy both $h = g^x$ and $c = g^y$. All values not enclosed in ()’s are assumed to be known to the verifier.

2.3 Zerocoin

The original Zerocoin protocol added anonymous currency to Bitcoin that was backed by bitcoins. A zerocoin was a commitment to a serial number S . Zerocoins were minted when a user submitted a transaction spending a fixed amount of

bitcoins (e.g., 1 bitcoin) and outputting a new zerocoin. The bitcoins were placed in an escrow pool and the new zerocoin added to a list of all zerocoins. Zerocoins could be spent to withdraw the same fixed bitcoins from the escrow pool by revealing the serial number of the coin and proving it came from the list of coins. This proof was examined by the distributed network running Bitcoin and, if valid and the serial number unused, the correct amount of bitcoins were transferred. Specifically, the proof was a zero-knowledge proof that 1) some coin had that serial number and 2) that that coin was on the list of minted coins. Because the proof is zero-knowledge, any given coin spend cannot be traced to its withdrawal and hence is anonymous.

The naive version of this proof, instantiated as “either this coin, or this coin, or this coin, or ...”, is of size $O(n)$. The principal cryptographic contribution of the original paper was finding a compact representation of the list of coins that still admitted a commitment scheme containing a serial number. Miers et al. accomplished this by using a cryptographic accumulator [3] to represent the list of coins as one group element, a proof due to Camenisch and Lysyanskaya [6] to prove that a committed value is accumulated, and finally a double discrete log proof [8] to prove that the committed value is actually a commitment to a serial number. This results in a proof that is constant size regardless of the number of coins on the list.

Unfortunately, the double discrete log proof is constructed using cut-and-choose methods which effectively repeat a single proof multiple times to decrease the probability of forgery. As a result, the proof is of size $\lambda \cdot 2k$ where k is the size of a single field element and λ is the soundness parameter of the proof. For 1024 bit commitments and an 80 bit security level, this results in a 20KB double discrete log proof and a total proof size (including the accumulator proof) of 25KB. Moreover, single threaded runtime for both verification and generation of the proof runs in $O(\lambda \cdot k)$.

Finally, as the proofs for spending a zerocoin need to be publicly verifiable to allow the withdrawal of bitcoins from the escrow pool, they must be non-interactive. To accomplish this, Zerocoin uses the Fiat-Shamir heuristic to transform the above interactive proofs into non-interactive ones. Moreover, the proof is actually used as a signature of knowledge, not just spending a coin, but also signing the Bitcoin address where the withdrawn bitcoins should be deposited.

3 Everlasting Anonymity

The original zero-knowledge proofs in Zerocoin were non-interactive Fiat-Shamir proofs where both the soundness and zero-knowledge property held only in the random oracle model. This is a rather large concern since, at some point in the future, it seems likely SHA256 will be broken in a way that makes it utterly unsuitable for instantiating a random oracle, just as MD5 and MD2 have been broken. Old Zerocoin proofs using that function will still be around, and their anonymity should be preserved if possible. Intuitively, this should not be an issue, however, absent further analysis, one cannot be sure anonymity is maintained.

Rather than attempting such an analysis, we take the expedient of detailing a simple modification to the proofs that, while still only achieving soundness in the random oracle model, achieves at least statistical zero-knowledge in the common reference string model. In the original (non-interactive) proofs, the challenge (i.e., the second move in a standard three-way “sigma” interactive zero-knowledge proof) was obtained by hashing what would have been the first move in the interactive version. In the random oracle model, a simulator can program a hash function to output arbitrary results. Accordingly, such a simulator could induce a verifier to accept a “proof” even though the simulator knew no witness to the statement being proved. Thus the original proof was zero-knowledge. Obviously when instantiated with an actual hash function, this property no longer strictly holds.

Prover		Verifier
Choose $r \in_R \mathbb{Z}_q$ Calculate $t = g^r$ Compute $c' = \mathcal{H}(t)$ Choose $r' \in_R \mathbb{Z}_q$ Calculate $com = g^{c'} h^{r'} \pmod{p}$ Compute $c = \mathcal{H}(com)$ Calculate $s = xc + r \pmod{q}$	$\xrightarrow{\text{Send } (t, com, r', s)}$	Compute $c' = \mathcal{H}(t)$ Compute $c = \mathcal{H}(com)$ Accept if $g^s = ty^c$ and $com = g^{c'} h^{r'} \pmod{p}$

Figure 3: Dishonest verifier Schnorr Protocol with Fiat-Shamir.

To fix this we propose applying a standard modification for converting from (interactive) honest verifier zero-knowledge proofs to (interactive) non-honest verifier proofs before applying the Fiat-Shamir heuristic: instead of making the first move in the protocol public, first commit to it and then reveal the move only after the challenge is output. Specifically, instead of hashing the first move of the transcript to create a challenge value, we hash a commitment to (the hash of) the first move of the transcript. See Figure 3 for an example using the Schnorr protocol. As a result, any simulator who can control the common reference string can construct the commitment scheme such that they can equivocate and decommit to a first move that satisfies the generated challenge. This is not a typical approach as Fiat-Shamir proofs rely on the random oracle model themselves. However, by using this approach we get proofs that are at least statistical zero-knowledge in the common reference string model, even if soundness still requires the random oracle model, i.e, from the point of view of a privacy critical system, the proofs fail safe.

4 Cost Effective Security Against Forgery and Double Spending

Conceptually, payment systems are subject to three types of attacks: theft of funds, forgery of funds, and double (or more) spending of legitimate attacker controlled funds. These are major issues for both theoretical and extent currency and payment systems, and there are a broad range of solutions which vary considerably in terms of both cost and effectiveness. On one end of the spectrum, e-cash schemes typically avoid all three attacks through the use of secure cryptographic primitives which require a staggeringly prohibitive amount of computational power to break. In contrast, on the decidedly low end of the spectrum, debit cards in the US provide little-to-no security against theft/cloning. Instead they leverage fraud detection and minimization procedures to get the costs of such attacks to acceptable levels without imposing too high an overhead on transactions (e.g., verifying multiple forms of ID for every single transaction).

Certainly, the cryptographic approach is superior provided it is achievable with little overhead. Unfortunately for Zerocoin, it is neither completely achievable nor cheap: as mentioned previously, spends for even modest security parameters reach 25KB and take 0.5 seconds to verify. Moreover, even if Zerocoin was cryptographically secure against such attacks, Bitcoin, upon which it depends, is not. Both double spends and forgery of zerocoins can be accomplished by breaking Bitcoin and without ever touching Zerocoin's underlying cryptographic primitives.

However, the approaches used by centralized credit card companies are antithetical to the decentralized nature of Bitcoin. Moreover, we prefer not to incur the administrative overhead, merchant fees, and chargebacks inherent in the fraud-management approach used by debit cards. Instead we opt for a middle ground: we create cryptographic primitives that are not cost effective to break.

4.1 The Homo-Economicus Security Model

Homo-economicus is a species of rational and narrowly self-interested actors typically found in economic papers. Since our construction provides everlasting anonymity in the common reference string model, we can safely ignore the thorny question of placing a monetary value on privacy and hence safely consider theft, forgery, and double spending attacks under the assumption that our attacker is a member of the species homo-economicus. This leads to a simple security requirement: the expected return from stealing, forging, or double (or more) spending a zerocoin should be less than the expected cost of mounting the required attack. In general, while potentially promising, this model has some large drawbacks. Estimating the real cost of a cryptographic attack is prohibitively difficult, requiring both considerable work in the concrete security model and an accurate cost function for generic computation. The theoretically elegant and simple solution to our problem is not to alter the Zerocoin construction at all. Instead, we would construct a game that, given an attacker who can forge a zerocoin, extracts the computational effort required. One would then assign a

monetary value to this work and ensure it is worth more than the resulting forged coin.

We make no such attempt here. Instead, we model our construction only in the expected number of calls an attacker must make to a hash oracle and use the reward for mining a Bitcoin block to establish the market value of computation. While this approach is inherently linked to Bitcoin, it serves our limited purposes well.

Of course, such a model discounts the possibility of someone who is not financially motivated (e.g., a government) wanting to destroy the currency. While this may be a legitimate concern, we note that an attacker who merely wants to disrupt Zerocoin could also easily attack/block the underlying Bitcoin network and likely at far lower cost.

4.2 Zerocoin Attack Surface

We examine how the choice of various security parameters interacts with attacks on Zerocoin and how to minimize these parameters in light of that. Again, due to everlasting anonymity, we neglect attacks on Zerocoin's anonymity properties.

Theft Actually stealing a user's zerocoin entails spending a coin with the same serial number. Since the Pedersen commitment containing a serial number (i.e., the coin) is information theoretically hiding, an attacker who cannot compromise a user's computer and wallet can only guess blindly. This is a very low probability event and can be made arbitrarily small by increasing the serial number length. If as an absolute minimal bound we assume 512 bit commitments, then we can have 512 bit serial numbers or, in the case of divisible coins, $512 - 64 = 448$ bit serial numbers. A theft probability of 1 in 2^{448} is too small to consider practically and hence we discount theft as a worry.

A second technical consideration for Zerocoin is that proof forgeries can deplete the escrow pool of bitcoins that zerocoins are exchanged for. This would effectively steal someone's coins. A simple solution to this is to operate with no explicit escrow pool, opting instead to destroy bitcoins when minting a zerocoin and create fresh bitcoins when spending one. As a result, forgery of a zerocoin results only in inflation. If forgery is very rare, this is a manageable problem.

Forgery Factoring the accumulator's RSA modulus allows an attacker to forge the coin membership proof and hence forge an unlimited number of coins. This is perhaps the single biggest target in Zerocoin. As a result, we have little choice but to recommend a large modulus, say 3072 bits.

A second avenue for forging a coin is to forge the zero-knowledge proof in a spend. Each such forgery results in one and only one forged coin (since even a forged proof has a unique serial number). As such, we want to make the cost of conducting n forgeries more than the value of n coins. The bulk of the remaining portion of this section will focus on techniques to accomplish this.

Double spending To double spend a coin, one must assign the coin two different serial numbers. This is equivalent to causing the commitment to open to two

separate values. Unfortunately, for simple Pedersen commitments, computing a single discrete log value — $\log_g(h)$ or $\log_h(g)$ — allows this to be done an infinite number of times, again giving us a single point of failure. We will discuss a modification to Pedersen commitments that makes this attack more expensive per instance, though does not eliminate entirely the aggregate effect.

4.3 Raising the Cost of Proof Forgeries

Forging a zero-knowledge proof implies guessing the challenge value prior to starting the protocol. For Fiat-Shamir based non-interaction zero-knowledge proofs, where the challenge is provided by the hash of the first move of the protocol, the only way to do this — assuming the hash function is a random oracle — is to repeatedly query the hash function until you get lucky. If the challenge value has length λ then the probability of forging the proof is $P(f) = 2^{-\lambda}$. Normally for zero-knowledge proofs we choose λ such that $P(f)$ is negligible, and hence, even with a concerted offline attack, a forgery is not feasible.

Suppose it takes b expected evaluations of $\mathcal{H}_B$ to mine a Bitcoin block. If v is the value of a coin and p is the payout from mining a block in terms of reward and collected transaction fees, then we need it to take q expected queries of $\mathcal{H}_B$ to forge the proof such that:

$$\frac{p}{b} > \frac{v}{q}$$

I.e., it pays more per hash calculation to try and mine a block than “mine” a proof forgery. Unfortunately, this analysis yields only a small reduction in the security parameter. The payout for mining a block in terms of transactions fees and the reward is roughly 2^4 .³ Mining such a block at current difficulty levels takes 2^{61} calls to $\mathcal{H}_B$. Assuming a zerocoin is worth one bitcoin, solving the above equation gives us $q = 2^{57}$ and hence $\lambda = 57$.

Proof of work Instead of a simple query to $\mathcal{H}_B$, we can make a single instance of the zero-knowledge proof hash function make a tunable number of calls w to $\mathcal{H}_B$ in much the same manner as PBKDF2. Thus it takes $q = 2^\lambda w$ expected queries to $\mathcal{H}_B$ to forge a proof.

As a result, we end up with a different boundary condition for forgery unprofitability:

$$\frac{(2^\lambda w)p}{b} > v$$

Again assuming the current reward of 25 bitcoins per block plus transaction fees, 2^{61} invocations of $\mathcal{H}_B$ to find a block, and $\lambda = 40$ bit proofs, we end up with approximately $\frac{(2^{40}w)2^4}{2^{61}} > v$. If zerocoins are each worth one bitcoin, this necessitates a value of w of roughly 2^{17} or about 130 thousand hash calls. Since $\mathcal{H}_B$ is the double SHA256 computation used by Bitcoin, we can use the extensive comparisons of Bitcoin mining power across hardware to estimate the cost of this approach. A low end Intel core i3 can compute 1.8 million hashes a second, a now more than a decade old Pentium IV can compute between 0.85 and 1.29

³ This is discounted to allow for lower payouts from, e.g., a mining cartel’s cut.

depending on the model, and an AMR Cortex A-9 such as found in the Samsung Galaxy SII can do 1.3 million hashes a second [1]. As such, this approach is surprisingly viable even for very modest hardware.

This approach has one major limitation: it gets worse as mining difficulty increases, and mining difficulty has been increasing very rapidly as application specific integrated circuits (ASIC) mining hardware comes online. Although one could easily (and should) exclude ASICs from forging proofs via trivial changes to the hash function (e.g., changing the padding or using triple SHA256) that invalidate the ASICs but do not affect hash throughput on a general purpose computer, this does not solve the problem. We can do nothing to address the drop in payout per hash that ASICs introduce by upping the number of hashes needed to mine a block but not changing the reward.⁴ Thus we would still eventually have to increase w beyond levels feasible on non specialized hardware.

Since the first move in the proof reveals nothing and our proofs allow for dishonest verifiers, this computation can be outsourced. However, paying for that outsourcing represents a catch-22: how do you anonymously pay to spend anonymous currency? While there are potential solutions to this involving small anonymous face-to-face Bitcoin transactions as a bootstrapping mechanism, they are less than ideal.

Rate-limiting forgeries A second option that does not place a computational or financial burden on individuals is to rate limit the proof's hash function. To do this, we split the proof over $n + 1$ blocks. The first block encodes the first moves of the protocol. The n^{th} block encodes responses to the challenge value. The λ bit challenge value is generated by taking the first $\frac{\lambda}{n}$ bits from each block of the $1, \dots, n$ blocks and hashing them to produce a challenge. For an honest prover, this entails no additional work (unlike the proof of work system) as they can satisfy the proof for any challenge and thus must merely wait for the block chain to advance before computing the proof. A dishonest prover, on the other hand, must get a specific challenge. As such, they must either mount many parallel attempts each with a different guess at the challenge value or control the block hashes and hence the challenge. The former can be prevented by merely limiting the number of transactions in a block (Bitcoin already effectively does this by limiting the size of a block).

The likelihood that a challenge value is the one guessed is still $2^{-\lambda}$. However, assuming a maximum of 1000 Zerocoin transactions per block, attempts can only be made every half second. If we assume 40 bit security levels for the proofs, we need an expected 2^{40} hash calls and thus making a single forged zerocoin would take 2^{39} seconds or roughly seventeen thousand years. Even at Bitcoin's current unrealized theoretical maximum transaction throughput of seven transactions a second [14] this would still take over 2400 years. This seems both a prohibitive amount of time for mounting an attack and, as a practical matter, an acceptable rate of coin forgery.

⁴ Recall that the difficulty of mining a block adjusts to keep blocks spaced at 10 minute intervals. Hence greater hashing power necessitates more hashes needed per block.

Manipulating the block chain to produce the correct challenge is even more difficult. An attacker must generate far more than n blocks in order to get the correct challenge. They must first generate all n blocks, complete with proof of work for each, and extract the challenge. The overwhelmingly likely case is that the challenge is wrong, and they must repeat the process. If this was done for $n = 2$ blocks and all bits were extracted only from the last block, this would require the attacker to compute 2^λ expected blocks to get the right challenge and hence make $2^{\lambda+61}$ calls to $\mathcal{H}_B$ at Bitcoin's current difficulty. The situation, however, is actually worse than that since the last block only contributes $\frac{\lambda}{n}$ bits as input to the hash function the attacker is trying to get to output the guessed challenge value. Thus the attacker cannot merely generate 2^λ fresh n^{th} blocks knowing that by the pigeonhole principle one of those will result in the right challenge. Instead, they must actually start with a fresh first block and generate the entire sequence before checking if it works.⁵ Not just does this increase the difficulty of mounting such an attack substantially, but because each block depends on the previous one, it adds in a sequential bottleneck that prevents fully parallelizing the attack process. Recall that six blocks is the threshold for normal Bitcoin transactions to be considered confirmed and as such the mere ability to compute six blocks efficiently, let alone $2^\lambda \cdot 6$ blocks, constitutes a massive attack on Bitcoin.

4.4 Raising the Cost of Double Spends

In the original Zerocoin construction of Miers et al., computing a single discrete log of $\log_g(h)$ or $\log_h(g)$ broke the binding property of Pedersen commitments completely and allowed arbitrary double spends. This is undesirable since a single 1024 bit discrete log instance may be in the range of things solvable by a well-funded organization in six months to a year. We wish to avoid such an attack without using larger moduli.

Instead of using a fixed $g, h \in G$ for our commitment group, we hash the serial number into G to select g, h at random using two different hash functions, $\mathcal{H}, \mathcal{H}'$. When spending a coin, we provide these bases in the proof and then the verifier both checks the proof and that the bases result from the hash of the serial number. As a result, assuming $\mathcal{H}, \mathcal{H}'$ are collision resistant, double spends occur exactly once for any given discrete log computation.

We accomplish this by using the hash of the coin serial number S to select g and h at random. This is enforced at verification time by the verifier simply checking that $g = \mathcal{H}(S)$ and $h = \mathcal{H}'(S)$ for the provided public proof inputs. We briefly outline why this modification preserves both the blinding and binding properties of a Pedersen commitment.

Pedersen commitments are information theoretically blinding because for a fixed commitment c and any given value x , there is randomness r that opens

⁵ It is possible to prune some of this work by checking if given, e.g., the first two of n blocks, any assignment of the remaining bits would hash to the correct challenge. We leave to future work the analysis of this strategy along with the best way to skew the sampling of bits from the n blocks to minimize it.

the commitment to that value and all such r values are equally likely, i.e., for a given g^x , there exists an r such that $c = g^x h^r \bmod p$. If we replace h with $h' = \mathcal{H}'(x || pad)$, then we merely shift the randomness r by $\log_{h'}(h)$ and do not change the distribution on r . Hence this still holds.

Pedersen commitments are computationally binding if the discrete log problem is hard. Given a commitment c that opens to two different values x, x' with randomness r, r' , one can compute the discrete log of h with respect to g by substituting in $g^l = h$ and solving $x + lr = x' + lr'$ since $g^x g^{lr} = g^{x'} g^{lr'} = g^{x+lr} = g^{x'+lr'}$. Since g and h are no longer fixed public parameters in our case, we cannot use a single violation of the blinding property to break an instance of the discrete log problem in G . It is probably possible to construct a security proof based on the assumption that the hash function is collision resistant and the discrete log problem is hard. As the rest of our constructions depend on the random oracle model for soundness, we take the expedient of programming the hash function to output the appropriate generators. This is sufficient for our purposes.

Of course, solving l discrete logs in a *fixed* $\mathbb{G}$ is not as hard as solving l discrete logs in *distinct* $\mathbb{G}_1, \dots, \mathbb{G}_l$. The exact security of this appears not to have been well studied. Some preliminary results indicate that for Pollard's Rho algorithm, the difficulty of computing $l < \epsilon \sqrt[3]{N}$ discrete logs is approximately $\sqrt{2NL}$ where N is the order of the group and $0 < \epsilon < 1$ [2]. The far faster class of index calculus methods are still sub-exponential when run on a fixed group. Specifically, they run in $L_p(\frac{1}{2}, \frac{1}{2})$ instead of $L_p(1, \frac{1}{2})$ with a sub-exponential space requirement $L_p(\frac{1}{2}, \frac{1}{2})$ [18]. What this means in practice is an interesting question. We note that both SSH and the Internet Key Exchange protocol used in IPv6 use groups for Diffie-Hellman that are fixed for far longer timespans than we are contemplating.

5 Divisible Cash

The original Zerocoin construction did not make particularly efficient use of the fact that coins are an information theoretically blinding and computationally binding commitment that can contain arbitrary data. These commitments were merely used as a container for a serial number. Yet there are a whole number of techniques for proving far more interesting statements about commitments. These techniques allow us to construct divisible coins. We are aware of an unpublished result that makes this observation in the context of a different Zerocoin construction entirely. Our purpose in this document is not to introduce divisibility but to point out how it can be achieved using the existing cryptographic construction.

Intuition. Instead of a coin being a commitment to a serial number, we propose committing to a serial number S and a balance B . The coin owner can divide the balance B_0 in an existing coin c_0 into two new coins c_1 and c_2 with balances B_1 and B_2 respectively. She does so by creating two new coins, proving that $B_0 = B_1 + B_2$, and revealing the serial number S_0 of the divided coin c_0 . Note

that because we do not reveal the balance of any coin in this construction and by the original Zerocoin construction the spends for the resulting c_1 and c_2 are unlinkable to their minting, we lose nothing by explicitly identifying the original coin c_0 . As such, we do not need to provide the expensive proof used for a spend, we can just identify the coin outright. This results in a highly efficient proof.

The technical question left to answer is how do we encode both the balance and the serial number in the coin? There are two possible constructions:

- We use multi-message commitments where one message is the serial number and one is the balance.
- We encode both the balance and serial number in one value in the commitment.

While conceptually elegant, multi-message commitments are problematic. In the case of Pedersen commitments [20], a commitment to a vector $\mathbf{m}$ of messages n is $(\prod_{i=1}^n g_i^{m_i})h^r$. Since the coin is then $g_1^{m_1}g_2^{m_2}h^r$, the double discrete log proof used for a coin spend must prove knowledge of three exponents instead of two. This adds approximately 10KB to the proof. With the encoding case, we can encode the balance as the l low order bits of the original serial number and use the high $2^{l-\epsilon}$ as the actual serial number. We merely open the coin using the existing spend proof, reveal the encoded value, and then anyone can extract out the serial number and balance.

Dividing a coin c_0 is not as straightforward. We must prove that $B_0 = B_1 + B_2$ and reveal the existing coin's serial number S_0 without revealing anything about the serial numbers for the new coins. We do this as follows:

$$\pi = \text{NIZKPoK}\{(S_1, S_2, r_0, r_1, r_2, B_0, B_1, B_2) : \\ (B_0 = B_1 + B_2) \wedge_{i=0}^2 (c_i = g^{B_i + 2^{l+\epsilon} S_i} h^{r_i} \wedge 0 \leq B_i < 2^l \wedge 0 \leq S_i < 2^l)\}$$

This proof can be accomplished with a variety of standard techniques for efficiently proving range restrictions [4, 5, 13, 15]. The granularity of the ranges these techniques admit vary and will define both the size l of the serial number and balance and space ϵ between the two values.

6 Conclusion

We demonstrate several useful extensions to Zerocoin. First, by removing the random oracle assumption for the zero-knowledge property of the proofs, we get everlasting security in the common reference string model. Second, and most importantly, we provide a means to model the cost of forging a coin and hence allow for cryptographic parameters to be picked to make such forgery uneconomic. As a result, we argue that one can safely reduce the soundness of the proofs from 80 bits to 40, reducing proof size from 25KB to 10KB and nearly halving proof generation and verification time on a single threaded implementation (or increasing throughput on a multithreaded one). The techniques used to accomplish this are specific both to Bitcoin and certain instantiations of hash functions for Fiat-Shamir proofs. We are hopeful future work will provide a general model for game-theoretic security for e-cash.

References

1. Mining hardware comparison. https://en.bitcoin.it/wiki/Mining_hardware_comparison, accessed: 2013-11-23
2. In: Vaudenay, S., Youssef, A. (eds.) *Selected Areas in Cryptography*, Lecture Notes in Computer Science, vol. 2259, pp. 212–229. Springer Berlin Heidelberg (2001), http://dx.doi.org/10.1007/3-540-45537-X_17
3. Benaloh, J., de Mare, M.: One-way accumulators: a decentralized alternative to digital signatures. In: *EUROCRYPT '93*. vol. 765 of LNCS, pp. 274–285 (1994)
4. Boudot, F.: Efficient proofs that a committed number lies in an interval. In: *EUROCRYPT 2000*. pp. 431–444. Springer (2000)
5. Camenisch, J., Chaabouni, R., et al.: Efficient protocols for set membership and range proofs. In: *Advances in Cryptology-ASIACRYPT 2008*, pp. 234–252. Springer (2008)
6. Camenisch, J., Lysyanskaya, A.: An efficient system for non-transferable anonymous credentials with optional anonymity revocation. In: *EUROCRYPT '01*. vol. 2045 of LNCS, pp. 93–118 (2001)
7. Camenisch, J., Stadler, M.: Efficient group signature schemes for large groups. In: *CRYPTO '97*. vol. 1296 of LNCS, pp. 410–424 (1997)
8. Camenisch, J.L.: *Group Signature Schemes and Payment Systems Based on the Discrete Logarithm Problem*. Ph.D. thesis, ETH Zürich (1998)
9. Eyal, I., Sirer, E.G.: Majority is not enough: Bitcoin mining is vulnerable (2013)
10. Fiat, A., Shamir, A.: How to prove yourself: Practical solutions to identification and signature problems. In: *CRYPTO '86* (1986)
11. Goldreich, O., Micali, S., Wigderson, A.: Proofs that yield nothing but their validity and a methodology of cryptographic protocol design. In: *FOCS* (1986)
12. Goldreich, O.: *A short tutorial of zero-knowledge* (2010)
13. Groth, J.: Non-interactive zero-knowledge arguments for voting. In: *Applied Cryptography and Network Security*. pp. 467–482. Springer (2005)
14. Lee, T.B.: Bitcoin needs to scale by a factor of 1000 to compete with Visa. Here's how to do it. Available at <http://www.washingtonpost.com> (November 2013)
15. Lipmaa, H.: On diophantine complexity and statistical zero-knowledge arguments. In: *Advances in Cryptology-ASIACRYPT 2003*, pp. 398–415. Springer (2003)
16. Meiklejohn, S., Pomarole, M., Jordan, G., Levchenko, K., McCoy, D., Voelker, G.M., Savage, S.: A fistful of bitcoins: characterizing payments among men with no names. In: *Internet measurement conference* (2013)
17. Miers, I., Garman, C., Green, M., Rubin, A.D.: Zerocoin: Anonymous distributed e-cash from bitcoin. In: *IEEE Symposium on Security and Privacy* (2013)
18. Mihalcik, J.: *An analysis of algorithms for solving discrete logarithms in fixed groups*. Master's thesis, Navel Post Graduate School (March 2010)
19. Nakamoto, S.: *Bitcoin: A peer-to-peer electronic cash system*, 2009 (2012), <http://www.bitcoin.org/bitcoin.pdf>
20. Pedersen, T.P.: Non-interactive and information-theoretic secure verifiable secret sharing. In: *CRYPTO '92*. vol. 576 of LNCS, pp. 129–140 (1992)
21. Reid, F., Harrigan, M.: An analysis of anonymity in the Bitcoin system. In: *Security and Privacy in Social Networks (SOCIALCOM)* (2011)
22. Schnorr, C.P.: Efficient signature generation for smart cards. *Journal of Cryptology* 4(3), 239–252 (1991)

Replacing a Random Oracle: Full Domain Hash From Indistinguishability Obfuscation

Susan Hohenberger
Johns Hopkins University
susan@cs.jhu.edu

Amit Sahai
UCLA
sahai@cs.ucla.edu

Brent Waters
University of Texas at Austin
bwaters@cs.utexas.edu

January 25, 2014

Abstract

Our main result gives a way to instantiate the random oracle with a concrete hash function in “full domain hash” applications.

The term full domain hash was first proposed by Bellare and Rogaway [BR93, BR96] and referred to a signature scheme from any trapdoor permutation that was part of their seminal work introducing the random oracle heuristic. Over time the term full domain hash has (informally) encompassed a broader range of notable cryptographic schemes including the Boneh-Franklin [BF01] IBE scheme and Boneh-Lynn-Shacham (BLS) [BLS01] signatures.

All of the above described schemes required a hash function that had to be modeled as a random oracle to prove security. Our work utilizes recent advances in indistinguishability obfuscation to construct specific hash functions for use in these schemes. We then prove security of the *original* cryptosystems when instantiated with our specific hash function.

Of particular interest, our work evades the impossibility result of Dodis, Oliveira, and Pietrzak [DOP05], who showed that there can be no black-box construction of hash functions that allow Full-Domain Hash Signatures to be based on trapdoor permutations. This indicates that our techniques applying indistinguishability obfuscation may be useful in the future for circumventing other such black-box impossibility proofs.

Susan Hohenberger is supported for this research effort in part by the National Science Foundation (NSF) CNS-1154035 and CNS-1228443; the Defense Advanced Research Projects Agency (DARPA) and the Air Force Research Laboratory (AFRL) under contract FA8750-11-2-0211, DARPA N11AP20006, the Office of Naval Research under contract N00014-11-1-0470, and a Microsoft Faculty Fellowship. The views expressed are those of the author and do not reflect the official policy or position of DARPA, the National Science Foundation, or the U.S. Government.

Amit Sahai is supported for this research effort in part from a DARPA/ONR PROCEED award, NSF grants 1228984, 1136174, 1118096, 1065276, a Xerox Faculty Research Award, a Google Faculty Research Award, an equipment grant from Intel, and an Okawa Foundation Research Grant. This material is based upon work supported by the Defense Advanced Research Projects Agency through the U.S. Office of Naval Research under Contract N00014-11-1-0389. The views expressed are those of the author and do not reflect the official policy or position of the Department of Defense, the National Science Foundation, or the U.S. Government.

Brent Waters is supported by NSF CNS-0915361 and CNS-0952692, CNS-1228599 DARPA through the U.S. Office of Naval Research under Contract N00014-11-1-0382, DARPA N11AP20006, Google Faculty Research award, the Alfred P. Sloan Fellowship, Microsoft Faculty Fellowship, and Packard Foundation Fellowship. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the Department of Defense or the U.S. Government.

1 Introduction

Since the introduction of the Random Oracle Model by Bellare and Rogaway [BR93], a major effort in cryptography has been to understand when and if random oracles can be instantiated with families of actual hash functions while maintaining security. Over the years, we have seen real progress in this effort: Firstly we have seen the discovery of alternative schemes that do not require random oracles but achieve the same security properties as earlier schemes that do require random oracles. For example, Cramer and Shoup [CS98] achieved efficient chosen ciphertext security from DDH hard groups. As another example Canetti, Halevi, and Katz [CHK07] achieved secure IBE without random oracles, following the seminal work of [BF01] giving IBE in the Random Oracle Model. More recently, we have seen the discovery of schemes that not only work in the standard model without random oracles, but work in a manner very similar to the original schemes that used random oracles (e.g. [HSW13, FHPS13] following schemes in the random oracle model [BF01, BLS01]). However, all of these schemes proven secure without random oracles required *changing the underlying cryptographic scheme* in addition to instantiating the random oracle with a concrete hash function. Thus, despite these advances, the following basic question has remained open:

Can we instantiate the random oracle with an actual family of hash functions for existing cryptographic schemes in the random oracle model, such as Full Domain Hash signatures?

In other words, can we achieve security *without changing the underlying cryptographic scheme* at all, but only by replacing the random oracle with a specific family of hash functions? In this work, we give the first positive answer to this question. We do this by leveraging the notion of indistinguishability obfuscation [BGI⁺01, BGI⁺12] that was recently achieved in the work of [GGH⁺13].

Our result is particularly interesting in light of negative results on the Random Oracle Model [CGH98, GK03, BBP04] which have called into question the secure applicability of the Random Oracle Model. Our work is the first to show natural examples of schemes that were originally invented with the Random Oracle Model in mind, that nevertheless remain secure when the random oracle is specifically instantiated.

In particular, our work evades the impossibility result of Dodis, Oliveira, and Pietrzak [DOP05], who showed that there can be no black-box construction of hash functions that allow Full-Domain Hash Signatures to be based on trapdoor permutations. Because we make use of obfuscation, our constructions are inherently non-black-box, and thus are not ruled out by this type of black-box impossibility result. This indicates that our techniques applying indistinguishability obfuscation may be useful in the future for circumventing other such black-box impossibility proofs.

Our Result. Our main result gives a way to instantiate the random oracle with a concrete hash function in “full domain hash” signatures. The full domain hash signature scheme was first proposed¹ in the original Bellare-Rogaway [BR93] paper as a way to build a signature scheme from any trapdoor permutation using the introduced random oracle heuristic. This work was very influential and formed the foundation for part of the PKCS#1 standard [KS98]. While the terminology of “full-domain hash” (FDH) originally applied to the trapdoor permutation signature scheme of Bellare and Rogaway, over time it has (informally) encompassed a broader range of notable cryptographic schemes including the Boneh-Franklin [BF01] IBE scheme, the Cocks’s IBE scheme [Coc01], and Boneh-Lynn-Shacham (BLS) [BLS01] signatures. Although these schemes exist in different algebraic domains and have different aims, they share common construction and proof structures that uses random oracle programming in very similar ways.

Our work develops a methodology for replacing the programming of a random oracle in these construction using indistinguishable obfuscation in a novel manner. We begin by describing a scheme that replaces the RO hash function in the original Bellare-Rogaway trapdoor permutation (TDP) signature scheme. Our newly instantiated scheme is then proven to be selectively secure.

Let’s begin by informally recalling the Bellare-Rogaway TDP-based full domain hash scheme. The signature setup algorithm generates a trapdoor permutation pair of functions g_{PK}, g_{SK}^{-1} . In addition, it chooses

¹The terminology “full-domain hash” was actually introduced by Bellare-Rogaway in 1996 [BR96]. They applied this label to the noted signature scheme of their earlier work.

a hash function $H(\cdot)$ that maps from the message space to the domain (and co-domain) of the permutation. The permutation g_{PK} and hash function are published as the verification key and the inverse g_{SK}^{-1} is kept secret. To sign a message m , the signer computes $g_{SK}^{-1}(H(m))$. To verify a signature σ on message m , the verifier simply checks whether $g_{PK}(\sigma) \stackrel{?}{=} H(m)$.

The proof of the Bellare-Rogaway FDH system utilizes the random oracle heuristic to model $H(\cdot)$ as a programmable random oracle. Suppose a poly-time attacker makes at most Q_H oracle queries. One can create a reduction algorithm to the security of the trapdoor permutation as follows. For all but one of the (unique) queries of a message m to the oracle, the reduction algorithm chooses a random value t from the domain and outputs $g_{PK}(t)$ as the result of the query. For any of these messages it is easy for the reduction algorithm to generate a signature on. It simply outputs t . However, at one query point m^* it programs the output of the random oracle to be $z^* = g_{PK}(t^*)$ where z^* was given from the trapdoor permutation challenger. If the attacker forges at this message, then the forgery will be t^* which is immediately the solution for the trapdoor permutation inversion.

Our first result is creating a *replacement* hash function for the oracle $H(\cdot)$ and developing a security proof without relying on the random oracle heuristic. To keep with our original goals, our only modifications will be to $H(\cdot)$ and we *will use the signature system construction as is*, with no changes to the underlying trapdoor permutation family. The two main tools we use to build $H(\cdot)$ are an indistinguishability obfuscator [BGI⁺01, GGH⁺13] and a recently introduced [BW13, BGI13, KPTZ13] primitive called constrained PRFs. In short, a constrained PRF key is a secret key K that allows the evaluator to evaluate the a PRF at a limited set of points, while the rest will appear pseudorandom to him. For our results, we only need a simple form of constrained PRFs called “punctured PRFs” [SW13]. In this setting a private key will be associated with a polynomial set S , where a key $K(S)$ can evaluate the PRF $F(K, x)$ at all x except when $x \in S$. For our proofs we only ever need S to be a singleton set.

We now overview the hash function construction and how we prove it to be selectively secure. (One could use the usual complexity leveraging arguments to claim adaptive security, but we will address adaptive security in a direct way shortly.) To create the hash function the reduction algorithm first chooses a puncturable PRF key K (note this “master key” can evaluate the PRF at all points). Next, the hash function itself will be an obfuscation of the program which on input m computes $g_{PK}(F(K, m))$. That is the program simply computes the PRF at point m and then applies the trapdoor permutation. We call this program Full Domain Hash. To prove security we will apply the “punctured programs” method of Sahai and Waters [SW13], where we surgically remove a key element of a program, but in a way that does not alter input/output functionality.

Our security proof is formed from a sequence of hybrids. In the first hybrid, we replace the obfuscation of the program Full Domain Hash with an obfuscation of an equivalent program called Full Domain Hash*. This program operates the same as the original except on input m^* , where m^* is the message the attacker selectively chose to attack (before seeing the verification key). At this point instead of computing $F(K, m^*)$ the program is simply hardwired to output a constant z^* to output where z^* is set to be $F(K, m^*)$. Since $z^* = F(K, m^*)$, the input/output behavior is identical. In addition, the program is not given the full PRF key K , but instead is given a punctured PRF key $K(\{m^*\})$. By the security of indistinguishable obfuscation the advantage of any poly-time attacker must be negligibly close between these hybrids. In the next hybrid experiment we replace z^* with a random value chosen from the domain/range of the permutation. The advantage between of this hybrid must also be close due to the constrained PRF security. Now we are finally in a position where we can reduce to the security of the trapdoor permutation. The reduction algorithm receives a TDP challenge z^* and hardcodes that in as the output of $H(m^*)$. It can use a signature on this to invert the challenge. At all other points it knows the punctured PRF key and can therefore compute valid signatures without knowing the inverse of the trapdoor permutation.

We remark that our reduction actually shares some of the spirit of the original random oracle reduction, where a challenge is programmed in at one point and signatures are made by knowing the pre images at all others. A key aspect is that the obfuscation hides the fact that at a certain hybrid m^* is treated differently. If an attacker were able to see inside the obfuscation it could actually see the preimages and break the scheme. Another interesting aspect is that our proof does not leverage the fact that the function $g_{PK}(\cdot)$ is a

permutation. It would go through equally well if we only assumed that it was an injective trapdoor function.

Overcoming the black-box impossibility We can now see more precisely why our work evades the impossibility result of Dodis, Oliveira, and Pietrzak [DOP05]. Our hash function is obfuscation of code that runs the underlying permutation. The obfuscation will intuitively hide the evaluation of this code. In particular, no attacker can tell if the trapdoor permutation was actually computed on an input or whether it was a special point where the output was hardcoded in. In the DOP negative result, they build an attack oracle that specifically leverages the black box access to the TDP to watch whenever it is called. It is interesting to see this very strong correlation between the negative result and how non-black box access to a primitive and indistinguishability obfuscation can combine to circumvent it.

Getting Adaptive Security. For our next result we show how to get adaptive (or standard) signature security without complexity leveraging for the case where the trapdoor permutation is the RSA function. The use of RSA as a trapdoor permutation candidate was suggested in Bellare-Rogaway'93 [BR93] and explicitly given in Bellare-Rogaway'96 [BR96]. The public parameters are an RSA modulus $N = pq$ for hidden primes p, q and an RSA exponent e chosen such that $\gcd(\phi(N), e) = 1$. The secret key is the integer d where $d \cdot e = 1 \pmod{\phi(N)}$. A signature on message m is of the form $H(m)^d \pmod{N}$ and one verifies a signature σ by checking if $H(m) \stackrel{?}{=} \sigma^e \pmod{N}$.

We develop a different set of techniques that can leverage the particular structure of the RSA function. The first new ingredient is use of admissible hash functions first introduced in the context of Identity-Based Encryption by Boneh-Boyen [BB04a]. We use a simplification due to Freire et. al. [FHPS13]. At a high level the system is a pair of a hash function $h : \{0, 1\}^{\ell(\lambda)} \rightarrow \{0, 1\}^{n(\lambda)}$ that hashes from the message space to n bit strings and an efficient randomized algorithm AdmSample . The sampling algorithm takes in the security parameter as well as second parameter Q which intuitively corresponds to the number of signature queries an attacker makes. It outputs a string $u \in \{0, 1, \perp\}^n$. Informally, we say that the system is admissible if the following conditions hold. Consider any sequence of Q values $x_1, \dots, x_Q$ and $x^* \neq x_i$. The event we consider is where the string $h(x_i)$ has a bit in common with u in at least one position, but $h(x^*)$ is different from u at all positions. (Note, if $u_j = \perp$ then it is different at position j from all bit strings.) If this event occurs with non-negligible probability, we say it is an admissible system. Intuitively, when used in a proof of a signature scheme, the admissible hash function is utilized to partition the message space into messages that can be signed in the query phase and those that can be used in the challenge phase. A sampled string u corresponds to a particular partition. When running a reduction, one hopes that the actual signature oracle queries and forgery message align with a partition, and the reduction aborts otherwise.

To build the hash function candidate, the setup first chooses a random $v \in \mathbb{Z}_N^*$ as well as exponents $a_{i,b}$ chosen randomly in $[0, \phi(N)]$, for all $i \in [1, n], b \in \{0, 1\}$. Next, it builds the hash function as an obfuscation of the program RSA Hash . The program will first compute $m' = h(m)$. Then, it computes and outputs $v^{\prod_{i \in [n]} a_{i,m'_i}}$.

Our proof proceeds in a few hybrid steps. In the first hybrid experiment the challenger creates a partition internally by calling $\text{AdmSample}(1^\lambda, Q) \rightarrow u$ for an attacker that makes at most $Q = Q(\lambda)$ queries. The game aborts and declares the attacker unsuccessful if any of the query messages or forgery message violates the partition. The property of admissible hashes states any attacker with non-negligible advantage in the real game will also have non-negligible advantage here. In the next hybrid, we change the way we sample the exponents $a_{i,b}$. One first chooses random $y_{i,b} \in [1, N]$. Then for when $u_i = b$ we set $c_{i,b} = e \cdot y_{i,b}$. If $u_i \neq b$ we set $c_{i,b} = e \cdot y_{i,b} + 1$. Note in the first case $c_{i,b}$ is a multiple of e and in the second case $e \nmid c_{i,b}$. The values $a_{i,b} = c_{i,b} \pmod{\phi(N)}$. We show that this way of choosing a values is statistically close to the previous uniform way, because $\gcd(\phi(N), e) = 1$.

Next, we use an alternative program where we directly use the $c_{i,b}$ values in place of the $a_{i,b}$ values. Since the group $\mathbb{Z}_N^*$ is of order $\phi(N)$ we have that $v^{\prod_{i \in [n]} a_{i,m'_i}} = v^{\prod_{i \in [n]} c_{i,m'_i}}$ for all m' . Therefore the input/output behavior is the same between the two programs and we can argue the advantage in the hybrids for poly-time attackers must be close by indistinguishability obfuscation. This is the critical hybrid experiment

in that it most radically departs from previous such proofs, by leveraging indistinguishability obfuscation. Observe that this hybrid experiment eliminates the need for the reduction to know $\phi(N)$, which is crucial to the reduction, since it uses $c_{i,b}$ values instead of $a_{i,b}$ values. However, if the values $c_{i,b}$ were completely visible to an attacker, they would be trivially distinguishable from the “true” uniform $a_{i,b}$ values. However, indistinguishability obfuscation guarantees that these values are hidden from the attacker, and that indeed the attacker cannot distinguish this hybrid from the previous one.

Finally, we show that any attacker that is successful in the last hybrid can be used to break the RSA assumption. For any signature query message m that respects the partition, the reduction will view $H(m)$ as v raised to some integer that is a multiple of e and taking the e -th root is then easy. Any forgery on m^* that respects the partition, the reduction will view $H(m^*)$ as v^z for some z where $\gcd(e, z) = 1$ and from this can derive $v^{1/e}$.

BLS Signatures and More We extend our techniques to replacing the random oracle in the BLS [BLS01] signature scheme. In Section 5 we give a candidate that has a selective proof of security based on the computational Diffie-Hellman problem (along with indistinguishability obfuscation). In Section 6, we give an adaptive proof of security based on an assumption equivalent to the n -Diffie-Hellman inversion assumption. The high level structures of these are similar to the respective selective and adaptive construction and proof methods above. The lower level mechanisms are adapted to the context of bilinear groups. Finally, in Section 7 we sketch how the BLS ideas extend to the Boneh-Franklin IBE scheme.

1.1 Other Related Work

Recently, the work of [BHK13] looked at a complementary question of identifying a definitional abstraction to replace the random oracle heuristic in many random oracle-based constructions. The abstraction is a notion of security called UCE (Universal Computational Extractor). The authors emphasize that a random oracle is *known* not to exist and “behaves like a random oracle” is not a rigorously defined property, whereas UCE is a well defined property of a hash function. They then show how several previous constructions proven secure in the random schemes can be proven secure if we assume the hash functions are UCE secure. One can then conjecture that standard cryptographic hash functions like SHA-256 may satisfy the UCE security notion. In contrast, our work is focused on providing *new* candidate constructions for hash functions, that allow for a security proof to work with the original constructions in the random oracle model. Interestingly, the work of [BHK13] does not encompass the case of Full Domain Hash signatures, arguably one of the most natural and well-studied constructions in the Random Oracle Model, that we address here.

2 Preliminaries

In this section, we define indistinguishability obfuscation, and variants of pseudo-random functions (PRFs) that we will make use of. All the variants of PRFs that we consider will be constructed from one-way functions.

2.1 Indistinguishability Obfuscation

The definition below is from [GGH⁺13]; there it is called a “family-indistinguishable obfuscator”, however they show that this notion follows immediately from their standard definition of indistinguishability obfuscator using a non-uniform argument.

Definition 1 (Indistinguishability Obfuscator ($i\mathcal{O}$)). A uniform PPT machine $i\mathcal{O}$ is called an *indistinguishability obfuscator* for a circuit class $\{\mathcal{C}_\lambda\}$ if the following conditions are satisfied:

- For all security parameters $\lambda \in \mathbb{N}$, for all $C \in \mathcal{C}_\lambda$, for all inputs x , we have that

$$\Pr[C'(x) = C(x) : C' \leftarrow i\mathcal{O}(\lambda, C)] = 1$$

- For any (not necessarily uniform) PPT adversaries $Samp, D$, there exists a negligible function α such that the following holds: if $\Pr[\forall x, C_0(x) = C_1(x) : (C_0, C_1, \tau) \leftarrow Samp(1^\lambda)] > 1 - \alpha(\lambda)$, then we have:

$$\left| \Pr [D(\tau, i\mathcal{O}(\lambda, C_0)) = 1 : (C_0, C_1, \tau) \leftarrow Samp(1^\lambda)] - \Pr [D(\tau, i\mathcal{O}(\lambda, C_1)) = 1 : (C_0, C_1, \tau) \leftarrow Samp(1^\lambda)] \right| \leq \alpha(\lambda)$$

In this paper, we will make use of such indistinguishability obfuscators for all polynomial-size circuits:

Definition 2 (Indistinguishability Obfuscator for $P/poly$). A uniform PPT machine $i\mathcal{O}$ is called an *indistinguishability obfuscator* for $P/poly$ if the following holds: Let $\mathcal{C}_\lambda$ be the class of circuits of size at most λ . Then $i\mathcal{O}$ is an indistinguishability obfuscator for the class $\{\mathcal{C}_\lambda\}$.

Such indistinguishability obfuscators for all polynomial-size circuits were constructed under novel algebraic hardness assumptions in [GGH⁺13].

2.2 Constrained PRFs

We first consider some simple types of constrained PRFs [BW13, BGI13, KPTZ13], where a PRF is only defined on a subset of the usual input space. We focus on *puncturable* PRFs, which are PRFs that can be defined on all bit strings of a certain length, except for any polynomial-size set of inputs:

Definition 3. A *puncturable* family of PRFs F mapping is given by a triple of Turing Machines Key_F , Puncture_F , and Eval_F , and a pair of computable functions $n(\cdot)$ and $m(\cdot)$, satisfying the following conditions:

- **[Functionality preserved under puncturing]** For every PPT adversary A such that $A(1^\lambda)$ outputs a set $S \subseteq \{0, 1\}^{n(\lambda)}$, then for all $x \in \{0, 1\}^{n(\lambda)}$ where $x \notin S$, we have that:

$$\Pr [\text{Eval}_F(K, x) = \text{Eval}_F(K_S, x) : K \leftarrow \text{Key}_F(1^\lambda), K_S = \text{Puncture}_F(K, S)] = 1$$

- **[Pseudorandom at punctured points]** For every PPT adversary (A_1, A_2) such that $A_1(1^\lambda)$ outputs a set $S \subseteq \{0, 1\}^{n(\lambda)}$ and state τ , consider an experiment where $K \leftarrow \text{Key}_F(1^\lambda)$ and $K_S = \text{Puncture}_F(K, S)$. Then we have

$$\left| \Pr [A_2(\tau, K_S, S, \text{Eval}_F(K, S)) = 1] - \Pr [A_2(\tau, K_S, S, U_{m(\lambda) \cdot |S|}) = 1] \right| = \text{negl}(\lambda)$$

where $\text{Eval}_F(K, S)$ denotes the concatenation of $\text{Eval}_F(K, x_1), \dots, \text{Eval}_F(K, x_k)$ where $S = \{x_1, \dots, x_k\}$ is the enumeration of the elements of S in lexicographic order, $\text{negl}(\cdot)$ is a negligible function, and U_ℓ denotes the uniform distribution over ℓ bits.

For ease of notation, we write $F(K, x)$ to represent $\text{Eval}_F(K, x)$. We also represent the punctured key $\text{Puncture}_F(K, S)$ by $K(S)$.

The GGM tree-based construction of PRFs [GGM84] from one-way functions are easily seen to yield puncturable PRFs, as recently observed by [BW13, BGI13, KPTZ13]. Thus we have:

Theorem 1. [GGM84, BW13, BGI13, KPTZ13] If one-way functions exist, then for all efficiently computable functions $n(\lambda)$ and $m(\lambda)$, there exists a puncturable PRF family that maps $n(\lambda)$ bits to $m(\lambda)$ bits.

Next we consider families of PRFs that are with high probability injective:

2.3 RSA Assumption and Shamir's Lemma

We begin by recalling (one of the) standard versions of the RSA assumption [RSA78].

Assumption 1 (RSA). Let λ be the security parameter. Let positive integer N be the product of two λ -bit, distinct odd primes p, q . Let e be a randomly chosen positive integer less than and relatively prime to $\phi(N) = (p-1)(q-1)$. Given (N, e) and a random $y \in \mathbb{Z}_N^*$, it is hard to compute x such that $x^e \equiv y \pmod{N}$.

We also make use of the following lemma due to Shamir.

Lemma 1 (Shamir [Sha83]). Given $x, y \in \mathbb{Z}_N$ together with $a, b \in \mathbb{Z}$ such that $x^a = y^b \pmod{N}$ and $\gcd(a, b) = 1$, there is an efficient algorithm for computing $z \in \mathbb{Z}_N$ such that $z^a = y \pmod{N}$.

2.4 Bilinear Groups and the CDH Assumption

Let $\mathbb{G}$ and $\mathbb{G}_T$ be groups of prime order p . A *bilinear map* is an efficient mapping $e : \mathbb{G} \times \mathbb{G} \rightarrow \mathbb{G}_T$ which is both: (*bilinear*) for all $g \in \mathbb{G}$ and $a, b \leftarrow \mathbb{Z}_p$, $e(g^a, g^b) = e(g, g)^{ab}$; and (*non-degenerate*) if g generates $\mathbb{G}$, then $e(g, g) \neq 1$.

Assumption 2 (Computational Diffie-Hellman). Let g generate a group $\mathbb{G}$ of prime order $p \in \Theta(2^\lambda)$. For all p.p.t. adversaries $\mathcal{A}$, the following probability is negligible in λ :

$$\Pr[a, b \leftarrow \mathbb{Z}_p; z \leftarrow \mathcal{A}(g, g^a, g^b) : z = g^{ab}].$$

2.5 The n -Diffie-Hellman Inversion Assumption and Equivalent Formulation

Our Section 6 construction of adaptively secure BLS signatures makes use of the n -Diffie-Hellman Inversion assumption [BB04b]. This is a parameterized family of assumptions, where the number of group elements involved increases with n . (In our application of Section 6 n will be dependent only on the security parameter.)

Assumption 3 (n -Diffie-Hellman Inversion). Let h generate a group $\mathbb{G}$ of prime order $p \in \Theta(2^\lambda)$. For all p.p.t. adversaries $\mathcal{A}$, the following probability is negligible in λ :

$$\Pr[b \leftarrow \mathbb{Z}_p; z \leftarrow \mathcal{A}(h, h^b, h^{b^2}, \dots, h^{b^n}) : z = g^{1/b}].$$

We will actually use an equivalent assumption, which is easier to work with in our proof. We call the assumption the n -DHI Equivalent assumption for the purposes of this paper. The assumption is stated as follows.

Assumption 4 (n -DHI Equivalent). Let g generate a group $\mathbb{G}$ of prime order $p \in \Theta(2^\lambda)$. For all p.p.t. adversaries $\mathcal{A}$, the following probability is negligible in λ :

$$\Pr[a \leftarrow \mathbb{Z}_p; z \leftarrow \mathcal{A}(g, g^a, g^{a^2}, \dots, g^{a^n}) : z = g^{a^{n+1}}].$$

We briefly sketch why an attacker $\mathcal{A}$ on the new assumption implies an attacker on the n -DHI assumption. Suppose, that an algorithm $\mathcal{B}$ is given a DHI instance $h, h^b, h^{b^2}, \dots, h^{b^n}$. Then, it creates an instance for $\mathcal{A}$ by setting $g = h^{b^n}$, $g^a = g^{b^{n-1}}$, $\dots$, $g^{a^n} = h$. Essentially, this sets $g = h^{b^n}$ and implicitly $a = b^{-1}$. Therefore, $g^{a^{n+1}} = h^{1/b}$. Thus an efficient attacker to the n -DHI Equivalent problem implies one to the n -DHI problem. The other direction of equivalence follows in an analogous manner.

3 Full-Domain Hash Signatures (Selectively Secure)

In this section, we revisit the Bellare-Rogaway Full-Domain Hash (FDH) signature scheme [BR93, BR96], and show how to make it selectively secure in the standard model by instantiating the random oracle in a specific way. We stress that we do not modify the Bellare-Rogaway FDH signature scheme in any way; the only new aspect of our construction is our instantiation of the random oracle with a specific function whose description becomes part of the public key.

Recall that the Bellare-Rogaway FDH signature scheme required a trapdoor permutation family. Our method, in fact, not only applies to trapdoor permutation families, but indeed to any *injective* trapdoor function family. We prove the selective security of the FDH signature scheme based on the security of the indistinguishability obfuscator, the security of a puncturable PRF family, and the security of an injective trapdoor function family.

For simplicity of exposition, we assume that there is a polynomial $\ell(\lambda)$ which denotes the length of messages to be signed; we denote this message space by $\mathcal{M} = \{0, 1\}^{\ell(\lambda)}$. More generally, a collision-resistant hash function may be used to hash messages to this size.

- **Setup**(1^λ) : The setup algorithm first runs $\text{TDFSetup}(1^\lambda)$ and that produces a public index PK along with a trapdoor SK, yielding the map $g_{\text{PK}} : \{0, 1\}^n \rightarrow \{0, 1\}^w$ together with its inverse. Next, the setup algorithm chooses a puncturable PRF key K for F where $F(K, \cdot) : \{0, 1\}^{\ell(\lambda)} \rightarrow \{0, 1\}^n$. Then, it creates an obfuscation of the of the program Full Domain Hash Figure 1. The size of the program is padded to be the maximum of itself and the program Full Domain Hash* of Figure 2. We refer to the obfuscated program as the function $H : \{0, 1\}^{\ell(\lambda)} \rightarrow \{0, 1\}^w$, which acts as the random oracle type hash function in the Bellare-Rogaway scheme.

The verification key VK consists of the trapdoor index PK as well as the hash function $H(\cdot)$. The secret key is the trapdoor SK as well as $H(\cdot)$.

- **Sign**(SK, $m \in \mathcal{M}$) : The signature algorithm outputs $\sigma = g_{\text{SK}}^{-1}(H(m)) \in \{0, 1\}^n$.
- **Verify**(VK, m, σ) The verification algorithm tests if $g_{\text{PK}}(\sigma) \stackrel{?}{=} H(m)$ and outputs accept if and only if this holds.

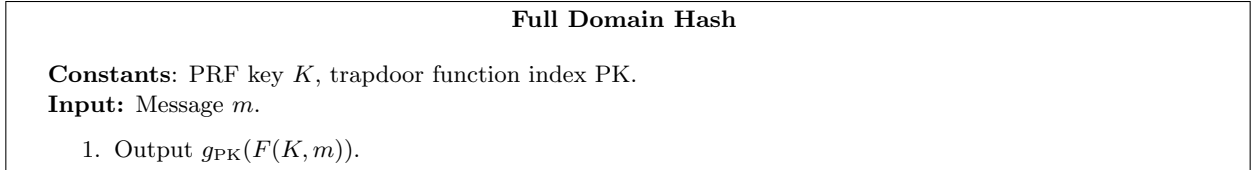


Figure 1: Full Domain Hash

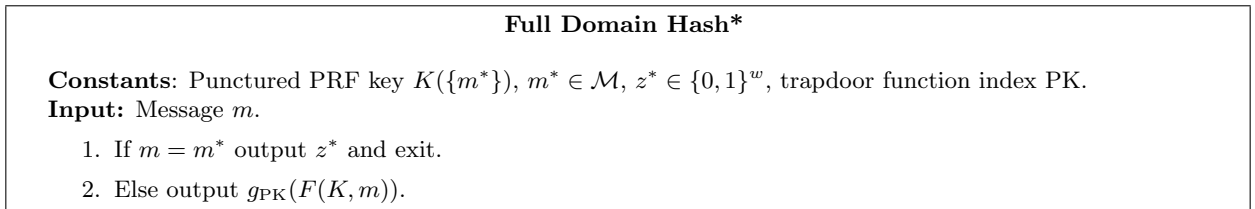


Figure 2: Full Domain Hash*

Theorem 2. If our obfuscation scheme is indistinguishably secure, F is a secure punctured PRF, and the injective trapdoor function is secure, then the above signature scheme is selectively secure.

We describe a proof as a sequence of hybrid experiments where the first hybrid corresponds to the original signature security game. We prove that a poly-time attacker's advantage must be negligibly close between

each successive one. Then, we show that any poly-time attacker in the final experiment that succeeds in forging with non-negligible probability can be used to invert the injective trapdoor function.

- **Hyb₀** : In the first hybrid the following game is played:
 1. The attacker selectively gives the challenger the message m^* .
 2. The TDF index is chosen by the challenger running $\text{TDFSetup}(1^\lambda)$.
 3. K is chosen as a key for the puncturable PRF.
 4. The hash function $H(\cdot)$ is created as an obfuscation of the program Full Domain Hash.
 5. The attacker queries the sign oracle a polynomial number of times on messages $m \neq m^*$. It receives back $g_{\text{SK}}^{-1}(H(m)) = F(K, m)$. (Note the equality holds since the function g_{PK} is injective.)
 6. The attacker sends a forgery σ^* and wins if $\text{Verify}(\text{VK}, m^*, \sigma^*) = 1$.
- **Hyb₁** : Is the same as **Hyb₀** except we let $z^* = g_{\text{PK}}(F(K, m^*))$ and let VK be the obfuscation of the program $\text{Verify Signature}^*$ of Figure 2.
- **Hyb₂** : Is the same as **Hyb₁** except $z^* = g_{\text{PK}}(t)$ for t chosen uniformly at random in $\{0, 1\}^n$.

Lemma 2. If our obfuscation scheme is indistinguishability secure, then the advantage of a poly-time attacker in **Hyb₀** is negligibly close to the advantage in **Hyb₁**.

Proof. We prove this lemma by giving a reduction to the indistinguishability security of the obfuscator. To do so, we must build the two algorithms *Samp* and *D*.

Samp(1^λ) behaves as follows: It invokes the adversary to obtain m^* and the adversary's state τ' . It runs $\text{TDFSetup}(1^\lambda)$ to obtain PK and SK . It then chooses K as a key for the puncturable PRF. It sets $z^* = g_{\text{PK}}(F(K, m^*))$. It sets $\tau = (m^*, \text{PK}, \text{SK}, K, \tau')$ and builds C_1 as the program for Full Domain Hash, and C_2 as the program for Full Domain Hash*.

Before describing *D*, we observe that by construction and the functionality preservation property of puncturable PRFs, the circuits C_1 and C_2 always behave identically on every input. Because of padding, both C_1 and C_2 have the same size. Thus, *Samp* satisfies the conditions needed for invoking the indistinguishability property of the obfuscator.

Now, we can describe the algorithm *D*, which takes as input τ as given above, and either the obfuscation of C_1 , which is the program Full Domain Hash, or C_2 , which is the program Full Domain Hash*. *D* creates the verification key for the signature scheme by combining PK with the obfuscated program as the hash function description. It then invokes the adversary on this verification key, and the adversary then makes requests for signatures on messages $m \neq m^*$. For each such message, *D* constructs the signatures $g_{\text{SK}}^{-1}(H(m)) = F(K, m)$, through its knowledge of K within τ . Finally, the attacker sends a forgery σ^* and wins if $\text{Verify}(m^*, \sigma^*) = 1$. If the attacker wins, *D* outputs 1.

By construction, if *D* receives an obfuscation of C_1 , then the probability that *D* outputs 1 is exactly the probability of the adversary winning in hybrid **Hyb₀**. On the other hand, if *D* receives an obfuscation of C_2 , then the probability that *D* outputs 1 is the probability of the adversary winning in hybrid **Hyb₁**.

The lemma follows. ■

Lemma 3. If our confined PRF is secure, then the advantage of a poly-time attacker in **Hyb₁** is negligibly close to the advantage in **Hyb₂**.

Proof. We prove this lemma by giving a reduction to the pseudorandomness property at punctured points for punctured PRFs. To do so, we must build the algorithms A_1 and A_2 .

$A_1(1^\lambda)$ simply invokes the adversary to obtain the challenge message m^* and state τ' , and outputs the singleton set $S = \{m^*\}$ and $\tau = (1^\lambda, \tau')$.

A_2 obtains as input τ , the punctured key K_S , the singleton set $S = \{m^*\}$, and either a value $t^* = F(K, m^*)$ or a uniformly random value t^* . Then, A_2 invokes $\text{TDFSetup}(1^\lambda)$ to obtain PK and SK . Now

given t^* , it can compute $z^* = g_{PK}(t^*)$. Note that this yields either the z^* value computed in hybrid Hyb_1 or in hybrid Hyb_2 . Since it knows K_S , now A_2 can obfuscate the program Full Domain Hash*, and then execute the adversary and answer its signature queries using the punctured key K_S . Finally, A_2 outputs 1 if the adversary succeeds.

By construction, the pseudorandomness property for punctured PRFs implies the lemma. ■

Lemma 4. If our injective trapdoor function is hard to invert, then the advantage of a poly-time attacker in Hyb_2 is negligible.

Proof. We prove this lemma by giving a reduction to the one-wayness of the injective trapdoor function. To do so, we build an inverting algorithm Inv .

Inv takes as input a public index PK for an injective trapdoor function, and a target $z^* = g_{PK}(t^*)$ for some (as yet unknown) random value t^* . The algorithm Inv then invokes the adversary to obtain m^* , and chooses a PRF key K and builds the punctured key $K(S)$ where $S = \{m^*\}$. It uses this key, together with PK and z^* , to obfuscate the program Full Domain Hash*. It can then execute the adversary, and use its knowledge of $K(S)$ to answer all adversary signing queries. The adversary then terminates with an attempted forgery σ^* on message m^* . By the definition of the program Full Domain Hash*, this forgery can only be valid if $g_{PK}(\sigma^*) = z^*$, and because g_{PK} is injective, this can only happen if $\sigma^* = t^*$. Thus if the adversary is successful, Inv can output σ^* as a valid pre-image of z^* .

We observe that by construction of Inv , the probability of success of Inv is exactly the probability that the attacker succeeds in hybrid Hyb_2 . The lemma follows. ■

These three lemmas together yield our main theorem that the above full domain hash signature scheme is selectively secure.

4 Adaptively Secure RSA Full Domain Hash Signatures

We first describe at a high level what advantage indistinguishability obfuscation gives us in this situation: In several previous constructions of adaptively secure schemes in the plain model starting with the adaptively secure IBE scheme of [BB04a], a special hash function was chosen that allowed for a “partitioning” proof of security. In essence, for this to work, the hash function should have two “modes”:

- In the “normal” mode, the hash function’s parameters are typically just chosen at random, and it behaves like an ordinary hash function.
- In the “partitioning” mode, the hash function parameters are chosen according to a special distribution. This special distribution allows for the efficient computation of the inverse of the hash value for a large fraction of points, but it has the property that computing the inverse of the hash value at any other point is computationally hard.

It is crucial that the input/output functionality of the hash function should be identical in the two modes, and we will also use this property. However, in previous proofs (like [BB04a]), it was also critical that the hash function parameters in “partitioning” mode be information theoretically indistinguishable from the parameters in “normal” mode, and thus the partition should be hidden from the adversary even when given the hash function parameters. This restriction significantly limited the applicability of this technique, as it could only be applied with algebraic structures that allowed for such “pseudorandom” hash parameters. Thanks to indistinguishability obfuscation, however, we can avoid this restriction by obfuscating the hash function description. Thus, even if the natural hash function parameters in “partitioning” mode clearly reveal the partition and thus are distinguishable from normal parameters, because the resulting hash function is *functionally* identical to a hash function in “normal” mode, the obfuscated hash function *must* hide the partition, and this allows the proof of adaptive security to go through.

In describing our signature scheme, For simplicity of exposition, we assume that there is a polynomial $\ell(\lambda)$ which denotes the length of messages to be signed; we denote this message space by $\mathcal{M} = \{0, 1\}^{\ell(\lambda)}$.

More generally, a collision-resistant hash function may be used to hash messages to this size. Below, for any polynomial in λ , after the first mention of this polynomial, we will often suppress the dependence on λ for ease of notation. Thus, below often we will simply refer to the size of messages to be signed by ℓ .

Before describing our construction, we first recall a (simplified) description of the notion of *admissible* hash functions due to [BB04a]. Our definition is a slight variation of the simplified definition due to [FHPS13].

Definition 4. Let ℓ, n and θ be efficiently computable univariate polynomials. We say that an efficiently computable function $h : \{0, 1\}^{\ell(\lambda)} \rightarrow \{0, 1\}^{n(\lambda)}$, and an efficient randomized algorithm AdmSample , is θ -*admissible* if the following condition holds:

For any $u \in (\{0, 1\} \cup \{\perp\})^n$, define $P_u : \{0, 1\}^\ell \rightarrow \{0, 1\}$ as follows: $P_u(x) = 0$ iff $\forall i : h(x)_i \neq u_i$, and otherwise (if $\exists i : h(x)_i = u_i$) we have $P_u(x) = 1$.

Then we require that for any efficiently computable polynomial $Q(\lambda)$, for all $x_1, \dots, x_Q, z \in \{0, 1\}^\ell$, where $z \notin \{x_i\}$, we have that

$$\Pr [P_u(x_1) = P_u(x_2) = \dots = P_u(x_Q) = 1 \wedge P_u(z) = 0] \geq 1/\theta(Q)$$

where the probability is taken only over $u \leftarrow \text{AdmSample}(1^\lambda, Q)$.

Theorem 3 (Admissible Function Families [BB04a], see also [FHPS13] for a simple proof). For any efficiently computable polynomials ℓ, n , there exists an efficiently computable polynomial θ such that there exist θ -admissible function families mapping ℓ bits to n bits.

We now show that we can leverage the structure of the RSA trapdoor permutation to prove adaptive security. The use of RSA as a candidate for a trapdoor permutation was first discussed in the original Bellare-Rogaway [BR93] paper, however, it was in [BR96] that Bellare and Rogaway gave an explicit full domain hash RSA construction. This construction formed the basis for part of the standard PKCS#1 [KS98].

- **Setup**(1^λ) : The setup algorithm first runs an RSA type setup. It chooses random primes p, q of λ bits each. We define $N = p \cdot q$ and $\phi(N) = (p-1)(q-1)$. We let e be a random chosen integer between 1 and $\phi(N)$ such that $\gcd(\phi(N), e) = 1$.

Next, it chooses integers $(a_{1,0}, a_{1,1}), \dots, (a_{n,0}, a_{n,1})$ each uniformly at random from the range $[1, \phi(N) - 1]$. In addition, it chooses a group element $v \in \mathbb{Z}_N^*$. It then creates an obfuscation of the of the program RSA Hash of Figure 3. The size of the program is padded to be the maximum of itself and the program RSA Hash* of Figure 4. We refer to the obfuscated program as the function $H(\cdot)$. This function $H(\cdot)$ will replace the random oracle in the RSA full domain hash scheme, but no other part of the scheme is modified.

The verification key VK consists of the integers N, e as well as the hash function $H : \{0, 1\}^{\ell(\lambda)} \rightarrow \mathbb{Z}_N^*$. The secret key is the integer d where $e \cdot d \equiv 1 \pmod{\phi(N)}$.

- **Sign**($\text{SK}, m \in \mathcal{M}$) : The signature algorithm outputs $\sigma = H(M)^d \pmod{N}$.
- **Verify**(VK, m, σ) The verification algorithm tests if $\sigma^e \equiv H(m) \pmod{N}$ and outputs accept if and only if this holds.

Remark 1. For simplicity of exposition we describe computing the programs output by first computing a integer $\pi(m')$ as a product of n integers and then raising v to this mod N . In practice, it might be more efficient to incrementally raise an accumulated value to each a_{i,m'_i} .

Theorem 4. If our obfuscation scheme is indistinguishably secure and the RSA assumption holds, the above signature scheme is existentially unforgeable against chosen message attacks.

We describe a proof as a sequence of hybrid experiments where the first hybrid corresponds to the original signature security game. In the first hybrid step we do a “partitioning” of the message space. Consider a poly-time attacker that makes $Q = Q(\lambda)$ signature queries $m_1, \dots, m_Q$ and attempts to forge on message

RSA Hash

Constants: RSA modulus N , integers $(a_{1,0}, a_{1,1}), \dots, (a_{n,0}, a_{n,1})$ each in $[1, \phi(N) - 1]$, and $v \in \mathbb{Z}_N^*$.

Input: Message m .

1. Compute $m' = h(m)$.
2. Compute the integer $\pi(m') = \prod_{i \in [n]} a_{i, m'_i}$.
3. Output $v^{\pi(m')} \pmod{N}$.

Figure 3: RSA Hash

RSA Hash*

Constants: RSA modulus N , integers $(c_{1,0}, c_{1,1}), \dots, (c_{n,0}, c_{n,1})$ each chosen as in Hyb_2 , and $v \in \mathbb{Z}_N^*$.

Input: Message m .

1. Compute $m' = h(m)$.
2. Compute the integer $\pi(m') = \prod_{i \in [n]} c_{i, m'_i}$.
3. Output $v^{\pi(m')} \pmod{N}$.

Figure 4: RSA Hash*

$m^* \neq m_i$ for all i . Roughly, at the beginning of Hyb_1 the challenger will now (behind the scenes) partition the message space such that a large fraction of messages will fall into a “query” space and a much smaller, but still non-negligible fraction of messages will fall into the “challenge” space. Furthermore, in this new game the attacker is only considered to have won if he both forged a signature *and* all his signature queries $m_1, \dots, m_n$ fall into the query space and m^* falls into the challenge space. We can show that if an attacker succeeds in the original security game (that does not have these additional restrictions on winning) with non-negligible advantage, then it will succeed in Hyb_1 with non-negligible advantage. Our system uses the Boneh-Boyen [BB04a] admissible hash function defined above, where if an attacker has advantage ϵ in Hyb_0 , he will have advantage $\epsilon/\theta(Q)$ in Hyb_1 .

After the first proof step we prove that a poly-time attacker’s advantage must be negligibly close between each successive hybrid experiment. We finally show that any poly-time attacker in the final experiment that succeeds with non-negligible probability can be used to break the RSA assumption.

- **Hyb₀** : In the first hybrid the following game is played.
 1. The challenger sets $N = p \cdot q$ and $\phi(N) = (p - 1)(q - 1)$. It chooses e as a random chosen integer between 1 and $\phi(N)$ such that $\gcd(\phi(N), e) = 1$.
 2. The challenger chooses integers $(a_{1,0}, a_{1,1}), \dots, (a_{n,0}, a_{n,1})$ each uniformly at random from the range $[1, \phi(N) - 1]$.
 3. The variable v is chosen uniformly at random in $\mathbb{Z}_N^*$.
 4. The hash function $H(\cdot)$ is created as an obfuscation of the program RSA Hash.
 5. The attacker queries the signing oracle at most Q times on messages $m_1, \dots, m_Q$. In its i th query, it receives back $H(m_i)^d \pmod{N}$.
 6. The attacker finally chooses a message m^* , sends a forgery σ^* , and wins if $\text{Verify}(\text{VK}, m^*, \sigma^*) = 1$.
- **Hyb₁** : Is the same as **Hyb₀** except the challenger begins by sampling a string $u \in (\{0, 1, \perp\})^n$ by calling $\text{AdmSample}(1^\lambda, Q) \rightarrow u$, where Q is an upper bound on the number of queries made by the adversary (this could, for example, be the running time of the adversary). At the end of the experiment, the attacker is only considered to be successful if both $\text{Verify}(\text{VK}, m^*, \sigma^*) = 1$ and $P_u(m^*) = 0$ and for all messages m queried, $P_u(m) = 1$. If the attacker is successful but this condition is not satisfied, we say that the hybrid “aborts.”

- **Hyb₂** : Is the same as **Hyb₁** except for the following modification. After sampling u , the challenger chooses integers $(c_{1,0}, c_{1,1}), \dots, (c_{n,0}, c_{n,1})$ in the following way. For $i \in [n]$ and $b \in \{0, 1\}$, $y_{i,b}$ is chosen uniformly at random from all integers in $[1, N]$. The challenger then sets

$$c_{i,b} = \begin{cases} e \cdot y_{i,b} & \text{if } b = u_i \\ e \cdot y_{i,b} + 1 & \text{if } b \neq u_i \end{cases}$$

Observe that $\gcd(e, e \cdot y_{i,b} + 1) = 1$, for all i, b . We note that if $b \neq u_i$, then either $u_i = 1 - b$ or $u_i = \perp$. Then for $i \in [n]$ and $b \in \{0, 1\}$, it sets $a_{i,b} = (c_{i,b} \bmod \phi(N))$.

- **Hyb₃** : Is the same as **Hyb₂** except the challenger creates the hash function $H(\cdot)$ as an obfuscation of the program RSA Hash^* using the values $(c_{1,0}, c_{1,1}), \dots, (c_{n,0}, c_{n,1})$. The “ a ” values are no longer computed or used.

Lemma 5. Consider an attacker that makes at most a polynomial of queries $Q = Q(\lambda)$ in **Hyb₀**. If the advantage of an attacker in **Hyb₀** is $\epsilon(\lambda)$, then the advantage of the attacker in **Hyb₁** will be at least $\epsilon(\lambda)/\theta(Q)$. In particular, any poly-time attacker with non negligible advantage in **Hyb₀** will also have non-negligible advantage in **Hyb₁**.

Proof. The lemma follows immediately from the function h satisfying the definition of a θ -admissibility, since the only independent choice of $u \leftarrow \text{AdmSample}(1^\lambda, Q)$ determines whether or not the hybrid aborts. ■

Lemma 6. The advantage of any attacker in **Hyb₁** is negligibly close to the advantage in **Hyb₂**.

Proof. Fix i, b , and consider the $a_{i,b}$ value that results from the choice of $c_{i,b}$ in hybrid **Hyb₂**. First, observe that the random choice of $y_{i,b}$ from the range $[1, N]$ is negligibly statistically close to a random choice of $y_{i,b}$ in the range $[1, \phi(N) - 1]$, since $N - \phi(N) = p + q - 1$. Thus, for the remainder of the argument, we can assume that each $y_{i,b}$ is chosen uniformly from the range $[1, \phi(N) - 1]$. Next, since $\gcd(e, \phi(N)) = 1$, we have that both $(e \cdot y_{i,b} \bmod \phi(N))$ and $(e \cdot y_{i,b} + 1 \bmod \phi(N))$ are distributed uniformly in the range $[1, \phi(N) - 1]$, and the lemma follows. ■

Lemma 7. If our obfuscation scheme is indistinguishability secure, then the advantage of any poly-time algorithm in **Hyb₂** is negligibly close to the advantage in **Hyb₃**.

Proof. We prove this lemma by giving a reduction to the indistinguishability security of the obfuscator. To do so, we must build the two algorithms *Samp* and *D*.

Samp(1^λ) behaves as follows: It invokes the adversary to obtain the adversary’s state τ' . It runs the RSA type setup as in the real scheme to generate primes p, q and sets $N = p \cdot q$ and $\phi(N) = (p - 1)(q - 1)$. It chooses e as a random integer between 1 and $\phi(N)$ such that $\gcd(\phi(N), e) = 1$. It sets integer d where $e \cdot d \equiv 1 \pmod{\phi(N)}$. It chooses $v \in \mathbb{Z}_N^*$ as a random element. It chooses $(c_{1,0}, c_{1,1}), \dots, (c_{n,0}, c_{n,1})$ and $(a_{1,0}, a_{1,1}), \dots, (a_{n,1}, a_{n,0})$ derived from them, as in **Hyb₂**. It sets $\tau = (N, p, q, e, d, (c_{1,0}, c_{1,1}), \dots, (c_{n,0}, c_{n,1}), (a_{1,0}, a_{1,1}), \dots, (a_{n,1}, a_{n,0}), v, \tau')$ and builds C_1 as the program for RSA Hash , and C_2 as the program for RSA Hash^* .

Before describing *D*, we observe that by construction, the circuits C_1 and C_2 always behave identically on every input. To show program equivalence, note that since $\mathbb{Z}_N^*$ is of order $\phi(N)$ for all m' , we have that

$$\begin{aligned} v^{\prod_i c_{i,m'_i}} \pmod{N} &= v^{(\prod_i c_{i,m'_i} \pmod{\phi(N)})} \pmod{N} = \\ v^{\prod_i (c_{i,m'_i} \pmod{\phi(N)})} \pmod{N} &= v^{\prod_i a_{i,m'_i}} \pmod{N}. \end{aligned}$$

With suitable padding, both C_1 and C_2 have the same size. Thus, *Samp* satisfies the conditions needed for invoking the indistinguishability property of the obfuscator.

Now, we can describe the algorithm D , which takes as input τ as given above, and either the obfuscation of C_1 , which is the program RSA Hash, or C_2 , which is the program RSA Hash*. D then invokes the adversary, which makes requests for signatures on messages. D constructs the signatures $H(m)^d$, through its knowledge of d within τ . Finally, the attacker sends a forgery-message pair (σ^*, m^*) and wins if $\text{Verify}(\text{VK}, m^*, \sigma^*) = 1$. If the attacker wins, D outputs 1.

By construction, if D receives an obfuscation of C_1 , then the probability that D outputs 1 is exactly the probability of the adversary winning in hybrid Hyb_0 . On the other hand, if D receives an obfuscation of C_2 , then the probability that D outputs 1 is the probability of the adversary winning in hybrid Hyb_1 .

The lemma follows. ■

Lemma 8. If the RSA assumption holds, then the advantage of an poly-time algorithm in Hyb_3 is negligible.

Proof. We prove this lemma by giving a reduction to the RSA problem. To do so, we build algorithm $\mathcal{B}$.

$\mathcal{B}$ takes as input an RSA challenge (N, e, v) where N is the product of two (unknown) primes p, q , e is randomly chosen from $[1, \phi(N)]$ such that $\gcd(\phi(N), e) = 1$ and $v \in \mathbb{Z}_N^*$. Next, $\mathcal{B}$ calls $\text{AdmSample}(1^\lambda, Q) \rightarrow u$, where Q is an upper bound on the number of queries made by the adversary. Then $\mathcal{B}$ chooses integers $(c_{1,0}, c_{1,1}), \dots, (c_{n,0}, c_{n,1})$ in the following way. For $i \in [n]$ and $b \in \{0, 1\}$ if $b = u_i$ then first $y_{i,b}$ is chosen uniformly at random from all integers in $[1, N]$, and $c_{i,b} = e \cdot y_{i,b}$. Otherwise $y_{i,b}$ is chosen uniformly at random from all integers in $[1, N]$, and $c_{i,b} = e \cdot y_{i,b} + 1$. Finally, $\mathcal{B}$ creates the hash function $H(\cdot)$ as an obfuscation of the program RSA Hash* using the N and values $(c_{1,0}, c_{1,1}), \dots, (c_{n,0}, c_{n,1})$ above, and the value v in the RSA Hash* program will be the value v in the RSA challenge. All these steps together simulate the setup phase of Hyb_3 . Now, it runs the attacker using the initial parameters generated above.

The attacker will then make at most Q signing queries each for a message m . We denote $m' = h(m)$ and the integer $\pi(m') = \prod_{i \in [n]} c_{i,m'_i}$. If $P_u(m) \neq 1$ $\mathcal{B}$ aborts and quits. Otherwise, $P_u(m) = 1$ and there exists an i where $e \mid c_{i,m'_i}$ and therefore $e \mid \pi(m')$. $\mathcal{B}$ can then compute the integer $t = \pi(m')/e$ and compute the (unique) signature on m as v^t .

Finally, the attacker will output an attempted forgery σ^* on some message m^* that is distinct from all the messages in the query phase. $\mathcal{B}$ first checks if the signature verifies and aborts if it does not. Next, it checks if $P_u(m^*) \neq 0$ and aborts if that is the case. Otherwise, $P_u(m^*) = 0$ and for all i we have $\gcd(e, c_{i,m'^*_i}) = 1$ and therefore $\gcd(e, \pi(m'^*)) = 1$. Following Shamir's theorem [Sha83], the attacker applies the Euclidean Algorithm to obtain integers α and β such that $\alpha \cdot e + \beta \cdot \pi(m'^*) = 1$. Therefore, since $(\sigma^*)^e = v^{\pi(m'^*)}$, it sets $z = v^\alpha \cdot (\sigma^*)^\beta$, and we have that $z^e = v^{\alpha e + \beta \pi(m'^*)} = v$. If σ^* was a successful forgery, then this value z is a solution to the RSA challenge.

We observe that by construction of $\mathcal{B}$, the probability of success of $\mathcal{B}$ is exactly the probability that the attacker succeeds in hybrid Hyb_3 . Importantly, whenever $\mathcal{B}$ aborted, the attacker by the rules of Hyb_3 was not considered to be successful since his queries or forgery violated the partition. The lemma follows. ■

Pulling together these four lemmas immediately gives our the main theorem that the above RSA full domain hash signature scheme is (adaptively) secure.

5 Selectively Secure BLS Signatures

We now give a concrete construction for the hash function modeled as a random oracle in the Boneh-Lynn-Shacham (BLS) signature scheme. BLS signatures fall into a broad interpretation (see e.g., [Boy08]) of the full domain hash paradigm of Bellare and Rogaway.

Below we give the BLS signature scheme with a concrete hash function built from an indistinguishability obfuscator. We prove the signature scheme selectively secure based on the computational Diffie-Hellman problem in bilinear groups and a indistinguishability obfuscator.

On a technical level this selective proof of security follows a very similar structure to that of our selectively secure scheme from trapdoor functions from Section 3. The main difference is that here we deal with

the mechanics of an algebraic bilinear group instead of a trapdoor function. We present the scheme for simplicity in terms of a symmetric bilinear group, however, we remark that moving to asymmetric groups is straightforward.

As in Section 3, we assume that there is a polynomial $\ell(\lambda)$ which denotes the length of messages to be signed; we denote this message space by $\mathcal{M} = \{0, 1\}^{\ell(\lambda)}$. More generally, a collision-resistant hash function may be used to hash messages to this size.

- **Setup**(1^λ) : The setup algorithm first runs the group generator on input 1^λ to produce a description of groups $\mathbb{G}, \mathbb{G}_T$ of prime order p along with generator $g \in \mathbb{G}$. These groups are related by a bilinear map $e : \mathbb{G} \times \mathbb{G} \rightarrow \mathbb{G}_T$. Next, it chooses a random exponent $a \in \mathbb{Z}_p$. Then, the setup algorithm chooses a puncturable PRF key K for F where $F(K, \cdot) : \{0, 1\}^{\ell(\lambda)} \rightarrow \mathbb{Z}_p$. Finally, it creates an obfuscation of the program BLS Selective Hash of Figure 5. The size of the program is padded to be the maximum of itself and the program BLS Selective Hash* of Figure 6. We refer to the obfuscated program as the function $H : \{0, 1\}^\ell \rightarrow \mathbb{G}$, which acts as the random oracle type hash function in the BLS scheme.

The verification key VK consists of the group descriptions $\mathbb{G}, \mathbb{G}_T$, the order p , the generator g and $A = g^a$ as well as the hash function $H(\cdot)$. The secret key is $a \in \mathbb{Z}_p$ as well as $H(\cdot)$.

- **Sign**(SK, $m \in \mathcal{M}$) : The signature algorithm outputs $\sigma = H(m)^a \in \mathbb{G}$.
- **Verify**(VK, m, σ) The verification algorithm tests if $e(\sigma, g) \stackrel{?}{=} e(A, H(m))$ and outputs accept if and only if this holds.

Remark 2. The confined PRFs from [BW13] use the GGM tree and get PRFs in range $\{0, 1\}^n$ for some n , whereas our PRFs need to hash to $\mathbb{Z}_p$. One can achieve a punctured PRF for the proper range by simply setting $n > 2 \lg(p)$ and taking interpreting the GGM output as an integer that is then mod by p . This is sufficient since sampling an integer in $[0, 2^n - 1]$ and then reducing it mod p is statistically close to choosing an integer in $[0, p - 1]$.

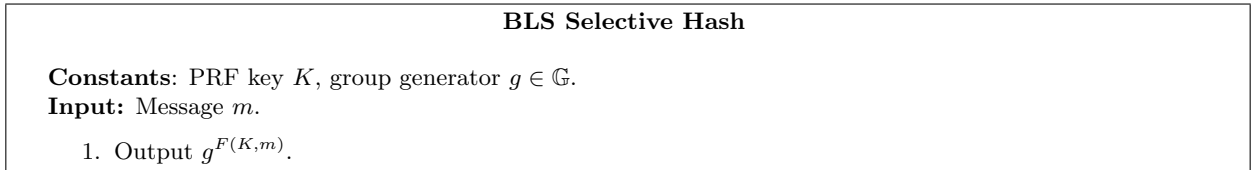


Figure 5: BLS Selective Hash

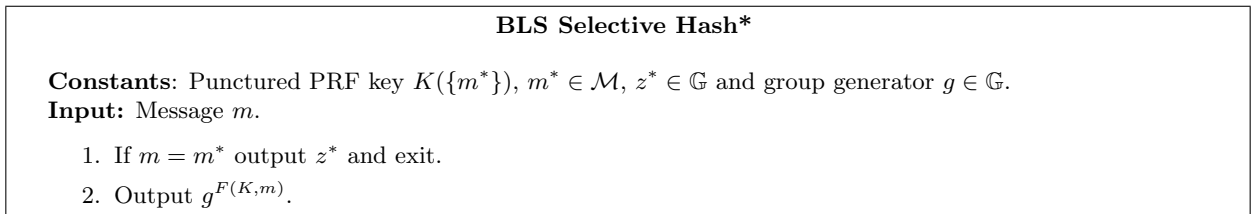


Figure 6: BLS Selective Hash*

Theorem 5. If our obfuscation scheme is indistinguishability secure, F is a secure punctured PRF, and the computational Diffie-Hellman problem holds in bilinear groups, then the above signature scheme is selectively secure.

We describe a proof as a sequence of hybrid experiments where the first hybrid corresponds to the original signature security game. We prove that a poly-time attacker's advantage must be negligibly close between each successive one. Then, we show that any poly-time attacker in the final experiment that succeeds in forging with non-negligible probability can be used to break the computational Diffie-Hellman assumption in bilinear groups.

- **Hyb₀** : In the first hybrid the following game is played:
 1. The attacker selectively gives the challenger the message m^* .
 2. The challenger runs the group generator to produce bilinear groups $\mathbb{G}, \mathbb{G}_T$ of order p with generator $g \in \mathbb{G}$. It then chooses a random exponent $a \in \mathbb{Z}_p$ for the secret key and sets $A = g^a$ as part of the verification key.
 3. K is chosen as a key for the puncturable PRF.
 4. The hash function $H(\cdot)$ is created as an obfuscation of the program BLS Selective Hash.
 5. The attacker queries the sign oracle a polynomial number of times on messages $m \neq m^*$. It receives back $H(m)^a = A^{F(K,m)}$.
 6. The attacker sends a forgery σ^* and wins if $\text{Verify}(m^*, \sigma^*) = 1$.
- **Hyb₁** : Is the same as **Hyb₀** except we let $z^* = g^{F(K,m^*)}$ and let VK be the obfuscation of the program BLS Selective Hash* of Figure 6.
- **Hyb₂** : Is the same as **Hyb₁** except $z^* = g^t$ for t chosen uniformly at random in $\mathbb{Z}_p$.

Lemma 9. If our obfuscation scheme is indistinguishability secure, then the advantage of an poly-time algorithm in **Hyb₀** is negligibly close to the advantage in **Hyb₁**.

Proof. We prove this lemma by giving a reduction to the indistinguishability security of the obfuscator. To do so, we must build the two algorithms *Samp* and *D*.

Samp(1^λ) behaves as follows: It invokes the adversary to obtain m^* and the adversary's state τ' . It runs the bilinear group setup on 1^λ to obtain the group descriptions $\mathbb{G}, \mathbb{G}_T$, order p and generator g . It then chooses a random $a \in \mathbb{Z}_p$ and sets $A = g^a$. It then chooses K as a key for the puncturable PRF. It sets $z^* = g^{F(K,m^*)}$. It sets $\tau = (m^*, z^*, g, p, A, K, \tau')$ and builds C_1 as the program for BLS Selective Hash, and C_2 as the program for BLS Selective Hash*.

Before describing *D*, we observe that by construction and the functionality preservation property of puncturable PRFs, the circuits C_1 and C_2 always behave identically on every input. Because of padding, both C_1 and C_2 have the same size. Thus, *Samp* satisfies the conditions needed for invoking the indistinguishability property of the obfuscator.

Now, we can describe the algorithm *D*, which takes as input τ as given above, and either the obfuscation of C_1 , which is the program BLS Selective Hash, or C_2 , which is the program BLS Selective Hash*. *D* creates the verification key for the signature scheme with the obfuscated program as the hash function description. It then invokes the adversary on this verification key, and the adversary makes requests for signatures on messages $m \neq m^*$. For each such message, *D* constructs the signatures $H(m)^a = A^{F(K,m)}$, through its knowledge of K and A within τ . Finally, the attacker sends a forgery σ^* and wins if $\text{Verify}(m^*, \sigma^*) = 1$. If the attacker wins, *D* outputs 1.

By construction, if *D* receives an obfuscation of C_1 , then the probability that *D* outputs 1 is exactly the probability of the adversary winning in hybrid **Hyb₀**. On the other hand, if *D* receives an obfuscation of C_2 , then the probability that *D* outputs 1 is the probability of the adversary winning in hybrid **Hyb₁**.

The lemma follows. ■

Lemma 10. If our confined PRF is secure, then the advantage of an poly-time algorithm in **Hyb₁** is negligibly close to the advantage in **Hyb₂**.

Proof. We prove this lemma by giving a reduction to the pseudorandomness property at punctured points for punctured PRFs. To do so, we must build the algorithms A_1 and A_2 .

$A_1(1^\lambda)$ simply invokes the adversary to obtain the challenge message m^* and state τ' , and outputs the singleton set $S = \{m^*\}$ and $\tau = (1^\lambda, \tau')$.

A_2 obtains as input τ , the punctured key K_S , the singleton set $S = \{m^*\}$, and either a value $t^* = F(K, m^*)$ or a uniformly random value $t^* \in \mathbb{Z}_p$. Then, A_2 runs the group generator on 1^λ to obtain

$(\mathbb{G}, \mathbb{G}_T, p, g)$, chooses a random $a \in \mathbb{Z}_p$, and sets $A = g^a$ to establish portions of VK and SK. Now given t^* , it can compute $z^* = g^{t^*}$. Note that this yields either the z^* value computed in hybrid Hyb_1 or in hybrid Hyb_2 . Since it knows K_S , now A_2 can obfuscate the program BLS Selective Hash*, and then execute the adversary and answer its signature queries using the punctured key K_S . Finally, A_2 outputs 1 if the adversary succeeds.

By construction, the pseudorandomness property for punctured PRFs implies the lemma. $\blacksquare$

Lemma 11. If the computational Diffie-Hellman assumption holds in bilinear groups, then the advantage of an poly-time algorithm in Hyb_2 is negligible.

Proof. We prove this lemma by giving a reduction to the hardness of the Computational Diffie-Hellman problem in the bilinear group $\mathbb{G}$. To do so, we build a CDH attacker $\mathcal{B}$.

$\mathcal{B}$ takes as input the tuple (g, g^a, g^b) for a group $\mathbb{G}$ of prime order p with a bilinear map $e : \mathbb{G} \times \mathbb{G} \rightarrow \mathbb{G}_T$. It sets $A = g^a$ as part of the verification key VK. The algorithm $\mathcal{B}$ then invokes the adversary to obtain m^* , and chooses a PRF key K and builds the punctured key $K(S)$ where $S = \{m^*\}$. It uses this key, together with VK and $z^* = g^b$, to obfuscate the program BLS Selective Hash*. It can then execute the adversary, and use its knowledge of $K(S)$ to answer all adversary signing queries. The adversary then terminates with an attempted forgery σ^* on message m^* . By the definition of the program BLS Selective Hash*, this forgery can only be valid if $\sigma^* = (z^*)^a = g^{ab}$. Thus if the adversary is successful, $\mathcal{B}$ can output σ^* as the solution to the CDH problem.

We observe that by construction of $\mathcal{B}$, the probability of success of $\mathcal{B}$ is exactly the probability that the attacker succeeds in hybrid Hyb_2 . The lemma follows. $\blacksquare$

Pulling together these three lemmas immediately gives our the main theorem that the above full domain hash signature scheme is selectively secure.

6 Adaptively Secure BLS Signatures

We now give a hash function for BLS signatures that can be used to prove adaptive (or standard) security. Our proof structure will follow in a similar path to that of our adaptively secure RSA full domain hash signatures in Section 4. In particular, we will again apply an admissible hash function to partition the message space in our proof. At the same time, there are important distinctions and corresponding challenges that arise in this setting.

Again, for simplicity of exposition, we assume that there is a polynomial $\ell(\lambda)$ which denotes the length of messages to be signed; we denote this message space by $\mathcal{M} = \{0, 1\}^{\ell(\lambda)}$ and we will often simply refer to the size of messages to be signed by ℓ . As in Section 4, we use a function $h : \{0, 1\}^{\ell(\lambda)} \rightarrow \{0, 1\}^{n(\lambda)}$, and an efficient randomized algorithm AdmSample that is θ -admissible.

Our construction is identical to that given in Section 6 with the exception of how the setup creates the hash function. The setup first chooses uniformly at random $(c_{1,0}, c_{1,1}), \dots, (c_{n,0}, c_{n,1})$ each in $\mathbb{Z}_p$. Then it obfuscates the program BLS Adaptive Hash of Figure 7 where the size of the program is padded to be the maximum of itself and the program BLS Adaptive Hash* of Figure 8. The obfuscated program is used as the function $H : \{0, 1\}^{\ell(\lambda)} \rightarrow \mathbb{G}$, which acts as the random oracle type hash function in the BLS scheme.

Our proof of security relies on indistinguishability obfuscation and our Diffie-Hellman Inversion equivalent assumption. Namely, that given $g, g^a, g^{a^2}, \dots, g^{a^n} \in \mathbb{G}$, it is hard to compute $g^{a^{n+1}}$.

Theorem 6. If our obfuscation scheme is indistinguishability secure and the Diffie-Hellman Inversion assumption holds in bilinear group $\mathbb{G}$, the above signature scheme is existentially unforgeable against chosen message attacks.

We describe a proof as a sequence of hybrid experiments where the first hybrid corresponds to the original signature security game. As in Section 4, in the first hybrid step we do a “partitioning” of the message space. After the first proof step, we prove that any poly-time attacker’s advantage must be negligibly close between each successive hybrid experiment. We finally show that any poly-time attacker in the final experiment

BLS Adaptive Hash

Constants: Bilinear group $\mathbb{G}$ with generator g and exponents $(c_{1,0}, c_{1,1}), \dots, (c_{n,0}, c_{n,1})$ each in $\mathbb{Z}_p$.

Input: Message $m \in \{0, 1\}^\ell$.

1. Compute $m' = h(m)$.
2. Output $g^{\prod_{i=1, \dots, n} c_{i, m'_i}}$.

Figure 7: BLS Adaptive Hash

BLS Adaptive Hash*

Constants: Bilinear group $\mathbb{G}$, elements $g, g^a, g^{a^2}, \dots, g^{a^n} \in \mathbb{G}$, for $i \in [n], b \in \{0, 1\}$ exponents $y_{i,b} \in \mathbb{Z}_p$ and $u \in \{0, 1\}^n$.

Input: Message $m \in \{0, 1\}^\ell$.

1. Compute $m' = h(m)$.
2. Let $\mu(m)$ be the set i such that $m'_i \neq u_i$. The algorithm computes the set size $|\mu(m)|$.
3. Output $(g^{a^{|\mu(m)|}})^{\prod_{i=1, \dots, n} y_{i, m'_i}}$.

Figure 8: BLS Adaptive Hash*

that succeeds with non-negligible probability can be used to break our assumption that is equivalent to the Diffie-Hellman Inversion assumption. See Section 2.5 for more on these assumptions.

- **Hyb₀** : In the first hybrid, the following game is played:
 1. The challenger runs the group generator to produce bilinear groups $\mathbb{G}, \mathbb{G}_T$ of order p with generator $g \in \mathbb{G}$. It then chooses a random exponent $a \in \mathbb{Z}_p$ for the secret key and sets $A = g^a$ as part of the verification key.
 2. It chooses uniformly at random $(c_{1,0}, c_{1,1}), \dots, (c_{n,0}, c_{n,1})$ each in $\mathbb{Z}_p$.
 3. The hash function $H(\cdot)$ is created as an obfuscation of the program BLS Adaptive Hash.
 4. The attacker queries the signing oracle at most Q times on messages $m_1, \dots, m_Q$. In its i th query, it receives back $H(m_i)^a$.
 5. The attacker finally chooses a message m^* , sends a forgery σ^* , and wins if $\text{Verify}(\text{VK}, m^*, \sigma^*) = 1$.
- **Hyb₁** : Is the same as **Hyb₀** except the challenger begins by sampling a string $u \in (\{0, 1, \perp\})^n$ by calling $\text{AdmSample}(1^\lambda, Q) \rightarrow u$, where Q is an upper bound on the number of queries made by the adversary (this could, for example, be the running time of the adversary). At the end of the experiment, the attacker is only considered to be successful if both $\text{Verify}(\text{VK}, m^*, \sigma^*) = 1$ and $P_u(m^*) = 0$ and for all messages m_i queried $P_u(m_i) = 1$. If the attacker is successful but this condition is not satisfied, we say that the hybrid “aborts.”
- **Hyb₂** : Is the same as **Hyb₁** except the for the following modification. The challenger first chooses exponents $(c_{1,0}, c_{1,1}), \dots, (c_{n,0}, c_{n,1})$ in the following way. For $i \in [n], b \in \{0, 1\}$ it chooses random $y_{i,b} \in \mathbb{Z}_p$ and then sets

$$c_{i,b} = \begin{cases} y_{i,b} & \text{if } b = u_i \\ a \cdot y_{i,b} & \text{if } b \neq u_i \end{cases}$$

- **Hyb₃** : Is the same as **Hyb₂** except the challenger creates the hash function $H(\cdot)$ as an obfuscation of the program BLS Adaptive Hash*.

Lemma 12. Consider an attacker that makes at most a polynomial of queries $Q = Q(\lambda)$ in Hyb_0 . If the advantage of an attacker in Hyb_0 is $\epsilon(\lambda)$, then the advantage of the attacker in Hyb_1 will be at least $\epsilon(\lambda)/\theta(Q)$. In particular, any poly-time attacker with non negligible advantage in Hyb_0 will also have non-negligible advantage in Hyb_1 .

Proof. The lemma follows immediately from the function h satisfying the definition of a θ -admissibility, since the only independent choice of $u \leftarrow \text{AdmSample}(1^\lambda, Q)$ determines whether or not the hybrid aborts. (This argument is identical to the corresponding one in the proof of Lemma 5 of Section 4.) ■

Lemma 13. The advantage of any poly-time algorithm in Hyb_1 is the same as its advantage in Hyb_2 .

Proof. The two hybrid experiment are equivalent as all $c_{i,b} \in \mathbb{Z}_p$ values are still chosen uniformly at random in both hybrids. (The step from Hyb_1 to Hyb_2 is a notational reorganization to set up the next proof step.) ■

Lemma 14. If our obfuscation scheme is indistinguishability secure, then the advantage of any poly-time algorithm in Hyb_2 is negligibly close to the advantage in Hyb_3 .

Proof. We prove this lemma by giving a reduction to the indistinguishability security of the obfuscator. To do so, we must build the two algorithms Samp and D .

$\text{Samp}(1^\lambda)$ behaves as follows: It invokes the adversary to obtain the adversary's state τ' . It runs the bilinear group setup to obtain $\mathbb{G}, \mathbb{G}_T, p, g$ and then chooses a random $a \in \mathbb{Z}_p$. For $i \in [n], b \in \{0, 1\}$ it chooses random $y_{i,b} \in \mathbb{Z}_p$ and then sets

$$c_{i,b} = \begin{cases} y_{i,b} & \text{if } b = u_i \\ a \cdot y_{i,b} & \text{if } b \neq u_i \end{cases}$$

It samples a string $u \in (\{0, 1, \perp\})^n$ by calling $\text{AdmSample}(1^\lambda, Q) \rightarrow u$, where Q is an upper bound on the number of queries made by the adversary. It sets $\tau = (\mathbb{G}, g, a, (c_{1,0}, c_{1,1}), \dots, (c_{n,0}, c_{n,1}), (y_{1,0}, y_{1,1}), \dots, (y_{n,0}, y_{n,1}), u, \tau')$ and builds C_1 as the program for BLS Adaptive Hash, and C_2 as the program for BLS Adaptive Hash*.

Before describing D , we observe that by construction, the circuits C_1 and C_2 always behave identically on every input. To show program equivalence, note that for all m' , we have that

$$g^{\prod_i c_{i,m'_i}} = g^{a^{|\mu(m')|} \cdot \prod_i y_{i,m'_i}} = (g^{a^{|\mu(m')|}})^{\prod_i y_{i,m'_i}}.$$

With suitable padding, both C_1 and C_2 have the same size. Thus, Samp satisfies the conditions needed for invoking the indistinguishability property of the obfuscator.

Now, we can describe the algorithm D , which takes as input τ as given above, and either the obfuscation of C_1 , which is the program BLS Adaptive Hash, or C_2 , which is the program BLS Adaptive Hash*. D then invokes the adversary, which makes requests for signatures on messages. D constructs the signatures $H(m)^a$, through its knowledge of a within τ . Finally, the attacker sends a forgery-message pair (σ^*, m^*) and wins if $\text{Verify}(\text{VK}, m^*, \sigma^*) = 1$. If the attacker wins, D outputs 1.

By construction, if D receives an obfuscation of C_1 , then the probability that D outputs 1 is exactly the probability of the adversary winning in hybrid Hyb_2 . On the other hand, if D receives an obfuscation of C_2 , then the probability that D outputs 1 is the probability of the adversary winning in hybrid Hyb_3 .

The lemma follows. ■

Lemma 15. If the Diffie-Hellman Inversion Assumption holds in bilinear group $\mathbb{G}$, then the advantage of any poly-time algorithm in Hyb_3 is negligible.

Proof. We prove this lemma by giving a reduction to the Diffie-Hellman Inversion problem. To do so, we build algorithm $\mathcal{B}$.

$\mathcal{B}$ takes as input a n -DHI challenge $(g, g^a, g^{a^2}, \dots, g^{a^n})$ from a bilinear group $\mathbb{G}$ of prime order p , where n is the same as the output length of the admissible hash function $h(\cdot)$. Next, $\mathcal{B}$ calls $\text{AdmSample}(1^\lambda, Q) \rightarrow u$, where Q is an upper bound on the number of queries made by the adversary. For $i \in [n], b \in \{0, 1\}$, it chooses random $y_{i,b} \in \mathbb{Z}_p$.

Finally, $\mathcal{B}$ creates the hash function $H(\cdot)$ as an obfuscation of the program BLS Adaptive Hash* using the DHI challenge values, the values $(y_{1,0}, y_{1,1}), \dots, (y_{n,0}, y_{n,1})$ above and u . All these steps together simulate the setup phase of Hyb_3 . Now, it runs the attacker using the initial parameters above, including $A = g^a$.

The attacker will then make at most Q signing queries each for a message m . We denote $m' = h(m)$. If $P_u(m) \neq 1$, $\mathcal{B}$ aborts and quits. Otherwise, $P_u(m) = 1$ and there exists an i where $m'_i = u_i$, meaning that the hash of m' will contain a power of a that is strictly less than n . Thus, the signature can be formed using only knowledge of the DHI input and the $y_{i,b}$ values.

Finally, the attacker will output an attempted forgery σ^* on some message m^* that is distinct from all the messages in the query phase. $\mathcal{B}$ first checks if the signature verifies and aborts if it does not. Next, it checks if $P_u(m^*) \neq 0$ and aborts if that is the case. Otherwise, $P_u(m^*) = 0$ and for all i we have $h(m^*)_i \neq u_i$. This means that the hash of m^* will be g^{a^n} raised to some known product of $y_{i,b}$ values. The signature therefore contains $g^{a^{n+1}}$ raised to some known product of $y_{i,b}$ values, since signatures contain one more factor of a in the exponent than their corresponding hash values. This value can be recovered by taking the proper root of the signature, i.e., $(\sigma^*)^{1/\prod_i y_{i,h(m^*)_i}} = g^{a^{n+1}}$, and thus if σ^* was a successful forgery, then this root of the signature is a solution to the DHI challenge.

We observe that by construction of $\mathcal{B}$, the probability of success of $\mathcal{B}$ is exactly the probability that the attacker succeeds in hybrid Hyb_3 . Importantly, whenever $\mathcal{B}$ aborted, the attacker by the rules of Hyb_3 was not considered to be successful since his queries or forgery violated the partition. The lemma follows. ■

Pulling together these four lemmas immediately gives our the main theorem that the above BLS signature scheme is (adaptively) secure.

7 Extensions to Boneh-Franklin IBE and Aggregate Signatures

Boneh-Franklin IBE We can adapt our techniques for proving security of BLS signatures to the Boneh-Franklin [BF01] Identity-Based Encryption system. BLS signatures directly correspond to IBE private keys in the BF scheme. The proof for the BF adapts with a few minor changes:

- For proving BF selectively secure we can use the decision Bilinear Diffie-Hellman assumption.
- The second random oracle in the BF scheme can be replaced with an extractor.
- For proving adaptive security we use the following assumption. Namely that given $g, g^s, g^a, g^{a^2}, \dots, g^{a^n}$ it is hard to distinguish $e(g, g)^{a^{n+1}s}$ from a random group element in $\mathbb{G}_T$. We note this assumption is weaker than the decision Bilinear Diffie-Hellman Exponent assumption [BGW05].

BLGS Aggregate Signatures Boneh, Gentry, Lynn and Shacham [BGLS03] showed that the BLS signatures are aggregateable by reduction to the BDH assumption. Later Bellare, Namprempre and Neven [BNN07] showed how an aggregate signature scheme could built directly from and reduced to the security of BLS signatures. Using their results we immediately get an aggregate signature scheme.

Acknowledgments

We thank Mihir Bellare for discussions relating to the origins and terminology of full domain hash signatures and other helpful discussions. We thank Dan Boneh for many helpful discussions and also for pointing out the equivalence of our assumption used in Section 6 to the Diffie-Hellman Inversion Assumption. We thank

Dennis Hofheinz for clarifications on admissible hash functions and pointing us to the simplified version we used. Finally, we are grateful to the anonymous reviewers of Eurocrypt 2014 for their helpful comments.

References

- [BB04a] Dan Boneh and Xavier Boyen. Secure identity based encryption without random oracles. In *CRYPTO*, pages 443–459, 2004.
- [BB04b] Dan Boneh and Xavier Boyen. Short signatures without random oracles. *IACR Cryptology ePrint Archive*, 2004:171, 2004.
- [BBP04] Mihir Bellare, Alexandra Boldyreva, and Adriana Palacio. An uninstantiable random-oracle-model scheme for a hybrid-encryption problem. In *EUROCRYPT*, pages 171–188, 2004.
- [BF01] Dan Boneh and Matthew K. Franklin. Identity-based encryption from the weil pairing. In *Proceedings of the 21st Annual International Cryptology Conference on Advances in Cryptology*, CRYPTO '01, 2001.
- [BGI⁺01] Boaz Barak, Oded Goldreich, Russell Impagliazzo, Steven Rudich, Amit Sahai, Salil P. Vadhan, and Ke Yang. On the (im)possibility of obfuscating programs. In *CRYPTO*, pages 1–18, 2001.
- [BGI⁺12] Boaz Barak, Oded Goldreich, Russell Impagliazzo, Steven Rudich, Amit Sahai, Salil P. Vadhan, and Ke Yang. On the (im)possibility of obfuscating programs. *J. ACM*, 59(2):6, 2012.
- [BGI13] Elette Boyle, Shafi Goldwasser, and Ioana Ivan. Functional signatures and pseudorandom functions. *IACR Cryptology ePrint Archive*, 2013:401, 2013.
- [BGLS03] Dan Boneh, Craig Gentry, Ben Lynn, and Hovav Shacham. Aggregate and verifiably encrypted signatures from bilinear maps. In *EUROCRYPT*, pages 416–432, 2003.
- [BGW05] Dan Boneh, Craig Gentry, and Brent Waters. Collusion resistant broadcast encryption with short ciphertexts and private keys. In *CRYPTO*, pages 258–275, 2005.
- [BHK13] Mihir Bellare, Viet Tung Hoang, and Sriram Keelveedhi. Instantiating random oracles via uces. *IACR Cryptology ePrint Archive*, 2013:424, 2013.
- [BLS01] Dan Boneh, Ben Lynn, and Hovav Shacham. Short signatures from the weil pairing. In *ASIACRYPT*, pages 514–532, 2001.
- [BNN07] Mihir Bellare, Chanathip Namprempre, and Gregory Neven. Unrestricted aggregate signatures. In *ICALP*, pages 411–422, 2007.
- [Boy08] Xavier Boyen. A tapestry of identity-based encryption: practical frameworks compared. *IJACT*, 1(1):3–21, 2008.
- [BR93] Mihir Bellare and Phillip Rogaway. Random oracles are practical: A paradigm for designing efficient protocols. In *ACM Conference on Computer and Communications Security*, pages 62–73, 1993.
- [BR96] Mihir Bellare and Phillip Rogaway. The exact security of digital signatures - how to sign with rsa and rabin. In *EUROCRYPT*, pages 399–416, 1996.
- [BW13] Dan Boneh and Brent Waters. Constrained pseudorandom functions and their applications. *IACR Cryptology ePrint Archive*, 2013:352, 2013.
- [CGH98] Ran Canetti, Oded Goldreich, and Shai Halevi. The random oracle methodology, revisited (preliminary version). In *STOC*, pages 209–218, 1998.

- [CHK07] Ran Canetti, Shai Halevi, and Jonathan Katz. A forward-secure public-key encryption scheme. *J. Cryptology*, 20(3):265–294, 2007.
- [Coc01] Clifford Cocks. An identity based encryption scheme based on quadratic residues. In *IMA Int. Conf.*, pages 360–363, 2001.
- [CS98] Ronald Cramer and Victor Shoup. A practical public key cryptosystem provably secure against adaptive chosen ciphertext attack. In *CRYPTO*, pages 13–25, 1998.
- [DOP05] Yevgeniy Dodis, Roberto Oliveira, and Krzysztof Pietrzak. On the generic insecurity of the full domain hash. In *CRYPTO*, pages 449–466, 2005.
- [FHPS13] Eduarda S. V. Freire, Dennis Hofheinz, Kenneth G. Paterson, and Christoph Striecks. Programmable hash functions in the multilinear setting. *IACR Cryptology ePrint Archive*, 2013:354, 2013.
- [GGH⁺13] Sanjam Garg, Craig Gentry, Shai Halevi, Mariana Raykova, Amit Sahai, and Brent Waters. Candidate indistinguishability obfuscation and functional encryption for all circuits. In *FOCS*, 2013.
- [GGM84] Oded Goldreich, Shafi Goldwasser, and Silvio Micali. How to construct random functions (extended abstract). In *FOCS*, pages 464–479, 1984.
- [GK03] Shafi Goldwasser and Yael Tauman Kalai. On the (in)security of the fiat-shamir paradigm. In *FOCS*, pages 102–113, 2003.
- [HSW13] Susan Hohenberger, Amit Sahai, and Brent Waters. Full domain hash from (leveled) multilinear maps and identity-based aggregate signatures. In *CRYPTO*, 2013.
- [KPTZ13] Aggelos Kiayias, Stavros Papadopoulos, Nikos Triandopoulos, and Thomas Zacharias. Delegatable pseudorandom functions and applications. *IACR Cryptology ePrint Archive*, 2013:379, 2013.
- [KS98] B. Kaliski and J. Staddon. Pkcs #1: Rsa cryptography specifications version 2.0, 1998.
- [RSA78] Ronald L. Rivest, Adi Shamir, and Leonard M. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Commun. ACM*, 21(2):120–126, 1978.
- [Sha83] Adi Shamir. On the generation of cryptographically strong pseudorandom sequences. *ACM Trans. Comput. Syst.*, 1(1):38–44, 1983.
- [SW13] Amit Sahai and Brent Waters. How to use indistinguishability obfuscation: Deniable encryption, and more. Cryptology ePrint Archive, Report 2013/454, 2013. <http://eprint.iacr.org/>.

Parallel Repetition Theorems for Interactive Arguments*

Kai-Min Chung[†] and Rafael Pass[‡]

Department of Computer Science, Cornell University
Ithaca, NY 14850, USA

April 18, 2013

Abstract

We present a simple proof of an optimal parallel repetition theorem for public-coin arguments. Our new proof additionally yields the first “Chernoff-type” parallel repetition theorems for public-coin arguments that match the parameters of the standard Chernoff bound.

1 Introduction

Interactive proof systems, introduced by Goldwasser, Micali and Rackoff [GMR89] and Babai and Moran [BM88] are one of the central tools in both modern cryptography and complexity theory. Roughly speaking, an interactive proof system allows a prover P to convince a verifier V that some instance x is a member of a language L .

Roughly speaking, the completeness property of an interactive proof states that if $x \in L$ and both players follow their prescribed strategies, the verifier accepts with some “high probability” $1 - c(|x|)$; the soundness property, on the other hand, requires that if $x \notin L$, then no matter what strategy an adversarial prover P^* uses, the honest verifier strategy V will reject with some high probability $1 - s(|x|)$. We refer to $c(\cdot)$ as the *completeness error* of the interactive proof systems, and $s(\cdot)$ as the *soundness error*. While the original notion of an interactive proof required that the soundness property holds against *all*, even computationally unbounded, cheating provers, a relaxed notion—called *interactive arguments*—was introduced by Brassard, Chaum and Crepeau [BCC88]: In an interactive argument, we only require the soundness property to hold against computationally bounded (technically, probabilistic polynomial-time, or non-uniform polynomial-time) algorithms.

Ideally, we would like the soundness error of an interactive proof or argument systems to be negligible. But, in many settings, our starting point is a protocol with somewhat large soundness error. For example, to design an interactive argument for a language L , it may be easier to first

*© Kai-Min Chung and Rafael Pass, 2013.

[†]chung@cs.cornell.edu. Supported in part by NSF Award CNS-1217821 and Pass’ Sloan fellowship..

[‡]rafael@cs.cornell.edu. Pass is supported in part by a Alfred P. Sloan Fellowship, Microsoft New Faculty Fellowship, NSF Award CNS-1217821, NSF CAREER Award CCF-0746990, NSF Award CCF-1214844, AFOSR YIP Award FA9550-10-1-0093, and DARPA and AFRL under contract FA8750-11-2-0211. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the Defense Advanced Research Projects Agency or the US Government.

design a protocol with soundness error $1/2$. A natural approach to decrease the soundness error is through *parallel repetition*: we run k instances of the original protocol in parallel and the verifier finally accepts if all instances are accepting. It is known that parallel repetition decrease soundness error at an optimal rate for the case of interactive proofs. For arguments (i.e., computational soundness), however, surprising things start happening: The works of Bellare, Impagliazzo, and Naor [BIN97] and Petrzak and Wikström [PW07] demonstrate protocols for which parallel repetition fails to amplify soundness beyond a constant.

On the other hand, the seminal work of Bellare, Impagliazzo and Naor [BIN97] demonstrates that parallel repetition reduces the soundness error for all three-message protocols. The results of [BIN97] demonstrated that parallel repetition reduces the soundness error of such protocols at an exponential rate, but did not establish an optimal rate (i.e., reducing the soundness error from ϵ to ϵ^k). Nevertheless, the more recent work by Canetti, Halevi and Steiner [CHS05] shows that parallel repetition indeed reduces the soundness error at an optimal rate for this class of protocols.

More recently, Pass and Venkatasubramanian [PV12] consider *public-coin* protocols, and demonstrates that parallel repetition decreases the soundness error for *constant-round* public-coin protocols at an optimal rate. Håstad, Pass, Wikström and Pierzak [HPWP08] show that parallel repetition, in fact, works for *all* (not necessarily constant-round) public-coin protocols, and decreases the soundness error at an exponential rate; finally, Chung and Liu [CL10] demonstrate that it in fact decreases at an optimal rate. The non-tight analysis of [HPWP08] is quite natural and modular; on the other hand, the tight analysis from [CL10] relies on a rather complicated analysis involving directly upper-bounding the success probability of the complete reduction (which carries little intuition for why the reduction works).

In this note we revisit the works of [HPWP08] and [CL10]: we present a new and modular *tight* analysis of parallel repetition for public-coin protocols. On a high-level, our proof follows the simpler framework of [HPWP08] (and thus enjoys the same modularity and simplicity), yet at the same time providing a clear intuition for why a tight analysis can be obtained.

As an additional application of this new analysis, we obtain the first general “Chernoff-type” parallel repetition theorems for public-coin argument that matches the parameters of the standard Chernoff bound: Ideally, we would like to have a way to simultaneously decrease both the completeness and the soundness error: just as for error reduction of the class **BPP**, the idea is to consider a *threshold verifier*, who accept whenever the fraction of accepting sessions is greater than a certain threshold (that is greater than the soundness error, or else there is no hope to reduce the soundness error). For error reduction of **BPP**, it follows by a standard Chernoff bound that such an approach works. For interactive arguments, such “Chernoff-type” parallel repetition theorems where first studied by Impagliazzo, Jaiswal, and Kabanets [IJK09] for the case of three-message protocols, tighter bounds were later established by Jutla [Jut10], and finally optimal bounds were established independently Chung et al. [CLLY10], and Holenstein and Schoenebeck [HS11]. Håstad et al. [HPWP10] provide Chernoff-type parallel repetition theorems for public-coin protocols and Chung and Liu [CL10] show Chernoff-type parallel repetition theorems with parameters matching the standard Chernoff bound, but only in the regime where the threshold is a additive constant larger than the soundness error of the original protocol (and as such do not apply to protocols where the soundness and completeness error are inverse polynomials). Our new analysis yields the same parameters as the standard Chernoff bound for *any* threshold.

2 Preliminaries

2.1 Interactive Proofs and Arguments

We recall the definition of interactive proofs and arguments.

Definition 1 (Interactive Proofs/Arguments) A pair of interactive algorithms (P, V) is an *interactive proof* for a NP language L with **completeness error** c and **soundness error** s if it satisfies the following properties:

- *Completeness:* For all $x \in L$ with NP witness w ,

$$\Pr[\langle P(w), V \rangle(x) = 1] = 1 - c(|x|).$$

- *Soundness:* For all adversarial provers P^* , and for every all $x \notin L$,

$$\Pr[\langle P^*, V \rangle(x) = 1] \leq s(|x|).$$

where $\langle P, V \rangle(x)$ denotes the output of V after communicating with P if both players get x as a common input. (P, V) is an *interactive argument* for L if the soundness property holds only against all non-uniform polynomial-time adversarial provers P^* .

2.2 KullbackLeibler divergence

Lemma 2 (chain rule) Let (X_1, X_2) and (Y_1, Y_2) be random variables. We have

$$\mathbf{KL}((X_1, X_2) || (Y_1, Y_2)) = \mathbf{KL}(X_1 || Y_1) + \mathbb{E}_{x \leftarrow X_1} [\mathbf{KL}(X_2 |_{X_1=x} || Y_2 |_{Y_1=x})].$$

Lemma 3 Let X be a random variable and W a (probabilistic) event.

$$\mathbf{KL}(X|_W || X) \leq \log \frac{1}{\Pr[W]}.$$

Lemma 4 Let $\vec{X} = (X_1, \dots, X_k)$ be independent random variables, and W a (probabilistic) event.

$$\sum_{i=1}^k \mathbf{KL}(X_i|_W || X_i) \leq \mathbf{KL}(\vec{X}|_W || \vec{X})$$

Lemma 5 For every $p, q, \delta \in (0, 1)$ such that $\delta \leq p/2$, we have

$$\mathbf{KL}(p||q) - \mathbf{KL}(p - \delta||q) \leq \delta \cdot \left(\log \frac{1}{q} + \log \frac{1}{\delta} + 2 \right).$$

Proof. By definition,

$$\begin{aligned} \mathbf{KL}(p||q) &= p \log \frac{p}{q} + (1-p) \log \frac{1-p}{1-q} \\ &= p \log p + p \log \frac{1}{q} + (1-p) \log(1-p) + (1-p) \log \frac{1}{1-q} \\ \mathbf{KL}(p - \delta||q) &= (p - \delta) \log \frac{p - \delta}{q} + (1 - p + \delta) \log \frac{1 - p + \delta}{1 - q} \\ &= (p - \delta) \log(p - \delta) + (p - \delta) \log \frac{1}{q} + (1 - p + \delta) \log(1 - p + \delta) + (1 - p + \delta) \log \frac{1}{1 - q} \end{aligned}$$

By further expanding, we have

$$\begin{aligned}
(p - \delta) \log(p - \delta) &= p \log(p - \delta) - \delta \log(p - \delta) \\
&= p \log p + p \log\left(1 - \frac{\delta}{p}\right) - \delta \log(p - \delta) \\
(p - \delta) \log \frac{1}{q} &= p \log \frac{1}{q} - \delta \log \frac{1}{q} \\
(1 - p + \delta) \log(1 - p + \delta) &= (1 - p) \log(1 - p + \delta) + \delta \log(1 - p + \delta) \\
&= (1 - p) \log(1 - p) + (1 - p) \log\left(1 + \frac{\delta}{1 - p}\right) + \delta \log(1 - p + \delta) \\
(1 - p + \delta) \log \frac{1}{1 - q} &= (1 - p) \log \frac{1}{1 - q} + \delta \log \frac{1}{1 - q}
\end{aligned}$$

Therefore,

$$\begin{aligned}
&\mathbf{KL}(p||q) - \mathbf{KL}(p - \delta||q) \\
&= -p \log\left(1 - \frac{\delta}{p}\right) + \delta \log(p - \delta) + \delta \log \frac{1}{q} - (1 - p) \log\left(1 + \frac{\delta}{1 - p}\right) - \delta \log(1 - p + \delta) - \delta \log \frac{1}{1 - q} \\
&\leq -p \log\left(1 - \frac{\delta}{p}\right) + \delta \log \frac{1}{q} - \delta \log(1 - p + \delta) \\
&\leq 2\delta + \delta \log \frac{1}{q} - \delta \log \delta = \delta \cdot (\log \frac{1}{q} + \log \frac{1}{\delta} + 2),
\end{aligned}$$

where the first inequality follows by dropping negative terms, the second inequality follows by the monotonicity of logarithm and using Taylor expansion. $\blacksquare$

2.3 A Lemma on Sampling

Lemma 6 *Let (X, Y) be a joint distribution over some finite domain. Let W be a deterministic event on (X, Y) . Consider the following experiment:*

- Sample $x \leftarrow X|_W$.
- Sample $y \leftarrow Y|_{W \wedge X=x}$ using rejection sampling; i.e., sample i.i.d. $y_1, y_2, \dots \leftarrow Y|_{X=x}$ and outputs the first y_t such that $(x, y_t) \in W$.

Let T be the number of sample used in the rejection sampling. We have $\mathbb{E}[T] = \frac{1}{\Pr[W]}$.

Proof. The lemma follows by the following calculation.

$$\begin{aligned}
\mathbb{E}[T] &= \sum_x \Pr[X = x|W] \cdot \mathbb{E}[T|X = x] \\
&= \sum_x \Pr[X = x|W] \cdot \frac{1}{\Pr[W|X = x]} \\
&= \sum_x \frac{\Pr[X = x \wedge W]}{\Pr[W]} \cdot \frac{\Pr[X = x]}{\Pr[W \wedge X = x]} \\
&= \sum_x \frac{\Pr[X = x]}{\Pr[W]} = \frac{1}{\Pr[W]}.
\end{aligned}$$

$\blacksquare$

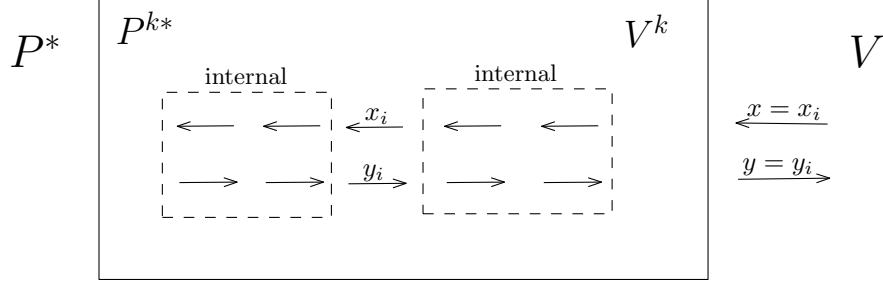


Figure 1: Interaction between P^* and V : P^* embeds the external verifier V in session i of V^k and internally emulates P^{k*} and the remaining $k - 1$ sessions $-i$ of V^k while forwarding P^{k*} 's message y for session i to V .

3 Parallel Repetition of Two-message Public-coin Protocols

As a starting point, we provide the key elements needed to prove an optimal parallel repetition theorem for two-message public-coin protocols. To set-up some notation, let us consider two-message public-coin protocols (P, V) , where the verifier V sends a uniformly random first-message x to P , receives back a second-message y , and finally *deterministically* decides to accept or reject based on the transcript (x, y) . We denote by (P^k, V^k) the k -fold parallel repetition of (P, V) ; here V^k sends a message $\vec{x} = (x_1, \dots, x_k)$, receives back $\vec{y} = (y_1, \dots, y_k)$, and accepts iff (x_i, y_i) is accepting for every coordinate $i \in [k]$. We refer to the different parallel executions of the protocol (P, V) inside (P^k, V^k) as the parallel *sessions*.

To prove that parallel repetition reduces the soundness error, we show how to transform any *parallel prover* P^{k*} that convinces V^k with probability ϵ to a *single-instance* prover P^* that convinces V with probability close to $\epsilon^{1/k}$. This implies that parallel repetition reduces the soundness error at an essentially optimal rate (from δ to δ^k). We may without loss of generality assume that P^{k*} is deterministic—its optimal random coins can always be fixed non-uniformly.¹

More precisely, P^* will internally emulate an execution of P^{k*} and use this execution in order to convince an external verifier V . On a high-level, the idea is quite straight forward. P^* picks one of the k sessions, i ; this session will be externally forwarded (between P^{k*} and V), and all the other sessions, $-i$, will be appropriately emulated internally. In other words, the external verifier V is “embedded” in some session i of V^k , and P^* internally emulates P^{k*} and the remaining $k - 1$ sessions $-i$ of V^k while forwarding P^{k*} 's message y for session i to V ; see Figure 1.

Recall that since we have assumed that P^{k*} is deterministic, the interaction between P^{k*} and V^k is determined solely by V^k 's message $\vec{x}$. Now, given an external message x , P^* needs to decide *the session*, i , into which to embed V 's message x (letting $x_i = x$), and to *choose the remaining* $k - 1$ *messages* $\vec{x}_{-i}$. Consider the following simple strategy for doing this: Upon receiving x , P^* picks $i \in [k]$ uniformly at random, and lets $x_i = x$. P^* then repeatedly samples $\vec{x}_{-i}$ at random and queries P^{k*} with the sampled message $\vec{x} = (x_i, \vec{x}_{-i})^2$, until P^{k*} convinces V^k on the query $\vec{x}$; if this happens, P^* externally forwards P^{k*} 's answer y_i to V . Additionally, if the number of samples exceeds a certain polynomial bound M (to be determined shortly), P^* gives up.

¹Alternatively, “close to optimal” coins can be uniformly fixed by sampling.

²This notation means that x_i is put into coordinate i of $\vec{x}$ and $\vec{x}_{-i}$ are put at coordinates $-i$.

To analyze the success probability of P^* , let us first allow P^* to make an unbounded number of samples (i.e., set $M = \infty$). As we shall see, if P^{k*} convinces V^k with probability ϵ , then P^* convinces V with probability $\geq \epsilon^{1/k}$. We then deal with the bounded-sample case at the end of the section (looking forward, as long as we make $\text{poly}(1/\epsilon)$ queries, having such a cut-off only slightly affects the success probability of P^*).

The main idea for analyzing (the unbounded sample version of) P^* is to consider an **Ideal** experiment, where P^* succeeds with probability 1 and next show that the actual execution of (P^*, V) , referred to as the **Real** experiment, and the **Ideal** experiment are close (using an appropriate choice of a distance measure), from which we can conclude that P^* succeeds with high probability in the **Real** experiment.

Let us start by formalizing the **Real** experiment.

The Real Experiment Consider an execution of (P^*, V) . V starts by selecting a uniformly random string $x \in \{0, 1\}^n$, where n is the length of V 's first message. Next, P^* , given x , selects a random coordinate i , lets $x_i = x$ and samples remaining $k-1$ coordinates $\vec{x}_{-i}$ conditioned on $P^{k*}(\vec{x})$ convincing V^k . If such a string $\vec{x}_{-i}$ exists, then P^* succeeds (since P^* can make an unbounded number of queries), and P^* fails otherwise (formally, if no such string exists, we let $\vec{x}_{-i} = \perp$). The output of the experiment is defined to be $(i, \vec{x})$.

First note that to prove that parallel repetition works (at an optimal rate) we need to show that P^* convinces V in the **Real** experiment with probability at least $\epsilon^{1/k}$. Secondly, observe that an equivalent way of defining the output $(i, \vec{x})$ of the experiment is as follows: uniformly sample $i \in [k]$, uniformly sample $x_i \in \{0, 1\}^n$, and finally uniformly sample $\vec{x}_{-i} \in \{0, 1\}^{(k-1)n}$ conditioned on $P^{k*}(\vec{x})$ convincing V^k .

The Ideal Experiment Let us turn to defining the **Ideal** experiment. The experiment is defined identically to the **Real** experiment, except that now we additionally select x_i conditioned on $P^{k*}(\vec{x})$ convincing V^k ; that is, uniformly sample $i \in [k]$, uniformly sample $x_i \in \{0, 1\}^n$ conditioned on $P^{k*}(\vec{x})$ convincing V^k , and finally uniformly sample $\vec{x}_{-i} \in \{0, 1\}^{(k-1)n}$ conditioned on $P^{k*}(\vec{x})$ convincing V^k ; again, the output of the experiment is defined to be $(i, \vec{x})$. Note that an equivalent way of defining the **Ideal** experiment is to uniformly sampling $i \in [k]$, and then directly uniformly sample $\vec{x} \in \{0, 1\}^{kn}$ conditioned on $P^{k*}(\vec{x})$ convincing V^k . Since P^{k*} convinces V^k with positive probability, it thus follows that in the **Ideal** experiment P^* convinces V with probability 1.

Going from Ideal to Real Observe that the only difference between the **Real** and the **Ideal** experiments is that in **Real** x_i is sampled uniformly at random, and in **Ideal** it is sampled at random conditioned on P^{k*} convincing V^k . Let W denote the event that P^{k*} convinces V^k . The statistical distance between the two experiments is thus the average (over i) statistical distance between x_i and $x_i|_W$. The following lemma due to Ran Raz [Raz98], developed in the context of parallel repetition of two-prover games (with statistical soundness), and first used by Impagliazzo, Jaiswal and Kabanets [IJK07] in the context of parallel repetition of three-round arguments, allows us to upper bound this distance.

Lemma 7 (Raz's Lemma [Raz98]) *Let $\vec{X} = (X_1, \dots, X_k)$ be independent random variables and*

W be an event. Then,

$$\frac{1}{k} \sum_{i=1}^k \mathbf{SD}(X_i|_W, X_i) \leq \sqrt{\frac{\log(1/\Pr[W])}{k}}.$$

Raz’s Lemma states that conditioning on a not “too small” event W cannot change the marginal distribution of X_i by too much (on average). In particular, the average statistical distance scales logarithmically with the probability of the event W . Raz’s Lemma together with the fact that P^* convinces V with probability 1 in the **Ideal** experiment implies that P^* convinces V in the **Real** experiment with probability at least $1 - \sqrt{(\log(1/\Pr[W]))/k}$, where by definition $\Pr[W] = \epsilon$. This already suffices to prove that parallel repetition reduces the soundness error at an exponential rate, but it does not give the “tight” bound (i.e., $\epsilon^{1/k}$). The reason that Raz’s Lemma does not provide a tight bound is that in our context, statistical distance is not the right distance measure between the **Real** and the **Ideal** experiments. In fact, the proof of Raz’s Lemma first provides a bound on the Kullback-Leibler divergence (KL divergence, for short) between the random variables, and then arrives a bound on their statistical distance by relying on a general bound between statistical distance and KL divergence.³ The “translation” between KL divergence and statistical distance, however, incurs a quadratic loss. By directly working with KL divergence, we can avoid it.⁴ Let us thus state Raz’s Lemma in its “KL form”.

Lemma 8 (Raz’s Lemma – KL version) *Let $\vec{X} = (X_1, \dots, X_k)$ be independent random variables and W be an event. Then,*

$$\frac{1}{k} \sum_{i=1}^k \mathbf{KL}(X_i|_W || X_i) \leq \frac{\log(1/\Pr[W])}{k}.$$

We omit the proof of Raz’s Lemma, but let us simply remark that the proof follows by a few lines of elementary (but clever) manipulations of KL divergence. By the chain rule for KL divergence, it follows that the KL divergence between the **Ideal** and **Real** experiments is at most $(\log(1/\Pr[W]))/k$.⁵ Let us now show how to get a lower bound on the success probability of P^* in the **Real** experiment. Let Suc_{Real} and $\text{Suc}_{\text{Ideal}}$ be indicator variables that indicate, respectively, whether P^* convinces V in the **Real** and the **Ideal** experiments.

$$\frac{\log(1/\Pr[W])}{k} \geq \mathbf{KL}(\text{Ideal} || \text{Real}) \geq \mathbf{KL}(\text{Suc}_{\text{Ideal}} || \text{Suc}_{\text{Real}}) = 1 \cdot \log \frac{1}{\Pr[\text{Suc}_{\text{Real}} = 1]}, \quad (1)$$

which implies that $\Pr[\text{Suc}_{\text{Real}} = 1] \geq \epsilon^{1/k}$ since $\Pr[W] = \epsilon$. The second inequality follows since applying the same function to two distributions can only decrease their KL divergence, whereas the last equality follows by the definition of KL divergence and the fact that $\Pr[\text{Suc}_{\text{Ideal}} = 1] = 1$. This concludes that P^* convinces V with probability at least $\epsilon^{1/k}$ in the **Real** experiment.

³Recall that $\mathbf{KL}(X||Y) = \sum_{x \in \text{supp}(X)} \Pr[X = x] \cdot \log \frac{\Pr[X=x]}{\Pr[Y=x]}$.

⁴A similar phenomena occurred already in the context of parallel repetition for “free” two-prover games; see [BRR⁺09].

⁵We note that the KL divergence is not symmetric, so it does not imply that the KL divergence between the **Real** and **Ideal** experiments is small (in fact, it’s infinite).

Handling The Bounded-Sample Case. In our analysis so far we have assumed that P^* can make an unbounded number of samples. Let us now show that its success probability is still high even if we impose a polynomial bound M on the number of samples it can make (and thus P^* becomes efficient). Let us first consider the *Ideal* experiment. The main observation is that, in the *Ideal* experiment, *in expectation*, P^* only needs to make $1/\epsilon$ samples to pick $\vec{x}_{-i}$ conditioned on $P^{k*}(\vec{x})$ convincing V^k (since x_i is also picked conditioned on $P^{k*}(\vec{x})$ convincing V^k , and P^{k*} convinces V^k with probability ϵ). Thus, if the allowed number of samples M is sufficiently larger than $1/\epsilon$, then by the Markov inequality, P^* can successfully convince V^k with probability “almost” 1, even if we restrict P^* to use at most M samples.⁶ Since the *Ideal* and the *Real* experiments are statistically close, this directly yields a lower bound on the success probability of P^* in the *Real* experiment. But as we saw, working with statistical distance does not give the tight bound. To obtain a tight bound, we again work with KL divergence. Here, the only difference is that $\Pr[\text{Suc}_{\text{Ideal}} = 1]$ is no longer 1, but can be made arbitrarily (inverse polynomially) close to 1 by increasing M . This is sufficient to conclude that $\Pr[\text{Suc}_{\text{Real}} = 1]$ can be made arbitrarily (inverse polynomially) close to $\epsilon^{1/k}$ as well (since the KL divergence of two binary random variables is a “smooth” function of the probabilities of both random variables).

4 Parallel Repetition for General Public-coin Protocols

Let us turn to demonstrate a parallel repetition theorem for general public-coin protocols with an arbitrary number of rounds. The first question to address is: What should the strategy of P^* be? As before, we let P^* externally interact with V by internally emulating an interaction of (P^{k*}, V^k) and embedding its external interaction with V into a random session i . Recall that for the case of two-message public-coin protocols, P^* just needs to select verifier messages $\vec{x}_{-i}$ for sessions $-i$, and does so in a way that guarantees that it will convince V (if such messages $\vec{x}_{-i}$ exist). For m -round protocols, P^* needs to select verifier messages $\vec{x}_{1,-i}, \dots, \vec{x}_{m,-i}$ for each round j , but must do so in a round-by-round fashion. That is, after receiving a message x_j from the external verifier V , it sets $x_{j,i} = x_j$ and must pick $\vec{x}_{j,-i}$ before knowing what $x_{j',i}$ is for $j' > m$; as a consequence, we can no longer pick messages $x_{j,-i}$ that guarantees “success” (even if we have an unbounded number of samples).

A first approach for picking $x_{j,-i}$, explored by [PV12], consists of letting P^* *greedily* select messages $\vec{x}_{j,-i}$ for sessions $-i$ that maximize (or rather “approximately” maximize) P^* ’s probability of convincing V ; this can be done using a *recursive sampling strategy*—roughly speaking, in each round j , P^* needs to evaluate how well P^* would do in the remaining rounds, and picks the best messages based on this evaluation (we briefly return to this approach in Section ??). [PV12] shows that by employing such a recursive sampling strategy, a tight parallel repetition theorem can be established. But, due to the recursive sampling, the running-time of the reduction becomes exponential in the number of rounds m , and this approach can thus only be used to get a parallel repetition theorem for *constant-round* public-coin protocols.

As it turns out, an even simpler *rejection sampling* strategy, first explored by [HPWP10], works: We consider a prover, P_{rej}^* , that selects the session $i \in [k]$ uniformly at random (just as before), and then at each round j , upon receiving the external verifier V ’s message $x_{j,i}$, P_{rej}^* selects $\vec{x}_{j,-i}$ using

⁶Since $M > 1/\epsilon$, we only get an efficient reduction as long as ϵ is an inverse polynomial. As a consequence, parallel repetition of arguments cannot decrease the soundness error beyond being “negligible”. As shown by [DJMW12], under some cryptographic assumptions, this is inherent.

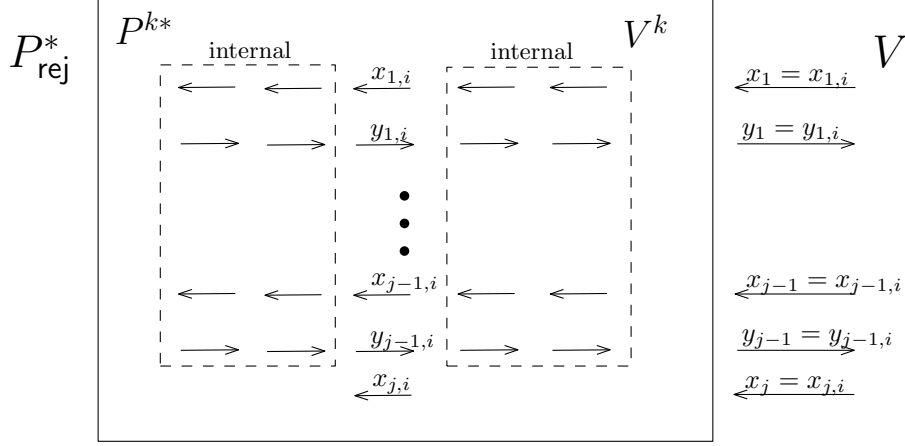


Figure 2: Interaction between P^*_{rej} and V .

rejection sampling as follows: P^*_{rej} repeatedly samples a random continuation of (P^{k*}, V^k) until it finds an *accepting continuation*, i.e., V^k accepts at the end of interaction (or a certain a-prior bound M on the number of samples is reached, in which case P^* aborts and fails). Then, P^* selects the corresponding messages in the accepting continuation as the messages of V_{-i} at round j . As for the two-message case, let us first consider the unbounded sample case (when $M = \infty$). That is, at each round j , P^*_{rej} simply selects $\vec{x}_{j,-i}$ conditioned on P^{k*} convincing V^k . See Figure 2 for an illustration.

As we shall see now, the analysis for the two-message case can be generalized to provide a tight lower bound on the success probability of P^*_{rej} . (The original analysis of [HPWP10] relied on *Ideal* v.s. *Real* paradigm considered in Section 3, but only showed that parallel repetition reduces the soundness error at an exponential rate; it did not provide a tight parallel repetition theorem. A tight analysis was first provided by [CL10] by directly (inductively) analyzing the success probability of P^*_{rej} . [CP13] provided an alternative tight analysis relying on *Ideal* v.s. *Real* paradigm.) As before, let's consider a *Real* experiment where P^*_{rej} interacts with V (that is, at each round j , $x_{j,i}$ is uniformly sampled and $\vec{x}_{j,-i}$ is uniformly sampled conditioned on P^{k*} convincing V^k). We will argue that if P^{k*} convinces V^k with probability ϵ , then P^*_{rej} convinces V with probability $\geq \epsilon^{1/k}$ in the *Real* experiment.

Again, let us compare it to an *Ideal* experiment where the external verifier also, at each round j , uniformly picks $x_{i,j}$ conditioned on P^{k*} convincing V^k . It follows (as before) that in the *Ideal* experiment, P^*_{rej} convinces V with probability 1. As before, let us now bound the distance between the *Real* and the *Ideal* experiments. Can we just use Raz's Lemma as before?

Consider a set of m "hybrid" experiments, where in H_j , the messages in the first j rounds are selected just as in *Ideal* (i.e., both $x_{j',i}$ and $\vec{x}_{j',-i}$ for $j' \leq j$ are sampled conditioned on P^{k*} convincing V^k), and the remaining $m - j$ rounds are selected just as in *Real* (i.e., for $j' > j$, only $\vec{x}_{j',-i}$ is sampled conditioned on P^{k*} convincing V^k , but $x_{j',i}$ is uniformly sampled without any conditioning). Clearly $H_0 = \text{Real}$ and $H_m = \text{Ideal}$. Furthermore, the only difference between two consecutive hybrids $j - 1$ and j is whether $x_{j,i}$ is sampled conditioned on P^{k*} convincing V^k or not, where i is uniformly chosen. Thus, it would seem we can just use Raz's Lemma exactly as

in the two-message case to bound the statistical distance between two such hybrids. This almost works. The only difference from the two-message case is that there now is a “prefix”—the earlier messages—that the event we condition on—i.e., P^{k*} convincing V^k —may depend on. The following slight generalization of Raz’s Lemma shows exactly this.

Lemma 9 (Raz’s Lemma—“Prefix-version” [Raz98]) *Let $(H, \vec{X}) = (H, X_1, \dots, X_k)$ be independent random variables and W be an event. Then,*

$$\frac{1}{k} \sum_{i=1}^k \text{SD}((H, X_i)|_W, (H|_W, X_i)) \leq \sqrt{\frac{\log(1/\Pr[W])}{k}}.$$

Let again W be the event that P^{k*} convinces V^k . The lemma directly implies that the statistical distance between any two consecutive hybrids H_{j-1} and H_j is at most $\sqrt{(\log(1/\Pr[W]))/k}$. Thus, by the triangle-inequality, the statistical distance between the **Real** and the **Ideal** experiments is at most $m \cdot \sqrt{(\log(1/\Pr[W]))/k}$, which yields a lower bound on the success probability of P_{rej}^* in the **Real** experiment that suffices to demonstrate that parallel repetition reduces the soundness error at an exponential rate.

Again, however, the bound is not tight due to the use of statistical distance. Additionally, due to the “hybrid argument” we have furthermore incurred a linear loss in the number of rounds m (thus, to make the soundness error small we need the number of parallel repetitions to grow polynomially with the number of rounds in the protocol). Perhaps surprisingly, we can still obtain a tight bound by working with KL divergence. In particular, by a few lines of elementary manipulations of KL divergence one can show that the KL divergence between the **Ideal** and the **Real** experiments is upper bounded by the same quantity as in the KL version of Raz lemma (Lemma 8).

Lemma 10 $\text{KL}(\text{Ideal}||\text{Real}) \leq \frac{\log(1/\Pr[W])}{k}$.

A tight lower bound $\epsilon^{1/k}$ on the success probability of P_{rej}^* in the **Real** experiment can now be derived by identically the same calculation as in Eq. (1). Finally, the bounded-sample case can be handled in essentially the same way as in the two-message case.

4.1 Formal Proof

In this section, we present a formal proof of tight parallel repetition theorem for public-coin protocols.

Theorem 11 *Let (P, V) be a public-coin interactive argument for a language L . There exists an oracle adversarial prover $P^{(\cdot)*}$ such that for every $k \in \mathbb{N}$, input $z \in \{0, 1\}^*$, every $\epsilon, \xi \in (0, 1)$, and every deterministic parallel adversarial prover P^{k*} , if*

$$\Pr[\langle P^{k*}, V^k \rangle(z) = 1] \geq \epsilon,$$

then

$$\Pr[\langle P^{(P^{k*})*}, V \rangle(k, \epsilon, \xi)(z) = 1] \geq \epsilon^{1/k} \cdot (1 - \xi).$$

Furthermore, $P^{(\cdot)*}$ runs in time $\text{poly}(|z|, k, \epsilon^{-1}, \xi^{-1})$ given oracle access to P^{k*} .

Proof. Let m denote the round complexity of (P, V) . Let us consider a $P_{\text{rej}}^{(\cdot)*}$ that interacts with V by the aforementioned *rejection sampling* with $M = \Theta(\frac{m}{\epsilon\xi} \log \frac{1}{\xi})$. Specifically, P_{rej}^* selects the session $i \in [k]$ uniformly at random, and then at each round j , upon receiving the external verifier V 's message $x_{j,i}$, P_{rej}^* selects $\vec{x}_{j,-i}$ using rejection sampling as follows: P_{rej}^* repeatedly samples a random continuation of (P^{k*}, V^k) until it finds an *accepting continuation*, i.e., V^k accepts at the end of interaction (or $M = \Theta(\frac{m}{\epsilon\xi} \log \frac{1}{\xi})$ samples is reached, in which case P_{rej}^* aborts and fails). Then, P_{rej}^* selects the corresponding messages in the accepting continuation as the messages of V_{-i} at round j .

By inspection, $P^{(\cdot)*}$ runs in time $\text{poly}(|z|, k, \epsilon^{-1}, \xi^{-1})$ on input z, k, ϵ , and ξ . It remains to show that if P^{k*} convinces V^k with probability at least ϵ , then $P^{(\cdot)*}$ convinces V with probability at least $\epsilon^{1/k} \cdot (1 - \xi)$. Let W denote the event that P^{k*} convinces V^k in the execution of $\langle P^{k*}, V^k \rangle(z)$. We consider the following **Real** experiment, which is the same as the execution of $\langle P_{\text{rej}}^{(P^{k*})*}(k, \epsilon, \xi), V \rangle(z)$ except that P_{rej}^* takes an unbounded number of samples (i.e., set $M = \infty$).

The Real Experiment Consider an execution of (P_{rej}^*, V) as follows. At beginning, P_{rej}^* selects a random coordinate $i \in [k]$. Then at each round $j \in [m]$, V selects a uniformly random $x_{j,i}$, and P_{rej}^* selects a random $\vec{x}_{j,-i}$ conditioned on W using rejection sampling (namely, repeatedly samples a random continuation of (P^{k*}, V^k) until it finds an *accepting continuation*, i.e., V^k accepts at the end of interaction, and selects the corresponding $\vec{x}_{j,-i}$). Let T_j denotes the number of samples P_{rej}^* takes. If no such $\vec{x}_{j,-i}$ exists, then P_{rej}^* *fails*, and we set $T_j = \infty$ and all remaining $\vec{x}_{j,-i}, \vec{x}_{j+1,-i}, \dots, \vec{x}_m = \perp$. P_{rej}^* *succeeds* if it does not fail. The output of the experiment is defined to be $(i, \vec{x}_1, \dots, \vec{x}_m)$.

Note that the event that $P^{(\cdot)*}$ convinces V in $\langle P^{(P^{k*})*}(k, \epsilon, \xi), V \rangle(z)$ corresponds to the event that in the **Real** experiment, P^* succeeds and $T_j \leq M$ for every $j \in [m]$. Let Suc_{Real} be the indicator random variable of this event. Our goal is to lower bound

$$\Pr[\langle P^{(P^{k*})*}(k, \epsilon, \xi), V \rangle(z) = 1] = \Pr[\text{Suc}_{\text{Real}} = 1].$$

We next compare it with an **Ideal** experiment, which is identical to the **Real** experiment, except that the messages $x_{1,i}, \dots, x_{m,i}$ are also selected conditioned on W .

The Ideal Experiment At beginning, P_{rej}^* selects a random coordinate $i \in [k]$. Then at each round $j \in [m]$, V selects a random $x_{j,i}$ conditioned on W , and P_{rej}^* selects a random $\vec{x}_{j,-i}$ conditioned on W using rejection sampling. Let T_j denotes the number of samples P_{rej}^* takes. The output of the experiment is defined to be $(i, \vec{x}_1, \dots, \vec{x}_m)$.

Note that sampling random $x_{1,i}, \vec{x}_{1,-i}, \dots, x_{m,i}, \vec{x}_{m,-i}$ conditioned on W step by step is equivalent to sampling the whole $\vec{x}_1, \dots, \vec{x}_m$ conditioned on W . Thus, the output distribution of the **Ideal** experiment is simply a uniformly random coordinate $i \in [k]$ and a uniformly random *accepting* transcript $(\vec{x}_1, \dots, \vec{x}_m)$, and P_{rej}^* never fails in the **Ideal** experiment. Let $\text{Suc}_{\text{Ideal}}$ be the corresponding indicator random variable of Suc_{Real} in the **Ideal** experiment; that is, $\text{Suc}_{\text{Ideal}}$ is the indicator random variable of the event that $T_j \leq M$ for every $j \in [m]$.

In what follows, we will show that (i) $\Pr[\text{Suc}_{\text{Ideal}} = 1] \geq m/M\epsilon$ and (ii) $\mathbf{KL}(\text{Ideal}||\text{Real}) \leq (\log(1/\Pr[W]))/k$, and derive the desired lower bound on $\Pr[\text{Suc}_{\text{Real}} = 1]$ from them.

Claim 12 $\Pr[\text{Suc}_{\text{Ideal}} = 1] \geq 1 - m/M\epsilon$.

Proof. Note that in the **Ideal** experiment, for every $i \in [k]$ and $j \in [m]$, the prefix $(\vec{x}_1, \dots, \vec{x}_{j-1}, x_{j,i})$ is chosen randomly conditioned on W and then P_{rej}^* selects a random $\vec{x}_{j,-i}$ conditioned on W using rejection sampling. Applying Lemma 6 with $X = (\vec{X}_1, \dots, \vec{X}_{j-1}, X_{j,i})$, $Y = X_{j,-i}$ and event W implies that $\mathbb{E}[T_j] = 1/\Pr[W] \leq 1/\epsilon$ for every $j \in [m]$. By the Markov inequality, we have $\Pr[T_j \leq M] \geq 1 - 1/M\epsilon$ for every $j \in [m]$, and thus it follows by an union bound that $\Pr[\text{Suc}_{\text{Ideal}} = 1] \geq 1 - (m/M\epsilon)$. $\blacksquare$

Claim 13 $\text{KL}(\text{Ideal}||\text{Real}) \leq (\log(1/\Pr[W]))/k$.

Proof. It is instructive to first prove the one-round case (i.e., $m = 1$), which is equivalent to the KL-version of Raz's Lemma. In this case by definition, $\text{Ideal} = (I, \vec{X}_1|_W)$ and $\text{Real} = (I, X_{1,I}, \vec{X}_{1,-I}|_{W, X_{1,I}})$. By applying chain rule, we have

$$\begin{aligned} \text{KL}(\text{Ideal}||\text{Real}) &= \text{KL}(I||I) + \mathbb{E}_I \left[\text{KL} \left(\vec{X}_1|_W || (X_{1,I}, \vec{X}_{1,-I}|_{W, X_{1,I}}) \right) \right] \\ &= \frac{1}{k} \sum_{i=1}^k \text{KL} \left(\vec{X}_1|_W || (X_{1,i}, \vec{X}_{1,-i}|_{W, X_{1,i}}) \right). \end{aligned}$$

For each term $\text{KL}(\vec{X}_1|_W || (X_{1,i}, \vec{X}_{1,-i}|_{W, X_{1,i}}))$, by applying chain rule again, we have

$$\begin{aligned} &\text{KL} \left(\vec{X}_1|_W || (X_{1,i}, \vec{X}_{1,-i}|_{W, X_{1,i}}) \right) \\ &= \text{KL}(X_{1,i}|_W || X_{1,i}) + \mathbb{E}_{X_{1,i}|_W} [\text{KL}(\vec{X}_{1,-i}|_{W, X_{1,i}} || \vec{X}_{1,-i}|_{W, X_{1,i}})] \\ &= \text{KL}(X_{1,i}|_W || X_{1,i}). \end{aligned}$$

Applying Lemma 4,

$$\frac{1}{k} \sum_{i=1}^k \text{KL} \left(\vec{X}_1|_W || (X_{1,i}, \vec{X}_{1,-i}|_{W, X_{1,i}}) \right) = \frac{1}{k} \sum_{i=1}^k \text{KL}(X_{1,i}|_W || X_{1,i}) \leq \frac{1}{k} \text{KL}(\vec{X}_1|_W || \vec{X}_1).$$

Therefore, by Lemma 3,

$$\text{KL}(\text{Ideal}||\text{Real}) \leq \frac{1}{k} \text{KL}(\vec{X}_1|_W || \vec{X}_1) \leq \frac{\log(1/\Pr[W])}{k}.$$

We proceed to consider the general case, which is proved by the same calculation, except that we first apply an additional chain rule to break up terms corresponding to each round.

$$\text{KL}(\text{Ideal}||\text{Real}) = \sum_{j=1}^m \mathbb{E}_{I, \vec{X}_{<j}|_W} \left[\text{KL}(\vec{X}_j|_{W, \vec{X}_{<j}} || (X_{j,I}|_{W, \vec{X}_{<j}}, \vec{X}_{j,-I}|_{W, \vec{X}_{<j}, X_{j,I}})) \right].$$

Now, for each term, the same calculation as before using Lemma 4 shows that

$$\mathbb{E}_{I, \vec{X}_{<j}|_W} \left[\text{KL}(\vec{X}_j|_{W, \vec{X}_{<j}} || (X_{j,I}|_{W, \vec{X}_{<j}}, \vec{X}_{j,-I}|_{W, \vec{X}_{<j}, X_{j,I}})) \right] \leq \frac{1}{k} \mathbb{E}_{\vec{X}_{<j}|_W} \left[\text{KL}(X_j|_{W, \vec{X}_{<j}} || X_j|_{\vec{X}_{<j}}) \right].$$

Applying another chain rule and Lemma 3 gives,

$$\mathbf{KL}(\text{Ideal}||\text{Real}) \leq \frac{1}{k} \mathbb{E}_{\vec{X}_{<j}|W} \left[\mathbf{KL}(X_j|_{W, \vec{X}_{<j}} || X_j|_{\vec{X}_{<j}}) \right] = \frac{1}{k} \mathbf{KL}(\vec{X}_{\leq m}|_W || \vec{X}_{\leq m}) \leq \frac{\log(1/\Pr[W])}{k}$$

■

We now derive the desired lower bound on the probability $\Pr[\text{Suc}_{\text{Real}} = 1]$ using Claim 12 and 13. Let $q = \Pr[\text{Suc}_{\text{Real}} = 1]$ and $\delta = m/M\epsilon$. Claim 13 implies that

$$\mathbf{KL}(1 - \delta || q) \leq \mathbf{KL}(\text{Suc}_{\text{Ideal}} || \text{Suc}_{\text{Real}}) \leq \mathbf{KL}(\text{Ideal} || \text{Real}) \leq (\log(1/\Pr[W]))/k \leq \log(\epsilon^{-1/k}),$$

where the first inequality follows by Claim 12, the second inequality follows since applying the same function to two distributions can only decrease their KL divergence, and the last inequality follows by the fact that $\Pr[W] \geq \epsilon$. By Lemma 5, we have

$$\mathbf{KL}(1 || q) \leq \mathbf{KL}(1 - \delta || q) + \frac{\delta}{4} + \delta \log \frac{1}{\delta} \leq \log(\epsilon^{-1/k}) + \frac{\delta}{4} + \delta \log \frac{1}{\delta}.$$

By definition, $\mathbf{KL}(1 || q) = \log(1/q)$, and thus

$$q \geq e^{-(\log \epsilon^{-1/k} + \frac{\delta}{4} + \delta \log \frac{1}{\delta})} \geq \epsilon^{1/k} \cdot \left(1 - \frac{\delta}{4} - \delta \log \frac{1}{\delta}\right) \geq \epsilon^{1/k} \cdot (1 - \xi),$$

where the second inequality uses $e^{-x} \geq 1 - x$. This completes the proof. ■

Chernoff-type theorem here?

References

- [BCC88] Gilles Brassard, David Chaum, and Claude Crépeau. Minimum disclosure proofs of knowledge. *Journal of Computer and System Sciences*, 37(2):156–189, 1988.
- [BIN97] Mihir Bellare, Russell Impagliazzo, and Moni Naor. Does parallel repetition lower the error in computationally sound protocols? In *FOCS*, pages 374–383, 1997.
- [BM88] László Babai and Shlomo Moran. Arthur-Merlin games: A randomized proof system, and a hierarchy of complexity classes. *J. Comput. Syst. Sci.*, 36(2):254–276, 1988.
- [BRR⁺09] Boaz Barak, Anup Rao, Ran Raz, Ricky Rosen, and Ronen Shaltiel. Strong parallel repetition theorem for free projection games. In *APPROX-RANDOM*, pages 352–365, 2009.
- [CHS05] Ran Canetti, Shai Halevi, and Michael Steiner. Hardness amplification of weakly verifiable puzzles. In *TCC*, pages 17–33, 2005.
- [CL10] Kai-Min Chung and Feng-Hao Liu. Parallel repetition theorems for interactive arguments. In *TCC*, pages 19–36, 2010.
- [CLLY10] Kai-Min Chung, Feng-Hao Liu, Chi-Jen Lu, and Bo-Yin Yang. Efficient string-commitment from weak bit-commitment. In Masayuki Abe, editor, *ASIACRYPT*. Springer-Verlag, December 2010.

- [CP13] Kai-Min Chung and Rafael Pass. Parallel repetition theorem for public-coin protocols. Manuscript in preparation, 2013.
- [DJMW12] Yevgeniy Dodis, Abhishek Jain, Tal Moran, and Daniel Wichs. Counterexamples to hardness amplification beyond negligible. In *TCC*, pages 476–493, 2012.
- [GMR89] Shafi Goldwasser, Silvio Micali, and Charles Rackoff. The knowledge complexity of interactive proof-systems. *SIAM Journal on Computing*, 18(1):186–208, 1989.
- [HPWP08] Johan Håstad, Rafael Pass, Douglas Wikström, and Krzysztof Pietrzak. An efficient parallel repetition theorem. Unpublished manuscript, 2008.
- [HPWP10] Johan Håstad, Rafael Pass, Douglas Wikström, and Krzysztof Pietrzak. An efficient parallel repetition theorem. In *TCC*, pages 1–18, 2010.
- [HS11] Thomas Holenstein and Grant Schoenebeck. General hardness amplification of predicates and puzzles - (extended abstract). In *TCC*, pages 19–36, 2011.
- [IJK07] Russell Impagliazzo, Ragesh Jaiswal, and Valentine Kabanets. Chernoff-type direct product theorems. In *CRYPTO*, pages 500–516, 2007.
- [IJK09] Russell Impagliazzo, Ragesh Jaiswal, and Valentine Kabanets. Chernoff-type direct product theorems. *J. Cryptology*, 22(1):75–92, 2009.
- [Jut10] Charanjit S. Jutla. Almost optimal bounds for direct product threshold theorem. In *TCC*, pages 37–51, 2010.
- [PV12] Rafael Pass and Muthuramakrishnan Venkitasubramaniam. A parallel repetition theorem for constant-round arthur-merlin proofs. *Transactions on Computation Theory*, 4(4):10, 2012.
- [PW07] Krzysztof Pietrzak and Douglas Wikström. Parallel repetition of computationally sound protocols revisited. In *TCC*, pages 86–102, 2007.
- [Raz98] Ran Raz. A parallel repetition theorem. *SIAM J. Comput.*, 27(3):763–803, 1998.

Online/Offline Attribute-Based Encryption

Susan Hohenberger¹ and Brent Waters²

Johns Hopkins University and University of Texas at Austin

Abstract. Attribute-based encryption (ABE) is a type of public key encryption that allows users to encrypt and decrypt messages based on user attributes. For instance, one can encrypt a message to any user satisfying the boolean formula (“crypto conference attendee” AND “PhD student”) OR “IACR member”. One drawback is that encryption and key generation computational costs scale with the complexity of the access policy or number of attributes. In practice, this makes encryption and user key generation a possible bottleneck for some applications.

To address this problem, we develop new techniques for ABE that split the computation for these algorithms into two phases: a preparation phase that does the vast majority of the work to encrypt a message or create a secret key *before* it knows the message or the attribute list/access control policy that will be used (or even the size of the list or policy). A second phase can then rapidly assemble an ABE ciphertext or key when the specifics become known. This concept is sometimes called “online/offline” encryption when only the message is unknown during the preparation phase; we note that the addition of unknown attribute lists and access policies makes ABE significantly more challenging.

One motivating application for this technology is mobile devices: the preparation work can be performed while the phone is plugged into a power source, then it can later rapidly perform ABE operations on the move without significantly draining the battery.

1 Introduction

Attribute-Based Encryption (ABE) was introduced by Sahai and Waters [20] as a more expressive form of encryption where one can encrypt according to some policy. For example, in a large corporate setting one might encrypt data to the policy of (“PROCUREMENT” AND “MANAGER”) OR “ACCOUNTING”. There are two main flavors of ABE. In Key-Policy ABE [10], a key is associated with a boolean formula ϕ and a ciphertext with a set S of attributes. One can decrypt iff the set S satisfies the formula ϕ . Alternatively, in Ciphertext-Policy ABE the roles are flipped; a key is associated with a set of attributes and the ciphertext with an access formula.

One challenge in building systems that use Attribute-Based Encryption is that the added functionality may come with a significant cost compared to standard public key cryptography. Consider a Key-Policy ABE system. Here the encryption time will scale with the number of attributes assigned to the ciphertext and key generation time will scale with the size of the boolean formula

ascribed to a user's private key. These costs could impact several applications. If the encryption algorithm is run on a mobile device, encryption time and battery power are of large importance. In other applications, authority servers that generate users' private keys may become a bottleneck. In both of these scenarios, *an exacerbating factor is that the cost for operations may vary widely between each ciphertext and key; thus forcing a system to provision for a load that matches a worst case scenario.* See [4, 18, 23] for further ABE performance cost details.

In this work, we aim to mitigate this problem by introducing methods for online/offline encryption and key generation in Attribute-Based Encryption. By moving the majority of the cost of an encryption and key generation into an offline phase, a system will be able to smooth the computational (and power) demand over a longer range of time, and thus only need the resources to handle the average case load.

Applications for this Technology One motivating application for splitting the work this way is that a mobile device could be programmed to automatically do ABE preparation work whenever it is plugged into a power source, and then when it is unplugged, ABE ciphertexts could be rapidly formed with a significant reduction in battery consumption.

Another potential advantage of splitting work this way is that in some applications the online and offline work can be performed in different devices. One might perform the offline work for several encryptions on a high-end server and store these intermediate ciphertexts on a sensor device such that the small device never needs to perform a full encryption. In other applications, for security reasons a designer might wish to limit the number of outward facing servers that have access to the master secret key (or equivalent). Using online/offline techniques he could have several servers performing offline operations, but relatively fewer required for the final online step to generate a user's private key. While a corrupted offline server (without the master secret) could not break the system, in collusion it could produce outputs that would allow an eventual key holder to do so. Therefore, application of this idea would require further analysis and techniques to mitigate this scenario.

Background on Online/Offline Cryptography Even, Goldreich and Micali [9] initiated online/offline techniques for signatures and Shamir and Tauman [22] introduced a general method using chameleon hash functions. In the context of signatures, one would like to perform most of the work for signing a message in the offline phase, but without knowing what the message to be signed is. Later in the online phase the signer will learn the message and given the offline work should be able to sign it relatively quickly.

The focus of our investigation is on moving encryption computation offline. In the basic encryption setting, the job is to perform most of the work for encryption offline, before the message is known. This is one of the reasons that stream ciphers, such as RC4, are sometimes preferred over certain block ciphers, because they operate by generating a pseudorandom string (which can be done offline) and then XORing it with the plaintext (in the online phase).

Let's next consider the task of moving encryption computation offline for Identity-Based Encryption (IBE), where neither the message nor the recipient's identity is known during the offline phase. Guo et al. [12] give an offline encryption system for Identity-Based Encryption (and other works [17, 16, 8, 21] proposed different variants). We illustrate the main idea as a KEM¹ variant of the Boneh-Boyen [5] IBE system. In the offline phase, one will create a ciphertext by encrypting to a random identity $x \in \mathbb{Z}_p$ with randomness $s \in \mathbb{Z}_p$. The resulting BB-type ciphertext will have the form $C_1 = g^s, C_2 = (u^x h)^s$ and the encapsulated key will be $e(g, g)^{as}$, where the bilinear group description $\mathbb{G}$ of order p and $g, u, h, e(g, g)^\alpha$ are in the public parameters. The offline algorithm will store these ciphertext components as well as remember x and s ; these together will consist of what we call an *intermediate ciphertext*. In the online phase, the encryptor will learn that she wishes to encrypt to a certain identity $\mathcal{I} \in \mathbb{Z}_p$. To do this, she simply adds a small "correction factor" $r \cdot (\mathcal{I} - x) \in \mathbb{Z}_p$ to the ciphertext components C_1, C_2 . The computation only takes one multiplication and subtraction in $\mathbb{Z}_p$. A modified decryption algorithm with the correct private key can then extract the required symmetric key. We note that treating the system as a Key Encapsulation Mechanism allows us to separate the issues of learning the identity in the online phase versus learning the message in the online phase.

The Challenge for ABE From the above description, one can see that the correction techniques critically rely on there being well-known algebraic relationships between the Boneh-Boyen hashes of different identities. Unfortunately, these do not exist in most initial ABE systems [10, 6, 24] as an attribute for string x would typically be represented as either a random group element h_x in the parameters or as the result of a (random oracle modeled) hash function $H(x)$. A second challenge is that the size and structure of ciphertext descriptors is more complex in ABE systems. For instance, in a KP-ABE system the number of attributes associated with a ciphertext may vary widely between each encryption. If one encrypts to a small number in each offline stage, the intermediate ciphertext may be not useable. If one encrypts to a large or maximum number in each offline phase, it can result in much wasted work. Using offline computation efficiently becomes a challenge in this setting. For ciphertext-policy ABE, finding a good solution is more challenging as the "unknown" is an complex access structure.

Our Contributions We develop new techniques for online/offline ABE encryption and key generation that tackle these challenges. The first non-trivial task is to identify ABE constructions that have the required algebraic structure to enable online/offline computation. Unfortunately, most existing schemes do not. However, a few do. We first identified the recent "large universe" construction of Lewko and Waters [14] as a candidate base scheme due to its algebraic structure

¹ A key encapsulation mechanism, where the public key ciphertext encapsulates a symmetric key which could later be used to symmetrically encrypt the plaintext.

that appears amenable to adding correction factors.² We finally decided to use a recent more efficient prime-order variant due to Rouselakis and Waters [19]. (We are not aware of any other ABE schemes that can support a similarly efficient online/offline tradeoff.)

We begin by designing online/offline encryption algorithms for Key-Policy ABE. For our first construction we assume a set number of attributes that will be associated with each ciphertext. In this setting we develop a correction technique for the KP-ABE [19] system. We prove security by directly reducing to the security of [19]. This has the advantage of simplicity in that we do not need to revisit the guts of the prior proof. In addition, we will automatically inherit any future improvements in the proof for the underlying scheme.

For reasons, discussed above assuming a fixed number of attributes per ciphertext is undesirable. To this end we come up with a method of “pooling” work done offline. In this system an encryptor will continuously create offline ciphertext pieces and add these to a pool. When the encryption algorithm later needs to encrypt to a set S of attributes, it grabs $|S|$ pieces from the pool connecting each one to a single attribute from S . The work per attribute is dominated by one multiplication in $\mathbb{Z}_p$. We describe this as a “connect and correct” approach.

We extend our offline encryption approach to the more complex case of Ciphertext-Policy ABE. The challenge here is that a CP-ABE ciphertext is associated with a Linear Secret Sharing Scheme (LSSS) matrix. Again, we develop a pooling technique. However, in this application for each row of the matrix M given online, we will need to correct each ciphertext component to an LSSS share in the exponent and to the corresponding attribute. Finally, we show how online/offline key generation can be derived from our encryption techniques. We observe a symmetry between CP-ABE encryption and KP-ABE key generation that allows us to develop an online/offline pair of algorithms for the latter.

Combining with Outsourcing for ABE We make a brief detour here to discuss how the results of this work might be combined with prior ABE results to make a practical overall system.

In 2011, Green, Hohenberger and Waters [11] presented a solution for outsourcing the decryption of ABE ciphertexts. That is, they assumed that ABE ciphertexts might be stored in the cloud. They then showed how a user can provide the cloud with a *single* translation key that allows the cloud to translate *any* ABE ciphertext satisfied by that user’s attributes into a very short El Gamal-style ciphertext, without the cloud being able to read any part of the user’s messages. These transmitted ciphertexts are short (saving on bandwidth and receiving time), but also quick to decrypt (with roughly one or two exponentiations). Thus, the ability to outsource decryption to the cloud allows a mobile device to quickly decrypt an ABE-encrypted message.

² Interestingly, [14] aimed for a large universe construction in the standard model and thus our use of the schemes’s additional structure is a byproduct of removing the random oracles.

Conversely, the results of this work allow a mobile device to quickly *encrypt* an ABE-encrypted message. These two results could be combined into one system, where a mobile device would be fully ABE operational while drastically reducing the computational costs for both decryption (with the help of the cloud) and encryption (with the help of a preparation phase while the phone charges). We believe that creative solutions of this sort can be implemented transparently, but will provide noticeably better performance for users.

2 Definitions for Online/Offline ABE

We work in the key encapsulation mechanism (KEM) setting, where the attribute-based ciphertext hides a symmetric session key that can then be used to symmetrically encrypt data of arbitrary length. The goal in the online/offline setting is to allow as much precomputation of attribute-based ciphertext as possible *without* knowing the intended access policy (ciphertext-policy) or set of attributes (key-policy). We refer the reader to [13] for a review of access structures, linear secret sharing schemes (LSSS) and related conventions.

Definition 1 (Online/Offline Attribute-Based KEM Specification). *Let S represent a set of attributes and $\mathbb{A}$ an access structure. For generality, we will define (I_{key}, I_{enc}) as the inputs to the extract and online encryption functions respectively. In a KP-ABE scheme $(I_{key}, I_{enc}) := (\mathbb{A}, S)$, while in a CP-ABE scheme, we have $(I_{key}, I_{enc}) := (S, \mathbb{A})$. We define the function f as follows:*

$$f(I_{key}, I_{enc}) := \begin{cases} 1 & \text{if } I_{enc} \in I_{key} \text{ in KP-AB setting} \\ 1 & \text{if } I_{key} \in I_{enc} \text{ in CP-AB setting} \\ 0 & \text{otherwise.} \end{cases}$$

An online/offline KP-AB (resp., CP-AB) key-encapsulation mechanism for access structure space $\mathcal{G}$ is a tuple of the following algorithms:

Setup $(\lambda, U) \rightarrow (\text{PK}, \text{MK})$. *The setup algorithm takes as input a security parameter λ and a universe description U , which defines the set of allowed attributes in the system. It outputs the public parameters PK and the master secret key MK.*

Extract $(\text{MK}, I_{key}) \rightarrow \text{SK}$. *The extract algorithm takes as input the master secret key MK and an access structure (resp., set of attributes) I_{key} and outputs a private key SK associated with the attributes.*

Offline.Encrypt $(\text{PK}) \rightarrow \text{IT}$. *The offline encryption algorithm takes as input the public parameters PK and outputs an intermediate ciphertext IT.*

Online.Encrypt $(\text{PK}, \text{IT}, I_{enc}) \rightarrow (\text{key}, \text{CT})$ *The online encryption algorithm takes as input the public parameters PK, an intermediate ciphertext IT and a set of attributes (resp., access structure) I_{enc} and outputs a session key key and a ciphertext CT.*

Decrypt(SK, CT) $\rightarrow$ key. The decryption algorithm takes as input a private key SK for I_{key} and a ciphertext CT associated with I_{enc} and decapsulates ciphertext CT to recover a session key key if S satisfies $\mathbb{A}$ or the error message $\perp$ otherwise.

For a fixed universe description U and $\lambda \in \mathbb{N}$, the KP-AB correctness property requires that for all $(PK, MK) \in \text{Setup}(\lambda, U)$, all $S \subseteq U$, all $\mathbb{A} \in \mathcal{G}$, all $SK \in \text{Extract}(MK, \mathbb{A})$, if $(key, CT) \in \text{Online.Encrypt}(PK, \text{Offline.Encrypt}(PK), S)$ and if S satisfies $\mathbb{A}$, then **Decrypt**(SK, CT) outputs key. CP-AB correctness is defined analogously, with the last inputs to **Extract** and **Online.Encrypt** reversed.

Security Model for Online/Offline AB-KEM Let $\Pi = (\text{Setup}, \text{Extract}, \text{Offline.Encrypt}, \text{Online.Encrypt}, \text{Decrypt})$ be an AB-KEM for access structure space $\mathcal{G}$, and consider the following experiment for an adversary $\mathcal{A}$, parameter λ and attribute universe U :

The Online/Offline AB-KEM experiment $\text{OO-ABKEM-Exp}_{\mathcal{A}, \Pi}(\lambda, U)$:

Setup. The challenger runs the Setup algorithm and gives the public parameters, PK to the adversary.

Phase 1. The challenger initializes an empty table T , an empty set D and an integer counter $j = 0$. Proceeding adaptively, the adversary can repeatedly make any of the following queries:

- **Create**(I_{key}): The challenger sets $j := j + 1$. It runs the key generation algorithm on I_{key} to obtain the private key SK and stores in table T the entry (j, I_{key}, SK) .
Note: Create can be repeatedly queried with the same input.
- **Corrupt**(i): If there exists an i^{th} entry in table T , then the challenger obtains the entry (i, I_{key}, SK) and sets $D := D \cup \{I_{key}\}$. It then returns to the adversary the private key SK. If no such entry exists, then it returns $\perp$.
- **Decrypt**(i, CT): If there exists an i^{th} entry in table T , then the challenger obtains the entry (i, I_{key}, SK) and returns to the adversary the output of the decryption algorithm on input (SK, CT) . If no such entry exists, then it returns $\perp$.

Challenge. The adversary gives a challenge value I_{enc}^* such that for all $I_{key} \in D$, $f(I_{key}, I_{enc}^*) \neq 1$. The challenger runs the algorithm **Online.Encrypt**(PK, **Offline.Encrypt**(PK), I_{enc}^*) to obtain (key^*, CT^*) . It then randomly selects a bit b . If $b = 0$, it returns (key^*, CT^*) to the adversary. If $b = 1$, it selects a random session key R in the session key space and returns (R, CT^*) .

Phase 2. Phase 1 is repeated with the restrictions that the adversary cannot

- trivially obtain a private key for the challenge ciphertext. That is, it cannot issue a Corrupt query that would result in a value I_{key} which satisfies $f(I_{key}, I_{enc}^*) = 1$ being added to D .
- issue a decryption query on the challenge ciphertext CT^* .

Guess. The adversary outputs a guess b' of b . The output of the experiment is 1 if and only if $b = b'$.

Definition 2 (Online/Offline AB-KEM Security). *An online/offline AB-KEM Π is CCA-secure (or secure against chosen-ciphertext attacks) for attribute universe U if for all probabilistic polynomial-time adversaries $\mathcal{A}$, there exists a negligible function negl such that:*

$$\Pr[\text{OO-ABKEM-Exp}_{\mathcal{A},\Pi}(\lambda, U) = 1] \leq \frac{1}{2} + \text{negl}(\lambda).$$

CPA Security. We say that a system is CPA-secure (or secure against chosen-plaintext attacks) if we remove the Decrypt oracle in both Phase 1 and 2.

Selective Security. We say that a system is *selectively* secure if we add an Init stage before Start where the adversary outputs the challenge I_{enc}^* (instead of waiting until Challenge).

3 A KP-ABE Scheme with Online/Offline Encryption

We now show how to extend the unbounded KP-ABE scheme of Rouselakis and Waters [19, Appendix C] to be an online/offline system. We will work in a key encapsulation mechanism (KEM) model as specified in Definition 2, so that we can focus on preparing for an unknown attribute set. Any plaintext can be encrypted in a hybrid manner during the online phase by a symmetric cipher keyed with the encapsulated key. We first show a simple system that assumes a bound P on the maximum number of attributes that can be used to encrypt a ciphertext. We show how to remove this bound in Section 3.2.

Setup(λ, U) The setup algorithm takes in a security parameter λ and a universe U of attributes. chooses a bilinear group $\mathbb{G}$ of prime order $p \in \Theta(2^\lambda)$. It also chooses random generators $g, h, u, w \in \mathbb{G}$ and picks a random exponent $\alpha \in \mathbb{Z}_p$. It then sets the keys as:

$$\text{PK} = (\mathbb{G}, p, g, h, u, w, e(g, g)^\alpha), \quad \text{MSK} = (\text{PK}, \alpha).$$

We assume that the universe of attributes can be encoded as elements in $\mathbb{Z}_p$.

Extract($\text{MSK}, (M, \rho)$) The extract algorithm takes as input the master secret key MSK and an LSSS access structure (M, ρ) . Let M be an $\ell \times n$ matrix. The function ρ associates rows of M to attributes. The algorithm initially chooses random values $y_2, \dots, y_n \in \mathbb{Z}_p$. It then computes ℓ shares of the master secret key as $(\lambda_1, \lambda_2, \dots, \lambda_\ell) := M \cdot (\alpha, y_2, \dots, y_n)^T$ (where T denotes the transpose). It then picks ℓ random exponents $t_1, t_2, \dots, t_\ell \in \mathbb{Z}_p$. For $i = 1$ to ℓ , it computes

$$K_{i,0} := g^{\lambda_i} w^{t_i} \quad K_{i,1} := (u^{\rho(i)} h)^{-t_i} \quad K_{i,2} := g^{t_i}.$$

The private key is $\text{SK} := ((M, \rho), \{K_{i,0}, K_{i,1}, K_{i,2}\}_{i \in [1, \ell]})$.

Offline.Encrypt(PK) The offline encryption algorithm takes in the public parameters only. Here we describe the basic system which assumes a maximum bound of P attributes will be associated with any ciphertext. We describe more advanced variations in Section 3.2. The algorithm first picks a random $s \in \mathbb{Z}_p$ and computes

$$\text{key} := e(g, g)^{\alpha s} \quad C_0 := g^s.$$

Next, for $j = 1$ to P , it chooses random $r_j, x_j \in \mathbb{Z}_p$ and computes

$$C_{j,1} := g^{r_j} \quad C_{j,2} := (u^{x_j} h)^{r_j} w^{-s}.$$

One can view this as encrypting for a random attribute x_j , where this will be corrected in the online phase. The work done in the offline phase is roughly equivalent to the work of the regular encryption algorithm in [19, Appendix C].

The intermediate ciphertext is $\text{IT} := (\text{key}, C_0, \{r_j, x_j, C_{j,1}, C_{j,2}\}_{j \in [1, P]})$.

Online.Encrypt(PK, IT, S) The online encryption KEM algorithm takes as input the public parameters, an intermediate ciphertext IT, and a set of attributes $S = (A_1, A_2, \dots, A_{k \leq P})$. For $j = 1$ to k , it computes $C_{j,3} := (r_j \cdot (A_j - x_j)) \bmod p$. Intuitively, this will correct to the proper attributes. It sets the ciphertext:

$$\text{CT} := (S, C_0, \{C_{j,1}, C_{j,2}, C_{j,3}\}_{j \in [1, k]}).$$

The encapsulated key is key . The dominant cost is one multiplication in $\mathbb{Z}_p$ per attribute in S .

Decrypt(SK, CT) The decryption algorithm in the KEM setting recovers the encapsulated key. It takes as input a ciphertext $\text{CT} = (S, C_0, \{C_{j,1}, C_{j,2}, C_{j,3}\}_{j \in [1, k]})$ for attribute set S and a private key $\text{SK} = ((M, \rho), \{K_{i,0}, K_{i,1}, K_{i,2}\}_{i \in [1, \ell]})$ for access structure (M, ρ) . If S does not satisfy this access structure, then the algorithm issues an error message. Otherwise, it sets $I := \{i : \rho(i) \in S\}$ and computes constants $w_i \in \mathbb{Z}_p$ such that $\sum_{i \in I} w_i \cdot M_i = (1, 0, \dots, 0)$, where M_i is the i -th row of the matrix M . Then it then recovers the encapsulated key by calculating $\text{key} :=$

$$\prod_{i \in I} (e(C_0, K_{i,0}) \cdot e(C_{j,1}, K_{i,1}) \cdot e(C_{j,2} \cdot u^{C_{j,3}}, K_{i,2}))^{w_i} = e(g, g)^{\alpha s} \quad (1)$$

where j is the index of the attribute $\rho(i)$ in S (it depends on i). This does not increase the number of pairing operations over [19, Appendix C], although it adds $|I|$ exponentiations.

Correctness If the attribute set S of the ciphertext is authorized, we have that $\sum_{i \in I} w_i \lambda_i = \alpha$. Therefore, **key**:

$$\begin{aligned}
&:= \prod_{i \in I} (e(C_0, K_{i,0}) \cdot e(C_{j,1}, K_{i,1}) \cdot e(C_{j,2} \cdot u^{C_{j,3}}, K_{i,2}))^{w_i} \\
&= \prod_{i \in I} (e(g^s, g^{\lambda_i} w^{t_i}) \cdot e(g^{r_j}, (u^{\rho(i)} h)^{-t_i}) \cdot e((u^{x_j} h)^{r_j} w^{-s} \cdot u^{r_j(\rho(i)-x_j)}, g^{t_i}))^{w_i} \\
&= \prod_{i \in I} (e(g, g)^{s\lambda_i} \cdot e(g, w)^{st_i} \cdot e(g, u)^{-r_j t_i \rho(i)}) \\
&\quad e(g, h)^{-r_j t_i} \cdot e(g, u)^{\rho(i)r_j t_i} \cdot e(g, h)^{r_j t_i} \cdot e(g, w)^{-st_i})^{w_i} \\
&= \prod_{i \in I} e(g, g)^{sw_i \lambda_i} = e(g, g)^{s\alpha}.
\end{aligned}$$

Recall that in the symmetric setting $e(g, u) = e(u, g)$, for all $g, u \in \mathbb{G}$, although this scheme can operate in an asymmetric setting with small alterations.

3.1 Proof of Selective Security

Discussion on Security. We shortly show that the security of our online/offline system can be directly based on the security of the underlying Rouselakis-Waters [19, Appendix C] system. The Rouselakis-Waters system that we reduce security to is selectively secure based on a “q-type” assumption in prime order groups. We remark that our techniques appear to be equally amenable to transforming the Lewko-Waters [15] system to an online/offline system. The Lewko-Waters system is proven selectively secure from a static assumption in composite order groups. If such a transformation were done (as well as a reduction to their scheme), the new scheme would inherit those assumptions.

In [10, Section 9], Goyal et al. discuss how to combine delegation in their ABE systems with the techniques of Canetti-Halevi-Katz [7] to build a CCA secure ABE scheme from a CPA one. We believe that a similar delegation structure exists in our schemes, so that similar techniques would likely work out (although we do not work out the details here).

Theorem 1. *The above online/offline KP-AB-KEM scheme is selectively CPA-secure with respect to Definition 2 assuming that the scheme of Rouselakis and Waters [19, Appendix C] is a selectively CPA-secure KP-ABE system.*

Proof. To prove the theorem, we will show that any PPT attacker $\mathcal{A}$ with a non-negligible advantage in the OO-ABKEM-Exp experiment against the above scheme, which we will denote $\Pi_{OO} = (\text{Setup}, \text{Extract}, \text{Offline.Encrypt}, \text{Online.Encrypt}, \text{Decrypt})$, can be used to break the selective CPA-security of the Rouselakis-Waters scheme, which we will denote $\Pi_{RW} = (\text{Setup}_{RW}, \text{Extract}_{RW}, \text{Encrypt}_{RW}, \text{Decrypt}_{RW})$, with a PPT simulator $\mathcal{B}$.

The simulator plays the challenger and interacts with $\mathcal{A}$ in OO-ABKEM-Exp with security parameter λ and the universe of attributes set to $U = \mathbb{Z}_p$.

Initialization Initially, $\mathcal{B}$ receives an attribute set $S^* = \{A_1^*, A_2^*, \dots, A_k^*\} \subseteq U$ from $\mathcal{A}$ and gives it to the RW challenger.

Setup Next, $\mathcal{B}$ receives the public parameters $\text{PK} = (\mathbb{G}, p, g, h, u, w, e(g, g)^\alpha)$ from the RW challenger and passes them to $\mathcal{A}$ unchanged.

Phase 1 The secret keys are the same in both schemes, so any key generation request from $\mathcal{A}$ is passed to the RW challenger to obtain the key.

Challenge $\mathcal{B}$ chooses two distinct, random messages m_0, m_1 in the RW message space and sends them to its RW challenger, and receives back a challenge ciphertext $\text{CT}_{RW}^* = (S^*, C, C_0, \{C_{j,1}, C_{j,2}\}_{j \in [1, |S^*|]})$. Here C is the encrypted message times $e(g, g)^{\alpha s}$, $C_0 = g^s$ and for each attribute $A_j \in S^*$, we have $C_{j,1} = g^{r_j}$ and $C_{j,2} = (u^{A_j} h)^{r_j} w^{-s}$.

It then selects random values $z_1, \dots, z_{|S^*|} \in \mathbb{Z}_p$ and computes the ciphertext CT_{OO}^* as (S^*, C_0) followed by

$$C_{j,1}^* := C_{j,1} = g^{r_j} \quad C_{j,2}^* := C_{j,2} \cdot u^{-z_j} = (u^{A_j} h)^{r_j} w^{-s} u^{-z_j} \quad C_{j,3}^* := z_j.$$

To see why this is a correctly formed ciphertext, one needs to recall the third pairing of equation 1, where one must compute $e(C_{j,2}^* \cdot u^{C_{j,3}^*}, K_{i,2})$, as well as observe that the ciphertext is randomized to have the proper distribution. The z_j blinding will cancel out in this step. Next, $\mathcal{B}$ guess which message was encrypted $\tau_B \in \{0, 1\}$ and computes $\text{key}_{\text{guess}} := C/m_{\tau_B}$. Finally, $\mathcal{B}$ then sends to $\mathcal{A}$ the tuple $(\text{key}_{\text{guess}}, \text{CT}_{OO}^*)$.

Phase 2 $\mathcal{B}$ proceeds as in Phase 1.

Guess Eventually, $\mathcal{A}$ outputs a bit τ_A . If $\tau_A = 0$ (meaning that $\mathcal{A}$ guesses that $\text{key}_{\text{guess}}$ is the key encapsulated by CT_{OO}^*), then $\mathcal{B}$ outputs τ_B . If $\tau_A = 1$ (meaning that $\mathcal{A}$ guesses that $\text{key}_{\text{guess}}$ is a random key), then $\mathcal{B}$ outputs $1 - \tau_B$. The distribution for $\mathcal{A}$ is perfect. Thus, if $\mathcal{A}$ has advantage ϵ in the OO-ABKEM-Exp experiment, then $\mathcal{B}$ breaks the RW KP-ABE system with the same probability.

3.2 A More Advanced System: Pooling Attributes for an Unbounded System

Previously, we presented a system that imposed a bound of P attributes associated with any ciphertext. We presented P as if it was a system-wide bound for all ciphertexts, for simplicity. A slightly less naive solution would involve creating a set of intermediate ciphertexts prepared for different sizes of attribute sets, and then pulling the “right-sized IT” off-the-shelf during the online phase (e.g., create one IT for a set of size 1, another for a set of size 2, etc.). However, these approaches could prove wasteful, as certain ITs may be created and stored without being used.

Pooling Construction. Instead, we introduce the idea of “pooling” to eliminate waste during the offline phase. The intermediate ciphertext is now comprised of two logical types of objects: a main module and an attribute module. During the offline phase(s), an arbitrary number of main and attribute modules are independently created. During the online phase for attribute set S , one main module and $|S|$ attribute modules will be consumed. The critical feature of this approach is that any attribute module can be attached to any main module. The online phase uses exactly what it needs, and any modules left in the pool can be used on subsequent ciphertexts.

Specifically, during **Offline.Encrypt**, a main module is computed as follows. It picks a random $s \in \mathbb{Z}_p$ and sets $\text{IT}_{\text{main}} := (\text{key}, C_0, C_w)$, where these values are computed as

$$\text{key} := e(g, g)^{\alpha s} \quad C_0 := g^s \quad C_w := w^{-s}.$$

During **Offline.Encrypt**, an attribute module is computed as follows. It picks a random $r, x \in \mathbb{Z}_p$ and sets $\text{IT}_{\text{att}} := (r, x, C'_1, C'_2)$, where these values are computed as

$$C'_1 := g^r \quad C'_2 := (u^x h)^r.$$

During **Online.Encrypt** for an attribute set S , the algorithm selects any one main module $\text{IT}_{\text{main}} := (\text{key}, C_0, C_w)$ and any $|S|$ attribute modules $\text{IT}_{\text{att},j} := (r_j, x_j, C'_{j,1}, C'_{j,2})$ available in the pool. Finally, it computes CT as $(S, C_0, \{C_{j,1}, C_{j,2}, C_{j,3}\}_{j \in [1, |S|]})$, where

$$C_{j,1} := C'_{j,1} = g^{r_j} \quad C_{j,2} := C'_{j,2} \cdot C_w = (u^{x_j} h)^{r_j} \cdot w^{-s} \quad C_{j,3} := r_j \cdot (A_j - x_j).$$

The encapsulated key is **key**.

Security Discussion. The dominant cost in the online encryption algorithm is 2 modular multiplications per attribute in S . To formally capture the pooling model, the specification and security definition in Section 2 would need to be expanded to have the **Offline.Encrypt** algorithm keep state (e.g., the pool) between iterations and to pass this state into **Online.Encrypt** as well. Since pooling does not impact the structure or distribution of the final ciphertexts over Section 3 and the adversary in the security experiment only views final ciphertexts, it is relatively straightforward to prove the selective security of the pooling scheme.

4 A CP-ABE Scheme with Online/Offline Encryption

We now turn our attention to developing online/offline CP-ABE systems. This is intuitively harder than KP-ABE, because the structure of ciphertext is more complex. We must now be able to create an intermediate ciphertext in the offline phase that can be quickly be translated to a ciphertext for a hitherto unknown access structure. To do this, we will use and extend the basic “correction” and

pooling concepts introduced for KP-ABE. Our online/offline system is based on the unbounded CP-ABE scheme of Rouselakis and Waters [19, Section 4], where again it takes a special algebraic structure to make this work, which most other CP-ABE systems do not appear to have. As before, we are working in the KEM model. We'll first show a simple system that assumes a bound P on the maximum number of rows in an LSSS access structure that will be used to encrypt. We will subsequently discuss how to remove this bound.

Setup(λ, U) The setup algorithm chooses a bilinear group $\mathbb{G}$ of prime order $p \in \Theta(2^\lambda)$. It also chooses random generators $g, h, u, v, w \in \mathbb{G}$ and picks a random exponent $\alpha \in \mathbb{Z}_p$. It then sets the keys as:

$$\text{PK} = (\mathbb{G}, p, g, h, u, v, w, e(g, g)^\alpha), \quad \text{MSK} = (\text{PK}, \alpha).$$

Again, we will view the attribute universe as consisting of elements in $\mathbb{Z}_p$.

Extract(MSK, S) The extract algorithm takes as input the master secret key MSK and an attribute set $S = \{A_1, A_2, \dots, A_k\} \subseteq \mathbb{Z}_p$. The algorithm chooses random values $r, r_1, r_2, \dots, r_k \in \mathbb{Z}_p$. It then computes $K_0 := g^\alpha w^r$, $K_1 := g^r$, and for $i = 1$ to k , it computes

$$K_{i,2} := g^{r_i} \quad K_{i,3} := (u^{A_i} h)^{r_i} v^{-r}.$$

The private key is $\text{SK} := (S, K_0, K_1, \{K_{i,2}, K_{i,3}\}_{i \in [1,k]})$.

Offline.Encrypt(PK) The offline encryption algorithm takes in the public parameters only. Here we describe the basic system which assumes a maximum bound of P rows in any LSSS access structure used in a ciphertext. We describe more advanced variations in Section 4.1. The algorithm first picks a random $s \in \mathbb{Z}_p$ and computes

$$\text{key} := e(g, g)^{\alpha s} \quad C_0 := g^s.$$

Next, for $j = 1$ to P , it chooses random $\lambda'_j, x_j, t_j \in \mathbb{Z}_p$ and computes

$$C_{j,1} := w^{\lambda'_j} v^{t_j} \quad C_{j,2} := (u^{x_j} h)^{-t_j} \quad C_{j,3} := g^{t_j}.$$

One can view this as encrypting for a random attribute x_j with a random “share” λ'_j of s , where this will be corrected in the online phase. We remark that the work done in the offline phase is roughly equivalent to the work of the regular encryption algorithm in [19, Section 4].

Intermediate ciphertext is $\text{IT} := (\text{key}, s, C_0, \{\lambda'_j, t_j, x_j, C_{j,1}, C_{j,2}, C_{j,3}\}_{j \in [1,P]})$.

Online.Encrypt(PK, IT, (M, ρ)) The online encryption KEM algorithm takes as input the public parameters, an intermediate ciphertext IT, and an LSSS access structure (M, ρ) , where M is an $\ell \times n$ matrix and $\ell \leq P$. It picks random $y_2, \dots, y_n \in \mathbb{Z}_p$, sets the vector $\mathbf{y} = (s, y_2, \dots, y_n)^T$ (where T denotes the transpose of the matrix) and computes a vector of shares of s as $(\lambda_1, \dots, \lambda_\ell)^T = M\mathbf{y}$.

For $j = 1$ to ℓ , it computes

$$C_{j,4} := \lambda_j - \lambda'_j \quad C_{j,5} := t_j \cdot (\rho(j) - x_j).$$

Intuitively, this will correct to the proper attributes and shares of s . It sets the ciphertext as:

$$\text{CT} := ((M, \rho), C_0, \{C_{j,1}, C_{j,2}, C_{j,3}, C_{j,4}, C_{j,5}\}_{j \in [1, k]}).$$

The encapsulated key is key . The dominant cost is one multiplication in $\mathbb{Z}_p$ per row of M .

Decrypt(SK, CT) The decryption algorithm in the KEM setting recovers the encapsulated key. It takes as input a ciphertext $\text{CT} = ((M, \rho), C_0, \{C_{j,1}, C_{j,2}, C_{j,3}, C_{j,4}, C_{j,5}\}_{j \in [1, k]})$ for access structure (M, ρ) and a private key $\text{SK} = (S, \{K_{i,0}, K_{i,1}, K_{i,2}\}_{i \in [1, \ell]})$ for access structure (M, ρ) . If S does not satisfy this access structure, then the algorithm issues an error message. Otherwise, it sets $I := \{i : \rho(i) \in S\}$ and computes constants $w_i \in \mathbb{Z}_p$ such that $\sum_{i \in I} w_i \cdot M_i = (1, 0, \dots, 0)$, where M_i is the i -th row of the matrix M . Then it then recovers the encapsulated key by calculating $\text{key} := e(g, g)^{\alpha s} =$

$$\frac{e(C_0, K_0)}{e(w^{\sum_{i \in I} C_{i,4} w_i}, K_1) \cdot \prod_{i \in I} (e(C_{i,1}, K_1))} \cdot \frac{1}{e(C_{i,2} \cdot u^{C_{i,5}}, K_{j,2}) \cdot e(C_{i,3}, K_{j,3}))^{w_i}} \quad (2)$$

where j is the index of the attribute $\rho(i)$ in S (it depends on i). We note that this decryption algorithm adds one pairing operation and $|I| + 1$ exponentiations over [19, Appendix C]. Alternatively, one could re-arrange the equation for no additional pairings at the cost of $2|I|$ exponentiations.

In the full version [13], we show correctness and prove the below theorem.

Theorem 2. *The above online/offline CP-AB-KEM scheme is selectively CPA-secure with respect to Definition 2 assuming that the scheme of Rouselakis and Waters [19, Section 4] is a selectively CPA-secure CP-ABE system.*

4.1 Pooling Attributes for an Unbounded Ciphertext-Policy System

In the previous section, we presented an online/offline system that imposed a bound of P rows on any LSSS access matrix associated with any ciphertext. As introduced in Section 3.2, we now show how to remove this bound by creating a “pool” from which to draw ready-made ciphertext components. As before, the intermediate ciphertext is comprised of two logical types of objects: a main module and an attribute module. During the offline phase(s), an arbitrary number of main and attribute modules are independently created. During the online phase for LSSS access structure (M, ρ) , one main module and ℓ attribute modules will be consumed, where M is an $\ell \times n$ matrix. Any attribute module can be attached to any main module.

Specifically, during **Offline.Encrypt**, a main module is computed as follows. It picks a random $s \in \mathbb{Z}_p$ and sets $\text{IT}_{main} := (\text{key}, C_0)$, where these values are computed as

$$\text{key} := e(g, g)^{\alpha s} \quad C_0 := g^s.$$

During **Offline.Encrypt**, an attribute module is computed as follows. It picks a random $\lambda, x, t \in \mathbb{Z}_p$ and sets $\text{IT}_{att} := (\lambda, x, t, C_1, C_2, C_3)$, where these values are computed as

$$C_1 := w^\lambda v^t \quad C_2 := (u^x h)^t \quad C_3 := g^t.$$

During **Online.Encrypt** for an LSSS access structure (M, ρ) , where M is an $\ell \times n$ matrix, the algorithm selects any one main module $\text{IT}_{main} := (\text{key}, C_0)$ and any ℓ attribute modules $\text{IT}_{att,j} := (\lambda_j, x_j, t_j, C_{j,1}, C_{j,2}, C_{j,3})$ available in the pool. It picks random $y_2, \dots, y_n \in \mathbb{Z}_p$, sets the vector $\mathbf{y} = (s, y_2, \dots, y_n)^T$ (where T denotes the transpose of the matrix) and computes a vector of shares of s as $(\lambda_1, \dots, \lambda_\ell)^T = M\mathbf{y}$.

Finally, it computes CT as $((M, \rho), C_0, \{C_{j,1}, C_{j,2}, C_{j,3}, C_{j,4}, C_{j,5}\}_{j \in [1, \ell]})$, where

$$C_{j,4} := \lambda_j - \lambda'_j \quad C_{j,5} := t_j \cdot (\rho(j) - x_j).$$

The encapsulated key is **key**. The dominant cost in the online encryption algorithm is one modular multiplication per row in M . The security discussion at the end of Section 3.2 applies here as well.

5 Online/Offline ABE Key Generation

Private key generation in ABE systems requires the master secret key MSK. This key is so valuable that any organization granting keys might do well to store it on only a small number of well-guarded servers. At the same time, this could create a bottleneck in systems with many users, especially when private keys are reissued each time period for revocation purposes. In this section, we discuss how the key generation operation in the KP-ABE system of Section 3 and the CP-ABE system of Section 4 can operate in an online/offline fashion as well. Thus, the bulk of the key generation work can be performed by servers that are truly *offline* (or otherwise well secured). These pre-computations can be passed to the online servers, where incoming requests can be processed quickly.

In the KP-ABE setting, a private key embeds an LSSS access structure, whereas in the CP-ABE setting, the private key embeds a set of attributes. We will borrow ideas from the prior two sections to deal with these objects, where again we can employ both the “correct and connect” and “pooling” concepts.

To capture online/offline key generation, one needs to replace the **Extract** algorithm with an offline algorithm that takes in the MK and produces a intermediate private key (or pool of private key parts) and an online algorithm that takes in this intermediate key (or pool) together with an access structure and then produces the private key. The security experiment is essentially unchanged except that the Create oracle (called in Phases 1 and 2) now calls **Offline.Extract** and **Online.Extract** in sequence to create a private key.

5.1 Online/Offline Key Generation for KP-ABE Keys

The **Setup** and encryption algorithms remain the same as Section 3. We present a pooling solution, and because the structure of the private keys change, so must the decryption algorithm.

Offline.Extract(MSK) There are no “main” key modules. A “row” module is computed by selecting random $\lambda', x, t \in \mathbb{Z}_p$ and outputting $I_{row} := (\lambda', x, t, K_0, K_1, K_2)$ where $K_0 := g^{\lambda'} w^t$, $K_1 := (u^x h)^{-t}$ and $K_2 := g^t$.

Online.Extract(pool, (M, ρ)) Let M be an $\ell \times n$ matrix. The algorithm initially chooses random values $y_2, \dots, y_n \in \mathbb{Z}_p$. It then computes ℓ shares of the master secret key as $(\lambda_1, \lambda_2, \dots, \lambda_\ell) := M \cdot (\alpha, y_2, \dots, y_n)$. Next select any ℓ row modules from the pool. For $i = 1$ to ℓ , set $K_{i,3} := \lambda_i - \lambda'_i$ and $K_{i,4} := t_i \cdot (\rho(i) - x_i)$. The private key is $SK := ((M, \rho), \{K_{i,0}, K_{i,1}, K_{i,2}, K_{i,3}, K_{i,4}\}_{i \in [1, \ell]})$. The dominant cost is one multiplication per row of M .

Decrypt(SK, CT) Using the prior steps and notation, it recovers the encapsulated key $:= \prod_{i \in I} (e(C_0, K_{i,0} \cdot g^{K_{i,3}}) \cdot e(C_{j,1}, K_{i,1} \cdot u^{K_{i,4}}) \cdot e(C_{j,2} \cdot u^{C_{j,3}}, K_{i,2}))^{w_i} = e(g, g)^{\alpha s}$. This adds $2|I|$ exponentiations over the construction in Section 3.

5.2 Online/Offline Key Generation for CP-ABE Keys

The CP-ABE system in Section 4 can be extended in a similar manner. In that system, there will be a “main” key module which contains K_0, K_1 and $K_v := v^{-r}$. The attribute modules are identical to those of Section 3.2 and the keys are assembled as in the online phase of 3.2. The decryption equation is then $key := e(C_0, K_0)/D$, where $D = e(w^{\sum_{i \in I} C_{i,4} w_i}, K_1) \cdot \prod_{i \in I} (e(C_{i,1}, K_1) \cdot e(C_{i,2} \cdot u^{C_{i,5}}, K_{j,2} \cdot u^{K_{j,4}}) \cdot e(C_{i,3}, K_{j,3}))^{w_i}$, resulting in $e(g, g)^{\alpha s}$.

6 Performance Analysis

We provide estimates on the performance of the proposed schemes in Figures 1 and 2. These numbers are extrapolated from operation times on a 256-bit Bareto-Naehrig curve using version 0.3.1 of the RELIC library [3]. Times are measured in milliseconds (averaged over 10,000 iterations) and were computed on an Intel Core i7 processor with 16GB RAM [2]. We ignore small numbers of operations which will be negligible by comparison, such as arithmetic in $\mathbb{Z}_p$.

A natural question to ask is: how much pre-processing can I do for an ABE encryption (similarly, key generation) before I know the message I want to encrypt or the access structure that I want to encrypt under? It may come as a surprise that the results are so drastic. Indeed, our estimates show that the answer to this question is: you can do *almost all* of the encryption work, before you know any of the specifics of what/to whom you are encrypting.

Indeed, our *worst-case* for encryption was key-policy ABE in pooling mode, and even then over 99% of the work could be done offline. Similarly, the worst-case for key generation was ciphertext-policy ABE in pooling mode, and even

Encryption Algorithm	Bilinear Operations	Est. Time $P = 10$	Est. Time $P = 100$
KP-ABE from [19, App. C]	$1\mathbb{E}_T + (3P + 2)\mathbb{E}_1 + 2P\mathbb{M}_1$	.133	1.134
KP-Offline Sec. 3	$1\mathbb{E}_T + (3P + 2)\mathbb{E}_1 + 2P\mathbb{M}_1$	.133	1.134
KP-Online Sec. 3	0	< .001	< .001
KP-Pool-Offline Sec. 3.2	$1\mathbb{E}_T + (3P + 2)\mathbb{E}_1 + P\mathbb{M}_1$	.133	1.132
KP-Pool-Online Sec. 3.2	$P\mathbb{M}_1$	< .001	.001
CP-ABE from [19]	$1\mathbb{E}_T + (5P + 1)\mathbb{E}_1 + 2P\mathbb{M}_1$	.203	1.870
CP-Offline Sec. 4	$1\mathbb{E}_T + (5P + 1)\mathbb{E}_1 + 2P\mathbb{M}_1$	.203	1.870
CP-Online Sec. 4	0	< .001	< .001
CP-Pool-Offline Sec. 4.1	$1\mathbb{E}_T + (5P + 1)\mathbb{E}_1 + 2P\mathbb{M}_1$	.203	1.870
CP-Pool-Online Sec. 4.1	0	< .001	.001

Fig. 1. Performance estimates for regular and online/offline encryption algorithms. We mapped these algorithms into the asymmetric bilinear setting, placing the ciphertexts in $\mathbb{G}_1$ and keys in $\mathbb{G}_2$. Let $\mathbb{E}_i$ (resp., $\mathbb{M}_i$) denote an exponentiation (reps., multiplication) in the group $\mathbb{G}_i$. The bilinear operations are the dominate cost, so we ignore minor factors such as arithmetic in $\mathbb{Z}_p$. The variable P represents the size of the attribute list (in KP-ABE) or the complexity of the access policy (in CP-ABE). The times are in seconds. It is helpful to compare the cost of the original scheme (with a citation) to the cost of the online phase of the given algorithms. In three of the four schemes presented, all bilinear group operations for encryption can be shifted to the offline phase.

then over 99% of the work could be done offline. It is also worth noting that the total computation required between the offline and online phases is nearly identical to the work required by the original scheme. Thus, the total work remains the same, but the vast majority of it can be shifted in time to a moment when the device is least busy or has access to a power source.

We remark that the operation counts given here for the schemes in [19] differ slightly from the summary given in that work. The counts from [19] were obtained from the Charm [1] benchmarking utility, which may have performed various optimizations, whereas ours are a strict count of operations from the algorithms as presented in the paper [19]. We do not expect these differences to have any significant impact on the estimates in Figures 1 and 2.

7 Conclusions

We are exploring methods to make attribute-based encryption (ABE) more efficient for deployment. To this end, we investigated how devices might quickly encrypt ABE messages or generate user keys, even for complex policies.

We developed new “connect and correct” techniques for ABE that split the computation for encryption and key generation into two phases: a preparation phase that does the vast majority of the work to encrypt a message or create a secret key *before* it knows the message or the attribute list/access control policy that will be used (or even the size of the list or policy). A second phase can then rapidly assemble an ABE ciphertext or key when the specifics become known.

Key Generation Algorithm	Bilinear Operations	Est. Time $P = 10$	Est. Time $P = 100$
KP-ABE from [19, App. C]	$5P\mathbb{E}_2 + 2P\mathbb{M}_2$	.370	3.703
KP-Pool-Offline Sec. 5.1	$5P\mathbb{E}_2 + 2P\mathbb{M}_2$	.370	3.703
KP-Pool-Online Sec. 5.1	0	< .001	< .001
CP-ABE from [19]	$(3P + 4)\mathbb{E}_2 + (2P + 1)\mathbb{M}_2$	.252	2.253
CP-Pool-Offline Sec. 5.2	$(3P + 4)\mathbb{E}_2 + (P + 1)\mathbb{M}_2$	.251	2.251
CP-Pool-Online Sec. 5.2	$P\mathbb{M}_2$	< .001	.003

Fig. 2. Performance estimates for regular and online/offline key generation algorithms. We mapped these algorithms into the asymmetric bilinear setting, placing the ciphertexts in $\mathbb{G}_1$ and keys in $\mathbb{G}_2$. Let $\mathbb{E}_i$ (resp., $\mathbb{M}_i$) denote an exponentiation (reps., multiplication) in the group $\mathbb{G}_i$. The bilinear operations are the dominate cost, so we ignore minor factors such as arithmetic in $\mathbb{Z}_p$. The variable P represents the size of the attribute list (in CP-ABE) or the complexity of the access policy (in KP-ABE). The times are in seconds. It is helpful to compare the cost of the original scheme (with a citation) to the cost of the online phase. In both schemes, our estimates show that over 99% of the work to generate a key can be shifted to the offline phase.

This concept is sometimes called “online/offline” encryption. We provided efficient constructions for both key-policy and ciphertext-policy ABE systems.

We provided performance estimates that showed over 99% of the computational work could be moved to offline phase in many scenarios. We expect that this technology could reduce battery consumption on mobile devices and help reduce the bottleneck on a master authority server tasked with generating user keys. Overall, it helps reduce the cost of bringing ABE into practice.

Acknowledgments

The authors thank Joseph Ayo Akinyele and Matthew Green for advice on performance numbers and other helpful comments. Susan Hohenberger was supported in part by NSF CNS-1154035 and CNS-1228443; the Defense Advanced Research Projects Agency (DARPA) and the Air Force Research Laboratory under contract FA8750-11-2-0211, DARPA N11AP20006, the Office of Naval Research under contract N00014-11-1-0470, and a Microsoft Faculty Fellowship. The views expressed are those of the authors and do not reflect the official policy or position of the Department of Defense or the U.S. Government.

References

1. Joseph A. Akinyele, Christina Garman, Ian Miers, Matthew W. Pagano, Michael Rushanan, Matthew Green, and Aviel D. Rubin. Charm: a framework for rapidly prototyping cryptosystems. *Journal of Cryptographic Engineering*, 3(2):111–128, 2013.
2. Joseph Ayo Akinyele and Matthew Green. Personal communication., 2013.

3. D. F. Aranha and C. P. L. Gouvêa. RELIC is an Efficient Library for Cryptography. <http://code.google.com/p/relic-toolkit/>.
4. John Bethencourt, Amit Sahai, and Brent Waters. Ciphertext-policy attribute-based encryption. In *IEEE Symposium on Security and Privacy*, pages 321–334, 2007.
5. Dan Boneh and Xavier Boyen. Efficient selective-id secure identity-based encryption without random oracles. In *EUROCRYPT*, pages 223–238, 2004.
6. Dan Boneh and Brent Waters. Conjunctive, subset, and range queries on encrypted data. In *TCC*, pages 535–554, 2007.
7. Ran Canetti, Shai Halevi, and Jonathan Katz. Chosen-ciphertext security from identity-based encryption. In *EUROCRYPT*, pages 207–222, 2004.
8. Sherman S. M. Chow, Joseph K. Liu, and Jianying Zhou. Identity-based on-line/offline key encapsulation and encryption. In *ASIACCS*, pages 52–60, 2011.
9. Shimon Even, Oded Goldreich, and Silvio Micali. On-line/off-line digital signatures. *J. Cryptology*, 9(1):35–67, 1996.
10. Vipul Goyal, Omkant Pandey, Amit Sahai, and Brent Waters. Attribute-based encryption for fine-grained access control of encrypted data. In *ACM Conference on Computer and Communications Security*, pages 89–98, 2006.
11. Matthew Green, Susan Hohenberger, and Brent Waters. Outsourcing the decryption of ABE ciphertexts. In *USENIX Security Symposium*, 2011.
12. Fuchun Guo, Yi Mu, and Zhide Chen. Identity-based online/offline encryption. In *Financial Cryptography*, pages 247–261, 2008.
13. Susan Hohenberger and Brent Waters. Online/offline attribute-based encryption, 2014. The full version is available from the IACR ePrint Archive, Report 2014/021.
14. Allison B. Lewko and Brent Waters. Unbounded HIBE and attribute-based encryption. In *EUROCRYPT*, pages 547–567, 2011.
15. Allison B. Lewko and Brent Waters. New proof methods for attribute-based encryption: Achieving full security through selective techniques. In *CRYPTO*, pages 180–198, 2012.
16. Joseph K. Liu and Jianying Zhou. An efficient identity-based online/offline encryption scheme. In *ACNS*, pages 156–167, 2009.
17. Zhongren Liu, Li Xu, Zhide Chen, Yi Mu, and Fuchun Guo. Hierarchical identity-based online/offline encryption. In *ICYCS*, pages 2115–2119, 2008.
18. Matthew Pirretti, Patrick Traynor, Patrick McDaniel, and Brent Waters. Secure attribute-based systems. In *ACM Conference on Computer and Communications Security*, pages 99–112, 2006.
19. Yannis Rouselakis and Brent Waters. Practical constructions and new proof methods for large universe attribute-based encryption. In *ACM Conference on Computer and Communications Security*, pages 463–474, 2013.
20. Amit Sahai and Brent Waters. Fuzzy identity-based encryption. In *EUROCRYPT*, pages 457–473, 2005.
21. S. Sharmila Deva Selvi, S. Sree Vivek, and C. Pandu Rangan. Identity based online/offline encryption and signcryption schemes revisited. In *InfoSecHiComNet*, pages 111–127, 2011.
22. Adi Shamir and Yael Tauman. Improved online/offline signature schemes. In *CRYPTO*, pages 355–367, 2001.
23. Patrick Traynor, Kevin R. B. Butler, William Enck, and Patrick McDaniel. Realizing massive-scale conditional access systems through attribute-based cryptosystems. In *NDSS*, 2008.
24. Brent Waters. Ciphertext-policy attribute-based encryption: An expressive, efficient, and provably secure realization. In *PKC*, pages 53–70, 2011.

Decentralized Anonymous Credentials: Identity and Reputation Management without Trusted Parties*

Christina Garman
cgarman@cs.jhu.edu

Matthew Green
mgreen@cs.jhu.edu

Ian Miers
imiers@cs.jhu.edu

The Johns Hopkins University Department of Computer Science
Baltimore, MD, USA

ABSTRACT

Anonymous credentials provide a powerful tool for making assertions about identity while maintaining privacy. However, a limitation of today's anonymous credential systems is the need for a trusted credential issuer — which is both a single point of failure and a target for compromise. Furthermore, the need for such a trusted issuer can make it challenging to deploy credential systems in practice, particularly in the *ad hoc* network setting (e.g., anonymous peer-to-peer networks) where no single party can be trusted with this responsibility.

In this work we propose a novel anonymous credential scheme that eliminates the need for a trusted credential issuer. Our approach builds on recent results in the area of electronic cash, and uses techniques — such as the calculation of a distributed transaction ledger — that are currently in widespread deployment in the Bitcoin payment system. Using this decentralized ledger and standard cryptographic primitives, we propose and provide a proof of security for a powerful anonymous credential system, one that allows users to make flexible identity assertions with strong privacy guarantees. We further show how our basic system can be extended to provide a variety of powerful features including *k*-show credentials and “updatable” credentials that admit a fully anonymous reputation system. From this we construct a distributed anonymous reputation system that assigns users a single global reputation score. Finally, we discuss a number of practical applications for our techniques, including resource management in *ad hoc* networks and prevention of Sybil attacks.

1. INTRODUCTION

Traditionally, making statements about identity on the Internet, whether actual assertions of identity (“I am Spar-

*This work was partially supported by DARPA and the Air Force Research Laboratory (AFRL) under contract FA8750-11-2-0211. This document is a pre-print for DARPA and not for public distribution.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

Copyright 20XX ACM X-XXXXX-XX-X/XX/XX ...\$15.00.

tacus”) or about one’s identity (“I am a gladiator”) involves centralized providers who verify the identity claim and issue a credential attesting to that verification. These organizations, which include Certificate Authorities, DNS maintainers, or login providers like Google and Facebook, play a large role in securing internet infrastructure, email, and financial transactions. Our increasing reliance on these providers elicits two concerns: first, the use of global identifiers raises issues about privacy and user tracking. Second — and more fundamentally — in many settings such as *ad hoc networks*, it may be challenging to identify parties who can be trusted to play this critical role.

Anonymous credentials, introduced by Chaum [19] and developed in a line of subsequent works [8, 11, 14, 12, 4], represent a powerful solution to the privacy concern: they deprive even colluding credential issuers and verifiers of the ability to identify and track their users. Unfortunately, anonymous credentials exacerbate the second issue. Most existing credential systems assume a trusted, centralized “issuer,” which is typically responsible both for *certifying* users’ identity statements and for *issuing* the corresponding anonymous credentials. Indeed, the trust requirement for an anonymous credential issuer is arguably higher than the corresponding requirement for a standard identity management system, since the anonymity properties make it difficult to detect provider malfeasance (e.g., the creation of false credentials). Ideally we could distribute both the certification of identities and the issuing of credentials.

Until recently, it seemed difficult to contemplate certifying identities without assuming a trusted party. However, several recent developments have offered an alternative and *decentralized* vision for addressing this problem. In these systems, exemplified by Namecoin [37], peers collaborate to maintain a distributed append-only transaction ledger for identity assertions. These systems employ the ledger to record mappings from names to public keys and can enforce complex semantics over the identity assertions. The Namecoin system has already been deployed to implement a decentralized version of the DNS system [25], and it is easy to imagine further extensions such as a decentralized SSL Public Key Infrastructure, time stamping, and reputation networks. The ability to establish and make statements about identity (e.g., “I have an identity, it required some effort to create, and I have used it in successful transactions”) seems particularly useful for applications related to *ad hoc networks*, which are famously vulnerable to Sybil attacks [26].

Even given the above systems, decentralized *issuing* of

anonymous credentials remains elusive. Specifically, decentralized systems seem incompatible with most existing anonymous credential techniques, which typically rely on blind signatures. The use of signatures necessitates a trusted issuer — or at least a small quorum of cooperating parties who hold shares of the private key. Unfortunately, implementing threshold cryptographic techniques seems challenging if we wish to widely distribute trust or deal with ad hoc applications where nodes routinely enter and exit the network. This appears to leave us with a binary choice: we must choose centralized anonymous credentials that protect our privacy or decentralized credentials that offer no privacy protection.

Our contribution. In this paper we show how to construct anonymous credentials in a decentralized setting. Our approach extends prior work on anonymous credentials but removes the need to appoint a trusted credential issuer. A consequence of this result is that our credential system can be instantiated on-demand and operated by an *ad hoc* group of mistrustful peers. We show how to extend our credential scheme to create *updatable* (e.g. stateful) anonymous credentials in which users obtain new credentials based on changing properties of their identity. Finally, from updatable credentials we construct the first fully peer-to-peer anonymous reputation system where users have a single global reputation score. We believe the reputation system may be of independent interest.

As a basic ingredient, our protocols require the existence of a *distributed public append-only ledger*, a technology which has most famously been deployed in distributed electronic currencies such as Bitcoin [36]. This ledger can be employed by individual nodes to make assertions about identity in a fully anonymous fashion *without* the assistance of a credential issuer.

We believe that our techniques have several immediate applications. These include:

- **Anonymous resource management in ad hoc networks.** Peer-to-peer networks are vulnerable to impersonation attacks, where a single party simulates many different peers in order to gain advantage against the network [26]. We show that our credentials may be useful in mitigating these attacks. The basic approach is to construct an *anonymous subscription service* [22, 9, 34] where parties would establish unique or costly pseudonyms (for example by completing a computational proof of work, submitting a payment, or demonstrating satisfaction of some network criteria). They can then assert possession on their identity under a specific set of restrictions, e.g., a limit to the number of requests they can make in each time period.
- **Anonymous reputation management for ad hoc networks.** A further extension of our construction allows parties to *update* credentials, i.e., by inserting new credentials into the block chain. This admits a variety of sophisticated enhancements such as fully anonymous *reputation systems* [42, 1] in which parties update and prove statements about a reputation score following successful (or unsuccessful) interactions. These systems can be useful in situations where reputation information improves network efficiency but where traditional (non-anonymous) reputation systems may expose high-reputation nodes to malicious attacks.

- **Auditable credentials.** Our techniques may also be used to extend existing centralized credential systems by allowing for public audit of issued credentials. This helps to guard against compromised credential issuers and allows the network to easily detect and revoke inappropriate credential grants. For example, in Direct Anonymous Attestation (DAA) one might want to prevent a malicious DAA authority from covertly granting certificates to users who do not have a TPM or whose TPM did not attest.

1.1 Overview of Our Construction

We now provide a brief overview for our construction, which is inspired by the electronic cash proposals of Sander and Ta-Shma [45] and Miers et al. [35]. We begin with a brief overview of decentralized identification techniques.

Decentralized identities. Decentralized identity systems such as Namecoin are based on two assumptions: the existence of a distributed append-only transaction ledger and a set of consensus rules for accepting new entries into this ledger. In a basic PKI system, users are permitted to assert ownership of arbitrary identity strings by submitting a transaction embedding the identity as well as a public key for some digital signature scheme. The network then evaluates the legitimacy of each claim according to rules defined in the protocol. In a simple example — which corresponds to the Namecoin DNS system — pseudonyms may be arbitrary strings and will be accepted by the network on the condition that the pseudonym has not previously been claimed.

Naturally it is possible to apply more complex rules to this process. For example, identity claims may be conditioned on the submission of a publicly-verifiable proof of work or even proof that a sum of electronic currency has been spent or destroyed. Identity claims may also leverage one or more existing non-anonymous identity credentials: for example, a proof that the claimant has a certificate from a PKI and/or a signature using a TPM enrollment key. Identities may also include externally verifiable assertions about the claiming party, e.g., that the party is contributing a specific amount of resources to a peer-to-peer network.

Issuing and showing credentials. The ability to establish identities and bind them to a public key ensures that users can assert their identity in a non-anonymous fashion, simply by issuing signatures from the corresponding secret key. Unfortunately, this does not immediately show us how to construct anonymous credentials, since traditional anonymous credentials consist of a signature computed by a credential issuer. Since no central party exists to compute the credential signature, this approach does not seem feasible without elaborate (and inefficient) use of threshold cryptography.¹

We instead take a different approach. To issue a new credential in our decentralized system, the user establishes an identity and related attributes as described above. She then attaches a vector commitment to her secret key sk_U along with the identity and attribute strings that are contained within her identity assertion. Finally, she includes a non-interactive proof that the credential is correctly constructed,

¹A possibility is to use ring signatures [43], which do not require a single trusted signer. Unfortunately these signatures grow with the number of participating signers and require expensive communication to generate.

i.e., that the attributes in the commitment correspond to those revealed in the identity assertion. The network will accept the identity assertion if and only if the assertion is considered valid, and the attached proof is valid.

At a later point an individual can prove possession of such a credential by proving the following two statements in zero-knowledge:

1. She knows a commitment C_i in the set $(C_1, \dots, C_N)$ of all credentials previously accepted to the block chain.
2. She knows the opening (randomness) for the commitment.

In addition to this proof, the user may simultaneously prove additional statements about the identity and attributes contained within the commitment C_i . The challenge in the above construction is to efficiently prove statements (1) and (2), i.e., without producing a proof that scales with N . Our solution, which adapts techniques from distributed e-cash systems [35], circumvents this problem by using an efficient publicly-verifiable accumulator [12] to gather the set of all previous commitments together. Using this accumulator in combination with an efficient membership proof due to Camenisch and Lysyanskaya in [13], we are able to reduce the size of this proof to $O(\lambda)$ for security parameter λ , rather than the $O(N \cdot \lambda)$ proofs that would result from a naive OR proof.

Of course, merely applying these techniques does not lead to a *practical* credential system. A key contribution of this work is to supply concrete instantiation of the above idea under well-studied assumptions, and to prove that our construction provides for consistency of credentials (ensuring multiple users cannot pool their credentials), the establishment of pseudonyms, and a long set of extensions built upon anonymous credentials. Last but not least we need to formally define and prove the security of a distributed anonymous credential scheme and provide some model for the distributed ledger. Our instantiation requires a single trusted setup phase, after which the trusted party is no longer required.²

Extensions: stateful credentials and reputation systems. We argue that our basic credential system is sufficient to permit many useful activities in a decentralized network. However, there are also cases where the user’s attributes may be a function of previous transactions on the network. One example is a decentralized reputation system, where the user periodically updates his “attribute” (reputation count) based on previous interactions with other users. In §6 and §7 we propose extensions to our basic anonymous credential system that allow users to update their credential based on interactions with other parties. The key to our proposal is that users can prove knowledge of a previous transaction and an interaction with a third party and use these proofs as the condition for obtaining a new credential. This means that users can interact and exchange reputation according to a set of network-wide transaction rules, *without* revealing which users interacted and how that interaction affected their score. Users may later prove statements about their current reputation score, within a range of their choice. We believe that this has numerous applications to peer networks with

²In §6 we discuss techniques for removing this trusted setup requirement.

nodes of varying capabilities, e.g., electing supernodes in hierarchical peer networks.

1.2 Outline of This Work

The remainder of this work is organized as follows. In the next section we discuss specific applications for decentralized anonymous credentials and argue that these systems can be used to solve a variety of problems in peer-to-peer networks. In §3 we describe the cryptographic building blocks of our construction, and in §4 we define the notion of a *decentralized anonymous credential scheme* and provide an ideal-world security definition. In §5 we provide an overview of our basic construction as well as a specific instantiation based on the Discrete Logarithm and Strong RSA assumption. In §6 we extend our basic construction to add a variety of useful features, including k -show credentials, stateful credentials, and credential revocation. Finally, in §7 we detail the first distributed, identity-bound, anonymous reputation systems where users have a single global reputation.

2. APPLICATIONS

In this section we discuss several of the applications facilitated by decentralized anonymous credentials. While we believe that these credential systems may have applications in a variety of environments, we focus specifically on settings where trusting a central credential issuer is not an option or where issued credentials must be publicly audited.

Mitigating Sybil attacks in ad hoc networks. Impersonation attacks can have grave consequences for both the security and resource allocation capabilities of ad hoc networks. A variety of solutions have been proposed to address this problem. One common approach is to require that clients solve *proofs of work*: resource-consuming challenges that typically involving either storage or computation [26]. Unfortunately it can be challenging to re-use a single proof of work in an anonymous fashion, i.e., without either identifying participants or allowing other users to clone the solution.

One solution to this problem is to use k -show anonymous credentials. In this approach, peers establish a single credential by solving a proof of work. This allows the peer to obtain a credential that can be used a limited number of times or a limited number of times within a given time period. When a peer exceeds the k -use threshold (e.g., by cloning the credential for a Sybil attack), the credential can be identified and revoked. We note that this proposal is a distributed variant of the *anonymous subscription service* concept, which was first explored in [22, 9].

Managing resource usage. In networks where peers both contribute and consume resources, ensuring fair resource utilization can be challenging. For example, a storage network might wish to ensure peers provide as much storage as they consume [39] or ensure that peers fairly use network bandwidth [40]. This can be problematic in networks that provide anonymity services (e.g. Tor) or VOIP³, where peers may be reluctant to identify which traffic they originated. An anonymous credential system allows peers to identify their contributions to routing traffic in exchange for a credential which they can then use to originate traffic. This helps to

³Prior its acquisition by Microsoft, Skype used a peer-to-peer overlay network

ensures that peers contribute resources back to the network while preserving their anonymity.

Reputation in peer-to-peer systems. As noted in [49], the lack of trust relationships can be a fundamental problem in peer-to-peer systems. This problem is exacerbated in anonymous communication networks, where peer interactions can reveal sensitive information about usage history. The distributed anonymous reputation system we describe in §7 allows nodes to establish a reputation based on a past history of positive interactions with other nodes. We offer two examples where this is useful.

First, consider the usage of Tor to circumvent censorship by state level actors. While the identities of Tor nodes themselves are public, entrance nodes (called *bridges*) need to remain both (1) unblocked and thus unknown to censors and (2) available to honest users. Wang et al. [48] propose a solution to these seemingly incompatible goals: distribute bridge addresses only to high-reputation users, and define a user's reputation as the uptime of the bridges they know about. Being centralized, their system requires a central party who must be trusted both not to reveal the whole bridge list and with accurately maintaining users' reputations. Our proposal allows us to eliminate this party.

Similarly, file sharing networks may also benefit from reputation, as these networks are vulnerable to malicious peers uploading inauthentic files. Kamvar et al. [32] identify this problem and propose a reputation system as a solution. However, their proposed solution associates each user with an "opaque identifier" and thus offers users very little privacy.

3. PRELIMINARIES

We make use of the following complexity assumptions and cryptographic building blocks to construct our scheme.

3.1 Complexity Assumptions

The security of our scheme relies on the following two complexity assumptions:

Strong RSA Assumption [3, 28]. Given a randomly generated RSA modulus n and a random element $y \in \mathbb{Z}_n^*$, it is hard to compute $x \in \mathbb{Z}_n^*$ and integer exponent $e > 1$ such that $x^e \equiv y \pmod{n}$. We can restrict the RSA modulus to those of the form pq , where $p = 2p' + 1$ and $q = 2q' + 1$ are safe primes.

Discrete Logarithm (DL) Assumption [23]. Let G be a cyclic group with generator g . Given $h \in G$, it is hard to compute x such that $h = g^x$.

3.2 Cryptographic Building Blocks

Zero-knowledge proofs. In a zero-knowledge protocol [30] a user (the prover) proves a statement to another party (the verifier) without revealing anything about the statement other than that it is true. Our constructions use zero-knowledge proofs that can be instantiated using the technique of Schnorr [47], with extensions due to, e.g., [21, 15, 17, 7]. We convert these into *non-interactive* proofs by applying the Fiat-Shamir heuristic [27]. When we use these proofs to authenticate auxiliary data, we refer to the resulting non-interactive proofs as *signatures of knowledge* as defined in [18].

When referring to these proofs we will use the notation of Camenisch and Stadler [16]. For instance, $\text{NIZKPoK}\{(x, y) :$

$h = g^x \wedge c = g^y\}$ denotes a non-interactive zero-knowledge proof of knowledge of the elements x and y that satisfy both $h = g^x$ and $c = g^y$. All values not enclosed in $()$'s are assumed to be known to the verifier. Similarly, the extension $\text{ZKSoK}[m]\{(x, y) : h = g^x \wedge c = g^y\}$ indicates a *signature of knowledge* on message m .

Accumulators [35]. An accumulator allows us to combine many values into one smaller value (the accumulator). We then have a single element, called the witness, that allows us to attest to the fact that a given value is actually part of the accumulator. Our constructions use an accumulator based on the Strong RSA assumption. The accumulator we use was first proposed by Benaloh and de Mare [5] and later improved by Baric and Pfitzmann [3] and Camenisch and Lysyanskaya [12]. We describe the accumulator using the following algorithms:

- **AccumSetup**(λ) $\rightarrow$ *params*. On input a security parameter, sample primes p, q (with polynomial dependence on the security parameter), compute $N = pq$, and sample a seed value $u \in QR_N, u \neq 1$. Output (N, u) as *params*.
- **Accumulate**(*params*, $\mathbf{C}$) $\rightarrow A$. On input *params* (N, u) and a set of prime numbers $\mathbf{C} = \{c_1, \dots, c_i \mid c \in [\mathcal{A}, \mathcal{B}]\}$,⁴ compute the accumulator A as $u^{c_1 c_2 \dots c_n} \pmod{N}$.
- **GenWitness**(*params*, v , $\mathbf{C}$) $\rightarrow \omega$. On input *params* (N, u) , a set of prime numbers $\mathbf{C}$ as described above, and a value $v \in \mathbf{C}$, the witness ω is the accumulation of all the values in $\mathbf{C}$ besides v , i.e., $\omega = \text{Accumulate}(\text{params}, \mathbf{C} \setminus \{v\})$.
- **AccVerify**(*params*, A , v , ω) $\rightarrow \{0, 1\}$. On input *params* (N, u) , an element v , and witness ω , compute $A' \equiv \omega^v \pmod{N}$ and output 1 if and only if $A' = A$, v is prime, and $v \in [\mathcal{A}, \mathcal{B}]$ as defined previously.

For simplicity, the description above uses the full calculation of A . Camenisch and Lysyanskaya [12] observe that the accumulator may also be *incrementally* updated, i.e., given an existing accumulator A_n it is possible to add an element x and produce a new accumulator value A_{n+1} by computing $A_{n+1} = A_n^x \pmod{N}$.⁵

Camenisch and Lysyanskaya [12] show that the accumulator satisfies a strong *collision-resistance* property if the Strong RSA assumption is hard. Informally, this ensures that no *p.p.t.* adversary can produce a pair (v, ω) such that $v \notin \mathbf{C}$ and yet **AccVerify** is satisfied. Additionally, they describe an efficient zero-knowledge proof of knowledge that a committed value is in an accumulator. We convert this into a non-interactive proof using the Fiat-Shamir transform and refer to the resulting proof using the following notation:

$$\text{NIZKPoK}\{(v, \omega) : \text{AccVerify}((N, u), A, v, \omega) = 1\}.$$

Verifiable Random Functions. A pseudorandom function (PRF) [29] is an efficiently computable function whose

⁴Where $\mathcal{A}$ and $\mathcal{B}$ can be chosen with arbitrary polynomial dependence on the security parameter, as long as $2 < \mathcal{A}$ and $\mathcal{B} < \mathcal{A}^2$. [13] For a full description, see [13, §3.2 and §3.3].

⁵This allows the network to maintain a running value of the accumulator and prevents individual nodes from having to recompute it [35].

- $RegNym(Nym_U^O, U, O)$: U logs into T_P with sk_U to register a nym with organization O . If she does not have an account, she first creates one. She gives T_P a unique random string Nym_U^O for use as her nym with O . T_P checks that the string is indeed unique and if so stores (Nym_U^O, U, O) and informs U .
- $MintCred(Nym_U^O, O, attrs, aux)$: U logs into T_P authenticating with sk_U . If Nym_U^O is not U 's nym with O or sk_U is wrong, reject. Otherwise, T_P checks that aux justifies issuing a credential under O 's issuing policy and if so generates a unique random id ID and stores $(Nym_U^O, U, ID, attrs)$. It then adds ID to its public list of issued credentials for O .
- $ShowOnNym(Nym_U^O, Nym_V^V, O, V, attrs, \mathbf{C})$: U logs into T_P with sk_U . If Nym_U^O is not U 's nym with O or Nym_V^V is not V 's nym with V , reject. Else, T_P checks if the tuple (Nym_U^O, U) exists, if ID associated with that tuple is in the set of credentials $\mathbf{C}$ that U provided, and if the given attributes $attrs$ match the attributes associated with that tuple. If all conditions hold, T_P informs V that Nym_V^V has a credential from O in the set $\mathbf{C}$. V then retrieves the set of credentials $\mathbf{C}_O$ issued by O from T_P and accepts T_P 's assertion if and only if $\mathbf{C} \subseteq \mathbf{C}_O$ and O 's issuing policy is valid $\forall c' \in \mathbf{C}_O$.
- $GetCredList(O)$: T_P retrieves the list of credentials for organization O and returns it.

Figure 1: Ideal Functionality. Security of a basic distributed anonymous credential system.

output cannot be distinguished (with non-negligible advantage) from random by a computationally bounded adversary. We denote the pseudorandom function as $f_k(\cdot)$, where k is a randomly chosen key. A number of PRFs possess efficient proofs that a value is the output of a PRF on a set of related public parameters. Two examples of this are the Dodis-Yampolskiy (DY) PRF [24] and the Naor-Reingold PRF [38].

Pedersen Commitments. A commitment scheme allows a user to bind herself to a chosen value without revealing that value to the recipient of the commitment. This commitment to the value ensures that the user cannot change her choice (i.e., *binding*), while simultaneously ensuring that the recipient of the commitment does not learn anything about the value it contains (i.e., *hiding*) [20]. In Pedersen commitments [41], the public parameters are a group G of prime order q , and generators $(g_0, \dots, g_m)$. In order to commit to the values $(v_1, \dots, v_m) \in \mathbb{Z}_q^m$, pick a random $r \in \mathbb{Z}_q$ and set $C = \text{PedCom}(v_1, \dots, v_m; r) = g_0^r \prod_{i=1}^m g_i^{v_i}$.

4. DECENTRALIZED ANONYMOUS CREDENTIALS

A traditional anonymous credential system has two types of participants: users and organizations. Users, who each have a secret key sk_U , are known by pseudonyms both to each other and organizations. Nym_A^O , for example, is the pseudonym of user A to organization O . Decentralized anonymous credentials have no single party representing the organization. Instead, this party is replaced with a quorum of users who enforce a specific *credential issuing policy* and collaboratively maintain a list of credentials thus far issued. For consistency with prior work, we retain the term “organization” for this group.

A distributed anonymous credential system consists of a global transaction ledger as well as the following (possibly probabilistic) algorithms:

- $\text{Setup}(1^\lambda) \rightarrow params$. Generates the system parameters.
- $\text{KeyGen}(params) \rightarrow sk_U$. Run by a user to generate her secret key.

- $\text{FormNym}(params, U, E, sk_U) \rightarrow (Nym_U^E, sk_{Nym_U^E})$. Run by a user to generate a pseudonym Nym_U^E and an authentication key $sk_{Nym_U^E}$ between a user U and some entity (either a user or an organization) E .
- $\text{MintCred}(params, sk_U, Nym_U^O, sk_{Nym_U^O}, attrs, aux) \rightarrow (c, sk_c, \pi_M)$. Run by a user to generate a request for a credential from organization O . The request consists of a candidate credential c containing public attributes $attrs$; the user's key sk_U ; auxiliary data aux justifying the granting of the credential; and a proof π_M that (1) Nym_U^O was issued to the same sk_U and (2) the credential embeds $attrs$.
- $\text{MintVerify}(params, c, Nym_U^O, aux, \pi_M) \rightarrow \{0, 1\}$. Run by nodes in the organization to validate a credential. Returns 1 if π_M is valid, 0 otherwise.
- $\text{Show}(params, sk_U, Nym_V^V, sk_{Nym_V^V}, c, sk_c, \mathbf{C}_O) \rightarrow \pi_S$. Run by a user to non-interactively prove that a given set of attributes are in a credential c in the set of issued credentials $\mathbf{C}_O$ and that c was issued to the same person who owns Nym_V^V . Generates and returns a proof π_S .
- $\text{ShowVerify}(params, Nym_V^V, \pi_S, \mathbf{C}_O) \rightarrow \{0, 1\}$. Run by a verifier to validate a shown credential. Return 1 if π_S is valid for Nym_V^V , 0 otherwise.

4.1 Security

We define our system in terms of an ideal functionality implemented by a trusted party T_P that plays the role that our cryptographic constructions play in the real system. All communication takes place through this ideal trusted party. Security and correctness for our system comes from a proof that this ideal model is indistinguishable from the real model provided the cryptographic assumptions hold. Our ideal functionality is outlined in Figure 1.

It consists of organizations who issue credentials and users who both prove that they have these credentials and verify such proofs. Organizations have only two things: 1) an efficient and publicly evaluable policy, *policy_O*, for granting credentials and 2) an append-only list of credentials meeting that policy maintained by the trusted party.

- **Setup**(1^λ) $\rightarrow$ $params$. On input a security parameter λ , run **AccumSetup**(1^λ) to obtain the values (N, u) . Next generate primes p, q such that $p = 2^w q + 1$ for $w \geq 1$. Let $\mathbb{G}$ be an order- q subgroup of $\mathbb{Z}_p^*$, and select random generators $g_0, \dots, g_n$ such that $\mathbb{G} = \langle g_0 \rangle = \dots = \langle g_n \rangle$. Output $params = (N, u, p, q, g_0, \dots, g_n)$.
- **KeyGen**($params$) $\rightarrow$ sk . On input a set of parameters $params$, select and output a random master secret $sk \in \mathbb{Z}_q$.
- **FormNym**($params, sk$) $\rightarrow$ (Nym, sk_{Nym}) . Given a user's master secret sk , select a random $r \in \mathbb{Z}_q$ and compute $Nym = g_0^r g_1^{sk}$. Set $sk_{Nym} = r$ and output (Nym, sk_{Nym}) .
- **MintCred**($params, sk, Nym_U^O, sk_{Nym_U^O}, attrs, aux$) $\rightarrow$ (c, sk_c, π_M) . Given anym Nym_U^O and its secret key $sk_{Nym_U^O}$; attributes $attrs = (a_0, \dots, a_m) \in \mathbb{Z}_q$; and auxiliary data aux , select a random $r' \in \mathbb{Z}_q$ and compute $c = g_0^{r'} g_1^{sk} \prod_{i=0}^m g_{i+2}^{a_i}$. Set $sk_c = r'$ and output (c, sk_c, π_M) where π_M is a signature of knowledge on aux that thenym and the credential both belong to the same master secret sk , i.e.:

$$\pi_M = \text{ZKSoK}[aux]\{(sk, r', r) :$$

$$c = g_0^{r'} g_1^{sk} \prod_{i=0}^m g_{i+2}^{a_i} \wedge Nym_U^O = g_0^r g_1^{sk}\}$$

Finally, submit the resulting values $(c, \pi_M, attrs)$ to the public transaction ledger.

- **MintVerify**($params, c, attrs, Nym_U^O, aux, \pi_M$) $\rightarrow$ $\{0, 1\}$. Given a credential c , attributes $attrs$, anym Nym_U^O , and proof π_M , verify that π_M is the signature of knowledge on aux . If the proof verifies successfully, output 1, otherwise output 0. The organization nodes should accept the credential to the ledger if and only if this algorithm returns 1.
- **Show**($params, sk, Nym_U^V, sk_{Nym_U^V}, c, attrs, sk_c, \mathbf{C}_O$) $\rightarrow$ π_S . Given a user's master secret sk ; anym Nym_U^V between the user and the verifier and its secret key $sk_{Nym_U^V}$; a credential c and its secret key sk_c ; the attributes $(a_0, \dots, a_m)$ used in the credential; and a set of credentials $\mathbf{C}$, compute $A = \text{Accumulate}(params, \mathbf{C}_O)$ and $\omega = \text{GenWitness}(params, c, \mathbf{C}_O)$ and output the following proof of knowledge:

$$\pi_S = \text{NIZKPoK}\{(sk, \omega, r', c, r, Nym_U^V) :$$

$$\text{AccVerify}(params, A, c, \omega) = 1 \wedge c = g_0^{r'} g_1^{sk} \prod_{i=0}^m g_{i+2}^{a_i} \wedge Nym_U^V = g_0^r g_1^{sk}\}$$

- **ShowVerify**($params, Nym_U^V, \pi_S, \mathbf{C}_O$) $\rightarrow$ $\{0, 1\}$. Given anym Nym_U^V , proof of possession of a credential π_S , and the set of credentials issued by organization O $\mathbf{C}_O$, first compute $A = \text{Accumulate}(params, \mathbf{C}_O)$. Then verify that π_S is the aforementioned proof of knowledge on c , $\mathbf{C}_O$, and Nym_U^V using the known public values. If the proof verifies successfully, output 1, otherwise output 0.

Figure 2: Our basic decentralized anonymous credential scheme.

4.2 Trusting the Ledger

An obvious question is whether the append-only transaction ledger is necessary at all. Indeed, if the list of valid credentials can be evaluated by a set of untrusted nodes, then it seems that a user (Prover) could simply maintain a credential list compiled from network broadcasts and provide this list to the Verifier during a credential show. However, this approach can enable sophisticated attacks where a malicious Verifier manipulates the Prover's view of the network to include a poisoned-pill credential that — although valid by the issuing heuristic — was not broadcast to anyone else. When the prover authenticates, she has completely identified herself.

The distributed transaction ledgers employed by networks such as Bitcoin and Namecoin provide a solution to this problem, as their primary purpose is to ensure a shared view among a large number of nodes in an adversarial network. In practice this is accomplished by maintaining a high degree of network connectivity and employing computational *proofs of work* to compute a hash chain.

For an attacker to execute the poisoned credential attack

against such a ledger, she would need to both generate and maintain a false view of the network to delude the prover. This entails both simulating the prover's view of the rest of the network complete with *all* its computational power and forging any assurances the prover might expect from known peers about the present state of the network. If the prover has a reasonable estimate of the actual network's power (e.g., she assumes it monotonically increases), then an attacker must actually have equivalent computational power to the entirety of the network to mount such an attack. For the purposes of this paper we assume such active attacks are impossible even if the attacker controls a simple majority of the computational power. Attackers are still free to attempt any and all methods of retroactively identifying a user and mount any other active attacks.

5. OUR CONSTRUCTION

We now provide an overview and instantiation of our construction.

5.1 Overview

Alice’s pseudonym with a given organization/user is an arbitrary identity that she claims in a transaction. She tags this value with a Pedersen commitment to her secret key sk and *signs* the resulting transaction using a signature of knowledge that she knows the secret key. There is no separate process for registering a pseudonym: instead they are simply used in issue and show to allow operations to be linked if necessary. Alice’s credential c is a vector Pedersen commitment to both sk and a set of public attributes $attrs = a_0, \dots, a_m$, which Alice also includes in her credential. To issue a credential, Alice provides the network with a credential, a pseudonym, her attributes, optionally some auxiliary data justifying the credential issue (e.g., a proof of work that Alice is not a Sybil), and a proof that (1) the commitment and the pseudonym contain the same secret key and (2) the attributes are in some allowed set. If all of this validates, the entry is added to the ledger. Alice shows the credential under a different pseudonym by proving in zero-knowledge that (1) she knows a credential on the ledger from the organization, (2) the credential opens to the same sk as her pseudonym, and (3) it has some attributes.

5.2 Concrete Instantiation

We instantiate our system with the cryptographic primitives discussed in Section 3. Specifically, we use Pedersen commitments and a Strong RSA based accumulator. The proofs are conducted using standard techniques [47, 12] and are similar to the proofs used by Miers et al. in [35]. See Figure 2 for details.

THEOREM 5.1. *The basic distributed anonymous credential system described in Figure 2 is secure in the random oracle model under the Strong RSA and the Discrete Logarithm assumptions.*

We sketch of the proof of Theorem 5.1 in Appendix A.

6. EXTENSIONS

We consider extending the basic system in several ways.

6.1 k -show Credentials

Damgård et al. [22] first suggested a credential system where users could only authenticate once per time period. Camenisch et al. [9] independently proposed a significantly more efficient construction that allows for up to k authentications per time period, with the ability to revoke all cloned credentials if a credential was used beyond this limit. Camenisch et al. suggested that these technique might be used to build anonymous subscription services, allowing users to access a resource (such as a website) within reasonable bounds. We briefly show that these same techniques can be applied to our basic credential system.

In the system of [9] an authority issues a credential on a user’s secret seed s . To show a credential for the i^{th} time in validity period t , the user generates a serial number S using a verifiable random function (VRF) as $S = f_s(0||t||i)$. She also includes a non-interactive zero-knowledge proof that this serial number is correctly structured.⁶ This technique can be applied to our construction: the user simply generates a

⁶The re-use of a credential would result in a repeated serial number, and yet the nature of the VRF’s output (for an honest user) ensures that attackers cannot link individual shows.

random seed s and includes this value in the commitment she stores in the transaction ledger, along with a non-interactive proof that she knows the value s . We note that for the trivial case of one-time show credentials, we can simply reveal the seed.⁷

6.2 Credentials with Hidden Attributes

In our basic construction of §5, users provide a full list of attributes when requesting and showing credentials. While this is sufficient for many applications, there exist cases where a user might wish to conceal the attributes requested or shown, opting instead to prove statements about them, e.g., proving knowledge of a secret key or proving that an attribute is within a certain range. We note that this requires only minor changes to our protocols — the user simply attaches a proof that the attribute values contained within the credential c satisfy some statement.

6.3 Stateful Credentials

A stateful anonymous credential system [20] is a variant of an anonymous credential system where credential attributes encode some state that can be updated by issuing new credentials. This credential issuance is typically conditioned on the user showing a previous credential and offering proof that the new credential should be updated as a function of the original.

Intuitively, we can add this capability quite easily due to the fact that our credentials are non-interactively issued. We construct a “single show” credential c embedding some state *state* in the attributes and a serial number S . Users are free to show c as many times as they like without revealing the serial number. However, to update the state of the credential, they must author a transaction that shows the original credential and reveals the serial number S and “mint” a new candidate credential c' containing the updated state *state'* (hidden inside of a commitment) and a proof that there exists a valid relationship between the state encoded in c and the new state in c' (for example, that the attributes have been incremented).

This requires only minor extensions to our basic scheme. In this case we add an **Update** algorithm that operates similarly to **MintCred** but includes the earlier credential and a proof of its construction. A valid proof of the existing credential now becomes a condition for the organization accepting the updated credential into the ledger. We provide a description of this new algorithm in Figure 3.

6.4 Credential Revocation

Several previous works have proposed techniques for revoking anonymous credentials [12, 10, 2]. We note that several of these techniques can be applied to our credential system. However, we also observe that our use of a centralized transaction ledger may improve the security of the existing proposals. Suppose our goal is only to revoke systems where a user’s key secret key has been publicly compromised. A common technique is to simply place such keys on a list and have the user prove the secret key in his commitment is not on the list. After all, no legitimate user’s key can appear on this list. However, we observe that without a globally-shared

⁷Camenisch et al. [9] describe a further extension that reveals the user’s identity in the event of a credential double-show. We omit the details here for space reasons but observe that the same technique can be applied to our construction.

- **Update**($params, sk, c, sk_c, \mathbf{C}_O, update\_relation, state'$) $\rightarrow (c', sk'_c, \pi_u)$. Given a credential c and associated secret key sk_c , a set of credentials $\mathbf{C}_O$, an updated state $state' = (s'_0, \dots, s'_m) \in \mathbb{Z}_q$, and an update relation $update\_relation$, generate a fresh random serial number $S' \in \mathbb{Z}_q$ and random value $r' \in \mathbb{Z}_q$ to form a new credential $c' = g_0^{r'} g_1^{sk} g_2^{S'} \prod_{i=0}^m g_{i+3}^{s'_i}$. Compute $A = \text{Accumulate}(params, \mathbf{C}_O)$ and $\omega = \text{GenWitness}(params, c, \mathbf{C}_O)$. Output (c', sk'_c, π_u) where $sk'_c = (S', state', r')$ and $\pi_u = \text{NIZKPoK}\{(sk, \omega, c, state, r, c', state', r') : \text{AccVerify}(params, A, c, \omega) = 1 \wedge c = g_0^r g_1^{sk} g_2^S \prod_{i=0}^m g_{i+3}^{s_i} \wedge c' = g_0^{r'} g_1^{sk} g_2^{S'} \prod_{i=0}^m g_{i+3}^{s'_i} \wedge update\_relation(state, state') = 1\}$
- **UpdateVerify**($params, c, \mathbf{C}_O, \pi_u$) $\rightarrow \{0, 1\}$. Given a stateful credential c , a credential set $\mathbf{C}_O$, and proof π_u , output 1 if π_u is correct, the proved state transition is a legal one, and the serial number S was not previously used. Otherwise 0.

Figure 3: Extensions for a stateful anonymous credential system. $update\_relation(\dots) = 1$ denotes that the update encodes some arbitrary state transition (e.g. $state' = state + 1$).

state, a malicious revocation authority might be able to feed different users different revocation lists, then collude with verifiers to identify users.

Although we use an append-only list and cannot simply delete the credential outright, we can add a marker stating that the credential is no longer valid. Since this list is public and shared amongst all the users, any user whose credential is revoked will simply not attempt to authenticate. We thus can realize almost any credential revocation policy without a trusted party who could deanonymize users.

6.5 Avoiding trusted setup

Setting up the accumulator in our basic construction requires generating an RSA modulus. However, its factorization is never used and in fact must remain unknown to prevent forged membership proofs. Because of this, we could use a single trusted setup phase where the factorization is destroyed immediately after the parameters are generated. A more elegant option is to use so called RSA UFOs [44] for accumulator parameters without a trapdoor, forgoing the need for any trusted parameter generation.

7. ANONYMOUS REPUTATION SYSTEMS

The techniques described thus far allow users to show a credential without fear of linking a pseudonym to their identity. In the extreme case, a user can dispense with persistent pseudonyms in their entirety and make all of her transactions unlinkable. For certain applications, however, this is not desirable: when a user's actions are unlinkable, malicious actions have no consequence and honest users have no way to prove good behavior. A solution to this problem is to create a reputation system in which users have a single global *reputation score* they can demonstrate across pseudonyms. Like an anonymous credential system, any proof of reputation should be completely anonymous even in the face of the retrospective collusion of all other parties in the system.

Although a fair amount of work has been conducted on systems with local reputation, reputation for pseudonyms only [33], or reputation systems secure only if some subset of trusted parties never collude even retroactively [31], there appears to be no distributed solution that provides our

desired properties. To the best of our knowledge the only construction to date which offers anonymous — even if all parties collude after the fact — global reputation is due to Androulaki et al. [1] and requires a semi-trusted third party to keep track of reputation.

Informally, the security model for Androulaki et al.'s scheme requires the following three properties which we use as a starting point for our construction:

1. Anonymity of peers during reputation exchange and reputation demonstration: pseudonyms cannot be linked to each other or to the underlying user by any collection of parties.
2. Unforgeability of reputation points.
3. Inflation resistance: the existence of some fixed number of points allocable per day.

The first two properties map intuitively to the security properties of a credential system (i.e., unforgeability of credentials and anonymity). The last one is a special restriction due to [6]: if reputation can be handed out in an unlimited manner, then the system is worthless.

At first glance, e-cash systems seem to provide an easy way to realize such a system: users spend “repcoins” with each other and an anonymous credential issued by the bank is used to attest to their repcoin balance. However, e-cash systems are only anonymous for the spender: a user and a bank can collude to identify who the repcoins were deposited with. A reputation system needs a superset of the privacy features provided by e-cash: both parties in a transaction need to maintain their anonymity and thus both spends and deposits must be blind. Androulaki et al.'s solution in fact uses blinded deposits to allow a bank trusted to track reputation to issue anonymous reputation credentials.⁸

7.1 A Basic Reputation System

Given a distributed, updatable anonymous credential system, we can instantiate a distributed, global, identity-bound anonymous reputation system as a special class of legal state transitions between stateful credentials. In essence, we view

⁸Making this system distributed would entail constructing both a distributed e-cash scheme and a distributed anonymous credential scheme. We opt for a cleaner approach.

- **RepDec**($params, sk, c, sk_c, \mathbf{C}_O, \Delta$) $\rightarrow (c', sk'_c, comm_\Delta, r_\Delta, \pi_{rd})$. Given a credential c and associated secret key sk_c , a set of credentials $\mathbf{C}_O$, and an amount to decrement the reputation $\Delta \in \mathbb{Z}_q$, calculate the new balance $balance' = balance - \Delta$. Then generate a fresh random serial number $S' \in \mathbb{Z}_q$ and random values $r', r_\Delta \in \mathbb{Z}_q$ to form a credential $c' = g_0^{r'} g_1^{sk} g_2^{S'} g_3^{balance'}$ and $comm_\Delta = g_0^{r_\Delta} g_1^\Delta$. Compute $A = \text{Accumulate}(params, \mathbf{C}_O)$ and $\omega = \text{GenWitness}(params, c, \mathbf{C}_O)$. Output $(c', sk'_c, comm_\Delta, r_\Delta, \pi_{rd})$ where $sk'_c = (S', balance, r')$ and

$$\begin{aligned} \pi_{rd} &= \text{NIZKPoK}\{(sk, c, c', \omega, balance, balance', \Delta, r, r', r_\Delta) : \\ &\quad \text{AccVerify}(params, A, c, \omega) = 1 \\ &\quad \wedge c = g_0^{r'} g_1^{sk} g_2^{S'} g_3^{balance} \wedge c' = g_0^{r'} g_1^{sk} g_2^{S'} g_3^{balance'} \\ &\quad \wedge comm_\Delta = g_0^{r_\Delta} g_1^\Delta \wedge balance' = balance - \Delta\} \end{aligned}$$
- **Replnc**($params, sk, c, sk_c, comm_\Delta, \Delta, r_\Delta, \mathbf{C}_O$) $\rightarrow (c', sk'_c, \pi_{re})$. Given a credential c and associated secret key sk_c , an amount to increment the reputation $\Delta \in \mathbb{Z}_q$, a commitment to the reputation change $comm_\Delta$ and its opening r_Δ , and a set of credentials $\mathbf{C}_O$, calculate the new balance $balance' = balance + \Delta$. Generate a fresh random serial number $S' \in \mathbb{Z}_q$ and random value $r' \in \mathbb{Z}_q$ to form a fresh credential $c' = g_0^{r'} g_1^{sk} g_2^{S'} g_3^{balance'}$ encoding the new balance. Compute $A = \text{Accumulate}(params, \mathbf{C}_O)$ and $\omega = \text{GenWitness}(params, c, \mathbf{C}_O)$. Output (c', sk'_c, π_{re}) where $sk'_c = (S', balance, r')$ and

$$\begin{aligned} \pi_{re} &= \text{NIZKPoK}\{(sk, c, c', \omega, balance, balance', \Delta, r, r', r_\Delta) : \\ &\quad \text{AccVerify}(params, A, c, \omega) = 1 \\ &\quad \wedge c = g_0^{r'} g_1^{sk} g_2^{S'} g_3^{balance} \wedge c' = g_0^{r'} g_1^{sk} g_2^{S'} g_3^{balance'} \\ &\quad \wedge comm_\Delta = g_0^{r_\Delta} g_1^\Delta \wedge balance' = balance + \Delta\} \end{aligned}$$
- **RepVerify**($c_A, c_B, \pi_{rd}, \pi_{re}, \mathbf{C}_O$) $\rightarrow \{0, 1\}$. Given credentials c_A and c_B , proofs π_{rd} and π_{re} , and a set of credentials $\mathbf{C}_O$, verify a reputation transaction between two parties by validating the proofs, checking that they use the same $comm_\Delta$, and checking that the revealed serial numbers were not previously used. Output 1 if all conditions hold, else 0.

Figure 4: Extensions for an anonymous reputation system.

a reputation system as simply an instance of an updatable credential system where the encoded state is a user's reputation. Updates affect a pair of stateful credentials, and we restrict state transitions to ones that conserve the total amount of reputation in the credential pair. In fact, we can realize more complex relations subject to that constraint, but for simplicity we limit our discussion here to the simple conservation of reputation.⁹ This system even allows for negative reputation, a notable feature which previously required a continually available online bank [46].

We define three additional functions in Figure 4 that allow users to transfer reputation and verify the transfer. Using **RepDec** Alice would decrement her reputation by the amount she wants to send and broadcast the resulting proof π_{rd} for insertion into the block chain. She would then send $comm_\Delta$ and its opening values (Δ, r_Δ) to Bob. Bob would run **Replnc** and also broadcast the resulting proof. Alice can "give" Bob negative reputation by forcing him to decrement his balance by some amount prior to an encounter and then giving back some (or none) of it after the encounter is completed. We neglect pseudonyms in the above definition for the sake of exposition, but they can be added in the usual way.

We leave a proof of security for the full version of this work but provide a short intuition here: provided that our updatable anonymous credential scheme is secure then so is this scheme since clearly no user can increase his reputation without another depleting hers. Anonymity, however,

is not as clear. Bob must know the transaction in which Alice decrements her reputation. Clearly this leaks some information. However, due to the anonymity property of our credential construction, the entire mint/update process cannot be linked to a previous or subsequent show. As such, no one learns anything that is linkable to any other transaction. Because of this unlinkability, we obtain an identity-bound reputation system where users do not need to discard their reputation when switching pseudonyms. Similarly, inflation resistance follows from the fact that we can specify arbitrary rules of accepting credentials into the block chain in the first place. As such we can limit how reputation is added to the network by limiting the rate at which reputation points are added.

8. RELATED WORK

Anonymous credentials. Introduced by Chaum [19] and developed in a line of subsequent works (e.g., [8, 11, 14]), anonymous credentials allow a user to prove that she has a credential issued by some organization, without revealing anything about herself other than that she has the credential. Under standard security definitions, even if the verifier and credential issuer collude, they cannot determine when the credential was issued, who it was issued to, or when it was or will be used. A common construction involves issuing a credential by obtaining a signature from an organization on a committed value (e.g., using the signature scheme of [12]) then proving in zero-knowledge that one has a signature under the organization's public key on that value. The contents of the commitment may be revealed outright or various

⁹Our policies are limited to the relations over committed values that can be proved in zero-knowledge. If we restrict ourselves to efficient proofs, we can realize a broad set of polynomial relations over the reputation of both parties.

properties can be proved on the committed values (e.g., Alice can prove she is over 21 years old). Extensions to this work describe credentials that can only be shown anonymously a limited number of times [9] or delegated to others [4]. All of these schemes require issuing organizations to maintain a secret key.

Bitcoin and append-only ledgers. Our construction relies on the existence of a *distributed append-only transaction ledger*, a technology that makes up the core component of the Bitcoin distributed currency: the log of all currency transactions called the block chain [36]. These ledgers are maintained by an ad hoc group of network nodes who are free to enter and leave the network (there is no key provisioning necessary for them to join). A typical transaction ledger consists of a sequence of blocks of data that are widely replicated among the participating nodes, with each block connected to the previous block using a hash chain. Nodes compete for the opportunity to add new blocks of transactions to the ledger by producing a partial hash collision over the new data and the hash of the last block in the chain. The hash collision serves two purposes: first, it is a computationally difficult to forge authenticator of the ledger and second, since finding a partial hash collision involves substantial computational effort, the peer who finds it is chosen “at random” with a probability proportional to the rate at which they can compute identify such partial collisions. As a result, an ad hoc group of mutually distrusting and potentially dishonest peers can correctly manage such a ledger provided that a majority of their computational power is held by honest parties. Recent experience with Bitcoin and Namecoin provides evidence that this assumption holds in practice.

Namecoin. Namecoin [37] is a decentralized identity system that uses the same block chain technology as Bitcoin. Although Namecoin is in part a currency system — its internal currency, the namecoin (NMC), is used to purchase a name in the same way one pays for a domain name — its primary purpose is to associate names with arbitrary data. A user can claim a name provided (1) they pay the price in NMC for it and (2) it is unclaimed. At that point, an entry is inserted into the block chain mapping the name to a public key and some arbitrary data. The public key allows the owner to update the data by signing a new record. The data allows for various uses. If it is an IP address, then one has a distributed DNS system (such a system, .bit, is already deployed). On the other hand, if it is a public key, the result is a basic PKI. The first-come first-served nature of Namecoin seems somewhat anachronistic, however it replicates in miniature the way normal DNS names are generally assigned, where the first person to claim the name gets it. Similarly, standard (non-extended validation) SSL certificates for a domain are typically issued to anyone who can demonstrate control of a domain (usually via an email to admin@domain).

9. CONCLUSION

In this work we constructed a distributed anonymous credential system and extended this system to provide a distributed, identity-bound reputation system where users have a single global reputation. Our constructions are secure in the random oracle model under standard cryptographic assumptions assuming the existence of a trustworthy global append-only ledger. To realize such a ledger we propose using the block chain system already in real world use with

the distributed cryptographic currency Bitcoin. Although we are limited in the class of identity assertions we can certify, we argue that several basic assertions are of particular use in peer-to-peer systems, as they can be used to mitigate Sybil attacks, ensure fair resource usage, and punish malicious peers while maintaining user anonymity.

Future work. We leave two open problems for future work. First, the proofs in this work assumed the security of a transaction ledger. We leave a precise formal model of the ledger, which attacks are allowable, and what bounds may be placed on their consequence as an open problem. Second, the efficiency of our construction can be improved. Although all of our algorithms are efficient (in that they do not scale with the size of the ledger), the need for double-discrete logarithm proofs leads to somewhat large proof sizes when showing a credential (roughly 40KB for modest parameters). Our construction may be optimized for certain applications that do not require the full flexibility of our construction. At the same time, we hope that advances in bilinear accumulators, mercurial commitments, or lattice based techniques may provide a more efficient construction.

Acknowledgments

Christina Garman and Ian Miers are supported in part by NSF CNS-1010928 and HHS 90TR0003/01.

Matthew Green is supported in part by the Defense Advanced Research Projects Agency (DARPA) and the Air Force Research Laboratory (AFRL) under contract FA8750-11-2-0211, the Office of Naval Research contract N00014-11-1-0470, NSF grant CNS-1010928 and HHS 90TR0003/01.

Applying to all authors, the contents and views expressed are solely those of the authors and do not reflect the official policy, position or views of the Department of Defense, the HHS or the U.S. Government.

10. REFERENCES

- [1] E. Androulaki, S. G. Choi, S. M. Bellovin, and T. Malkin. Reputation systems for anonymous networks. *Privacy Enhancing Technologies*. 2008.
- [2] G. Ateniese, D. Song, and G. Tsudik. Quasi-efficient revocation of group signatures. In *Financial Cryptography*, 2003.
- [3] N. Barić and B. Pfitzmann. Collision-free accumulators and fail-stop signature schemes without trees. In *EUROCRYPT*, 1997.
- [4] M. Belenkiy, M. Chase, M. Kohlweiss, and A. Lysyanskaya. P-signatures and noninteractive anonymous credentials. In *Theory of Cryptography*. 2008.
- [5] J. Benaloh and M. de Mare. One-way accumulators: a decentralized alternative to digital signatures. In *EUROCRYPT*, 1994.
- [6] R. Bhattacharjee and A. Goel. Avoiding ballot stuffing in ebay-like reputation systems. *P2PECON '05*, 2005.
- [7] S. Brands. Rapid demonstration of linear relations connected by boolean operators. In *EUROCRYPT*, 1997.
- [8] S. A. Brands. *Rethinking public key infrastructures and digital certificates: building in privacy*. The MIT Press, 2000.

- [9] J. Camenisch, S. Hohenberger, M. Kohlweiss, A. Lysyanskaya, and M. Meyerovich. How to win the clonewars: efficient periodic n-times anonymous authentication. CCS, 2006.
- [10] J. Camenisch, M. Kohlweiss, and C. Soriente. An accumulator based on bilinear maps and efficient revocation for anonymous credentials. In *Public Key Cryptography*, 2009.
- [11] J. Camenisch and A. Lysyanskaya. An efficient system for non-transferable anonymous credentials with optional anonymity revocation. EUROCRYPT, 2001.
- [12] J. Camenisch and A. Lysyanskaya. Dynamic accumulators and application to efficient revocation of anonymous credentials. In *CRYPTO*, 2002.
- [13] J. Camenisch and A. Lysyanskaya. Dynamic accumulators and application to efficient revocation of anonymous credentials. In *CRYPTO*, 2002. Extended Abstract.
- [14] J. Camenisch and A. Lysyanskaya. A signature scheme with efficient protocols. SCN'02, 2003.
- [15] J. Camenisch and M. Michels. Proving in zero-knowledge that a number n is the product of two safe primes. In *EUROCRYPT*, 1999.
- [16] J. Camenisch and M. Stadler. Efficient group signature schemes for large groups. In *CRYPTO*, 1997.
- [17] J. L. Camenisch. *Group Signature Schemes and Payment Systems Based on the Discrete Logarithm Problem*. PhD thesis, ETH Zürich, 1998.
- [18] M. Chase and A. Lysyanskaya. On signatures of knowledge. In *CRYPTO*, volume 4117 of LNCS, pages 78–96, 2006.
- [19] D. Chaum. Security without identification: transaction systems to make big brother obsolete. *Communications of the ACM*, 1985.
- [20] S. Coull, M. Green, and S. Hohenberger. Access controls for oblivious and anonymous systems. In *TISSEC*, 2011.
- [21] R. Cramer, I. Damgård, and B. Schoenmakers. Proofs of partial knowledge and simplified design of witness hiding protocols. In *CRYPTO*, 1994.
- [22] I. Damgård, K. Dupont, and M. Ø. Pedersen. Unclonable group identification. EUROCRYPT, 2006.
- [23] W. Diffie and M. Hellman. New directions in cryptography. *IEEE Transactions on Information Theory*, 1976.
- [24] Y. Dodis and A. Yampolskiy. A verifiable random function with short proofs and keys. PKC, 2005.
- [25] Dot-bit. Available at <http://dot-bit.org/>.
- [26] J. R. Douceur. The sybil attack. In *Peer-to-Peer Systems*. 2002.
- [27] A. Fiat and A. Shamir. How to prove yourself: Practical solutions to identification and signature problems. In *CRYPTO*, 1986.
- [28] E. Fujisaki and T. Okamoto. Statistical zero knowledge protocols to prove modular polynomial relations. In *CRYPTO*, 1997.
- [29] O. Goldreich, S. Goldwasser, and S. Micali. How to construct random functions. *Journal of the ACM*, 1986.
- [30] O. Goldreich, S. Micali, and A. Wigderson. Proofs that yield nothing but their validity and a methodology of cryptographic protocol design. In *FOCS*, 1986.
- [31] O. Hasan, L. Brunie, and E. Bertino. Preserving privacy of feedback providers in decentralized reputation systems. *Computers & Security*, 2012.
- [32] S. D. Kamvar, M. T. Schlosser, and H. Garcia-Molina. The eigentrust algorithm for reputation management in P2P networks. WWW '03, 2003.
- [33] M. Kinateder and K. Rothermel. Architecture and algorithms for a distributed reputation system. In *Trust Management*. 2003.
- [34] M. Z. Lee, A. M. Dunn, B. Waters, E. Witchel, and J. Katz. Anon-pass: Practical anonymous subscriptions. In *IEEE Security and Privacy*, 2013.
- [35] I. Miers, C. Garman, M. Green, and A. Rubin. Zerocoin: Anonymous distributed e-cash from bitcoin. In *IEEE Security and Privacy*, 2013.
- [36] S. Nakamoto. Bitcoin: A peer-to-peer electronic cash system. 2008.
- [37] Namecoin. Available at <http://namecoin.info/>.
- [38] M. Naor and O. Reingold. Number-theoretic constructions of efficient pseudo-random functions. *Journal of the ACM (JACM)*, 2004.
- [39] T.-W. Ngan, D. S. Wallach, and P. Druschel. Enforcing fair sharing of peer-to-peer resources. In *Peer-to-Peer Systems II*. 2003.
- [40] D. Obenshain, T. Tantillo, A. Newell, C. Nita-Rotaru, and Y. Amir. Intrusion-tolerant cloud monitoring and control. In *LADIS*. 2012.
- [41] T. Pedersen. Non-interactive and information-theoretic secure verifiable secret sharing. In *CRYPTO*, 1991.
- [42] P. Resnick, K. Kuwabara, R. Zeckhauser, and E. Friedman. Reputation systems. *Commun. ACM*, 2000.
- [43] R. L. Rivest, A. Shamir, and Y. Tauman. How to leak a secret. In *ASIACRYPT*. 2001.
- [44] T. Sander. Efficient accumulators without trapdoor extended abstract. In *Information and Communication Security*, volume 1726 of LNCS, pages 252–262, 1999.
- [45] T. Sander and A. Ta-Shma. Auditable, anonymous electronic cash (extended abstract). In *CRYPTO*, 1999.
- [46] S. Schiffner, S. Clauss, and S. Steinbrecher. Privacy and liveness for reputation systems. In *Public Key Infrastructures, Services and Applications*. Springer Berlin Heidelberg, 2010.
- [47] C.-P. Schnorr. Efficient signature generation for smart cards. *Journal of Cryptology*, 4(3):239–252, 1991.
- [48] Q. Wang, Z. Lin, N. Borisov, and N. J. Hopper. rBridge: user reputation based tor bridge distribution with privacy preservation. 2009.
- [49] B. Zhu, S. Setia, and S. Jajodia. Providing witness anonymity in peer-to-peer systems. CCS, 2006.

APPENDIX

A. PROOF OF SECURITY FOR OUR BASIC SYSTEM

We now provide a sketch of the proof of security for our basic distributed anonymous credentials system.

Our basic approach is to show that for every real-world adversary $\mathcal{A}$ against the credential system, we can construct an ideal-world adversary $\mathcal{S}$ against the ideal-world system such that the transcript of $\mathcal{A}$ interacting with the real sys-

tem is computationally indistinguishable from the transcript produced by $\mathcal{A}$ interacting with $\mathcal{S}$. We assume a static corruption model in which the adversary controls some set of users and leave a proof in the adaptive corruption model for future work. We also assume that our zero-knowledge signatures of knowledge include an efficient extractor and simulator and that the *params* are created using a trusted setup process.

Our proof assumes the existence of a global, trusted transaction ledger, which we use as a black box. We leave a complete proof that considers this construction and models it to future work.

We begin by sketching the simulator $\mathcal{S}$ for our system.

A.1 Description of the Simulator

Minting a credential. When a user controlled by the adversary with nym Nym_U^O wants a credential, the user first generates $(c, \pi_M, attrs)$. When the simulator receives notification of this, it first verifies that the credential and proof are valid and meet the organization's policy. If so it employs the knowledge extractor for the signature of knowledge on π_M to get (sk, aux) .

The simulator then checks if it has a record of (U, sk, Nym_U^O) on its list of users. If the user with key sk and nym Nym_U^O exists, then $\mathcal{S}$ retrieves sk_U associated with (U, sk, Nym_U^O) and proceeds. If it is not on the list, the simulator checks if it has previously seen a user with key sk . If the user with key sk is not present, then the simulator creates a user U and runs $RegNym(Nym_U^O, U, O)$ to register Nym_U^O and obtain sk_U for further interactions with T_P . $\mathcal{S}$ then stores (U, sk, sk_U, Nym_U^O) in its list of users controlled by the adversary. If a user U with key sk exists, then it runs $RegNym(Nym_U^O, U, O)$ to register Nym_U^O and adds Nym_U^O to U 's record.

Once the simulator has registered the nym or verified it already exists, it runs $MintCred(Nym_U^O, O, attrs, aux)$. The simulator then transmits the credential information to the trusted store and acknowledges the credential's issuance. $\mathcal{S}$ stores $(sk, Nym_U^O, attrs, aux, c, \pi_M)$ in its list of granted credentials.

When an honest user, through T_P , wants to establish a credential, the simulator creates a credential c (using the publicly available *attrs*) and uses the simulator for the signature of knowledge π_M to simulate the associated proof. It then transmits the credential information $(c, \pi_M, attrs)$ to the trusted store.

Showing a credential. When a user controlled by the adversary wants to show a credential from organization O to verifier V with which it has nym Nym_U^O and Nym_U^V respectively, the user first generates π_S . When the simulator receives notification of this, it verifies the proof as in the real protocol (rejecting if it is invalid). If the show verifies, it runs the knowledge extractor for the proof of knowledge on π_S to get sk .

The simulator then checks if it has a record of $(U, sk, Nym_U^O, Nym_U^V)$ on its list of users. If the user with key sk and nym Nym_U^O and Nym_U^V exists, then $\mathcal{S}$ retrieves sk_U associated with (U, sk, Nym_U^O) and proceeds. If the record does not exist, either in part or in full, the simulator checks if it has previously seen a user with key sk . If the user with key sk is not present, then the simulator creates a user U and runs $RegNym(Nym_U^O, U, O)$ and $RegNym(Nym_U^V, U, V)$

to register Nym_U^O and Nym_U^V and obtain sk_U for further interactions with T_P . $\mathcal{S}$ then stores $(U, sk, sk_U, Nym_U^O, Nym_U^V)$ in its list of users controlled by the adversary. If a user U with key sk exists, then it runs $RegNym(Nym_U^O, U, O)$ (resp. $RegNym(Nym_U^V, U, V)$) to register Nym_U^O (resp. Nym_U^V) and adds Nym_U^O (resp. Nym_U^V) to U 's record.

Now, the simulator $\mathcal{S}$ runs $ShowOnNym(Nym_U^O, Nym_U^V, O, V, \mathbf{C})$ where $\mathbf{C}$ is obtained by the simulator through a call to $GetCredList(O)$.

When an honest user (through T_P) wants to show a credential to a verifier V controlled by the adversary, the simulator runs the zero-knowledge simulator for π_S to simulate a proof that it then sends to V .

A.1.1 Proof (sketch) of a Successful Simulation

Our basic distributed anonymous credentials system is secure under the Strong RSA and the Discrete Logarithm assumptions if the view of the adversary in the real protocol is computationally indistinguishable from his view in the simulation. This means that the simulator must only fail with negligible probability.

We first begin by discussing the signatures/proofs π_M and π_S . Under the Discrete Logarithm assumption, π_M is a computational zero-knowledge signature of knowledge on *aux* of the values sk , r , and r' such that the nym Nym_U^O and the credential c both belong to the same master secret sk . The proof is clearly zero-knowledge because it can be constructed using standard techniques [47]. One can see that the only way to forge this proof would be to cause a collision on the commitments, which occurs with negligible probability under the Discrete Logarithm assumption [41]. Under the Strong RSA and Discrete Logarithm assumptions, π_S is a statistical non-interactive zero-knowledge proof of knowledge of the values sk , ω , c , Nym_U^V , r , and r' such that ω is a witness that c is in the accumulator A and nym Nym_U^V and the credential c both belong to the same master secret sk . We can again see that this proof can be constructed using known techniques [47, 12] similar to the proofs used by Miers et al. in [35]. In order to forge such a proof, the adversary would need to either find a collision on the commitments or forge an accumulator membership witness. We previously discussed how the first case occurs with negligible probability. The second case occurs with negligible probability under the Strong RSA assumption due to [12]. See the full version of the paper for a formal treatment/reduction of these statements.

Intuitively, we can now see that the simulator will fail negligibly because it deals solely with zero-knowledge signatures of knowledge and zero-knowledge proofs of knowledge, which have efficient extractors and simulators. Our proofs π_M and π_S have knowledge extractors that succeed with probability $1 - \nu(\lambda)$ for some negligible function $\nu(\cdot)$. Since signatures and proofs are the sole point of failure for our simulator described above, it fails with negligible probability. Because the only thing the adversary sees is the zero-knowledge proofs and signatures, the simulated signatures and proofs are computationally indistinguishable from legitimate ones, and the adversary cannot cause our simulator to fail except with negligible probability, the adversary cannot distinguish between an interaction with the simulator and the real protocol.

SCORAM: Oblivious RAM for Secure Computation*

Xiao Shaun Wang¹, Yan Huang², T-H. Hubert Chan³, abhi shelat⁴ and Elaine Shi¹

¹UMD ²IU Bloomington

¹²{wangxiao, elaine, yanhuang}@cs.umd.edu

³HKU

³hubert@cs.hku.hk

⁴UVa

⁴abhi@virginia.edu

ABSTRACT

Oblivious RAM (ORAM) data structures have traditionally been measured by their *bandwidth overhead* and *client storage*. We observe that when using ORAM structures to build secure computation protocols for RAM programs, the *size* and *depth* of the circuit that implement the ORAM operations is a more relevant measure of performance. For example, we note that, in the context of secure computation, an ORAM design with an asymptotic bandwidth complexity of $O(\log n)$ can be easily out-performed by an “inferior” scheme due to its complicated read/write algorithm.

We therefore embark on a study of the *circuit-complexity* of several recently proposed ORAM constructions. Through careful implementations and experiments, we show that asymptotic analysis is not indicative of true performance of ORAMs when they are used in a secure computation protocol for RAM programs with realistic memory sizes.

We then present SCORAM, a heuristic *compact* ORAM design optimized for use in secure computation protocols. Our new design is almost 10x smaller and also faster than all other designs we have tested for realistic memories of size 4MB–2GB and for failure probabilities of 2^{-80} . This new SCORAM makes it feasible to perform more secure computations on gigabyte-sized data sets.

1. INTRODUCTION

Two-party secure computation refers to a cryptographic technique that allow Alice, who holds private input x and Bob, who holds private input y , to jointly compute $f(x, y)$ without revealing any information to each other aside from the output $f(x, y)$. All known efficient constructions of such generic cryptographic protocols require an *oblivious* representation of the function f being evaluated to ensure that the control flow of the algorithm does not depend on its input and therefore leak partial information. The standard approach to creating such an oblivious representation is to gen-

erate a binary circuit from the description of f . This is the strategy chosen by dozens of prior works [?, ?, ?, ?, ?, ?, ?, ?] on secure computation.

A well-known drawback to the approach of creating a boolean circuit for f is that the size of the circuit for f —and thus the running time of the secure computation protocol for Alice and Bob—relates to the worst-case running time and space complexity of f . This circuit blow-up is especially problematic in the case when the most efficient implementation of f is in the RAM model of computation in which programs can access any cell of its working memory in $O(1)$ time (the RAM model represents how programs typically run on modern processors). In contrast, a circuit requires $O(s)$ time to access a particular element of its “memory” where s is the (worst-case) space complexity of the machine that implements f .

To avoid this circuit overhead, Ostrovsky and Shoup [?] considered whether RAM programs could be directly implemented as secure computation protocols without having to transform them to circuits. Unfortunately, the memory locations that are accessed by a program often *leak* intermediate values of the program and thereby contradict the security guarantees of secure computation. To overcome this issue, Ostrovsky and Shoup use an Oblivious RAM (ORAM) data structure proposed first by Goldreich [?] to compile RAM programs into secure computation protocols.

Intuitively, an ORAM is a method for compiling a RAM program into another RAM program whose memory access pattern does not depend on any input to the program (thus, the memory access pattern is said to be *oblivious*). ORAM techniques have been widely studied in other contexts [?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?]. The goals of these prior works have been to (a) reduce the bandwidth overhead of the ORAM (b) reduce the “client storage” of the ORAM, and (c) reduce the server’s overall memory overhead. Remarkably, state of the art approaches to ORAM design limit the overhead in all three aspects to various combinations of $O(\log^c(n))$ for $c \in \{0, 1, 2, 3\}$!

Towards large-scale secure computation. At present, secure computation is limited to small problems because large datasets incur very-large overheads in the circuit-model. Secure computation in the RAM model offers an asymptotically efficient way to circumvent these problems and can potentially scale secure computation to large datasets. Indeed, the development of practical ORAM techniques and demonstrations of practical ORAM implementations [?] have lead to recent notable works that demonstrate how ORAM-based secure computation protocols enable dramatic efficiency im-

*The research was supported in part by a grant from Hong Kong RGC under the contract HKU719312E.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
Copyright 20XX ACM X-XXXXX-XX-X/XX/XX ...\$15.00.

ORAM	Circuit Size (Asymptotic Bounds)	Circuit Size (gates)		Number of Inputs	
		$N = 2^{20}$	$N = 2^{29}$	$N = 2^{20}$	$N = 2^{29}$
Linear Scan ORAM	$O(DN)$	142.6M	$\approx 82678.1\text{M}$	33.5M	17180.0M
LO ORAM	$O(D \log N)\omega(1)$				
CLP ORAM	$\tilde{O}(\log^5 N + D \log^3 N)\omega(1)$	29.4M	121.8M	0.6M	2.1M
Binary Tree ORAM	$O(\log^4 N + D \log^2 N)\omega(1)$	38.5M	127.7M	7.0M	24.2M
naive PATH ORAM	$O(\log^4 N + D \log^2 N)\omega(1)$	87.3M	278.1M	0.1M	0.3M
PATH SCORAM	$\tilde{O}(\log^3 N + D \log N)\omega(1)$	37.2M	111.7M	0.1M	0.3M
SCORAM	N/A (heuristic)	4.6M	13.0M	0.3M	0.9M

Table 1: **Performance metrics for different ORAM schemes when used in Secure Computation.** N is the number of blocks in the ORAM, D is number of bits in each block (i.e., the payload bit length), and the security parameter is set to be $O(\log N)\omega(1)$. We use the notation $g(N) = O(f(N))\omega(1)$ to denote that for any $\alpha(N) = \omega(1)$, it holds that $g(N) = O(f(N)\alpha(N))$. The notation $\tilde{O}$ hides $\log \log N$ factors. All concrete measurements are for payload bit-length $D = 32$ bits with probability of SCORAM failure set to 2^{-80} and include all recursive instances of the ORAM to store the position map. The circuit size reports all gates, and the number of inputs defines cost of input preprocessing, e.g., OT.

improvements in processing large, secret datasets. For example, the work of Gordon *et al.* [?] shows how repetitive binary search queries can be securely computed in amortized sublinear time. Liu *et al.* [?] show how, under big data sizes, RAM-model secure computation significantly outperforms the circuit model even for *run-once* tasks, including KMP-string matching and shortest path. This work extends this vision by tackling the problem of finding very efficient ORAMs to enable secure computation on giga-byte sized datasets.

1.1 Contribution

We embark on a comprehensive study of practical secure computation ORAM techniques. We do so via both theoretical analysis of several metrics used to judge ORAMs and several experiments performed on optimized implementations of 4 state-of-the-art ORAM designs. We report on a new heuristic ORAM design that outperforms all other ORAMs we considered.

Our first observation is that the traditional measures of an ORAM scheme do not properly predict ORAM performance. ORAM has been considered for two primary application scenarios: the privacy-preserving cloud outsourcing scenario [?, ?, ?], and the secure processor application [?, ?]. In these settings, an ORAM’s overhead is measured by its “bandwidth overhead”, i.e., how many data units the ORAM must retrieve to serve a single memory query¹.

We observe that the bandwidth overhead metric is inadequate at characterizing the performance of an ORAM in a secure computation application. Instead, the *circuit complexity* of the ORAM algorithm—a measure that is traditionally ignored by the literature—plays a critical role in the efficiency ORAM-based secure computation. Although this paper focuses on using ORAM in a garbled circuits approach to secure computation, we note that the metrics we put forth are meaningful when SCORAM is used in other secure computation techniques as well.

Re-evaluate and improve existing ORAMs. We then

¹The bandwidth overhead metric is equivalent to the blowup of the runtime of the Oblivious RAM in comparison with the non-oblivious RAM machine – this was the terminology used originally by Goldreich and Ostrovsky [?].

conduct a systematic evaluation of several state-of-the-art ORAM schemes (e.g., Binary Tree ORAM [?], Path ORAM [?], CLP ORAM [?]) when applied to the secure computation setting. Table 1 reports both the theoretical and concrete complexity of these existing ORAM schemes. Our concrete parameters are derived from carefully optimized implementations of these algorithms.

In particular, we observe that the basic tree ORAM by Shi *et al.* [?] has the same asymptotic efficiency as the Path ORAM [?] (when implemented with a naive circuit), but the latter has a much larger concrete circuit when implemented for reasonable parameters. We therefore propose PATH SCORAM, a new circuit construction for Path ORAM, achieving better asymptotic overhead than its naive counterpart. However, this asymptotically efficient PATH SCORAM requires 3 oblivious sorts during the circuit construction, which incurs a large constant in practice. As a result, we observe that its empirical performance is not a clear win over asymptotically worse schemes such as the simple Binary Tree ORAM [?] or the CLP ORAM [?].

A new ORAM scheme. Based on the experience gained, we shift the focus: instead of aiming for an *asymptotically* more efficient ORAM, we aim to construct a scheme that is *empirically* the most efficient. We argue that within $\text{poly } \log N$ complexity ranges, optimizing for asymptotics can be misguided. For conceivable data sizes ranging from gigabytes to terabytes, $\log N$ is typically 20 to 40, and can be easily overwhelmed by even moderate constants (e.g. 100) that may be incurred by asymptotically superior schemes. Similar observations of asymptotics vs. practical performance are widespread, e.g. by Stefanov *et al.* for constructing small-domain PRPs [?].

Our result is a heuristic ORAM scheme, SCORAM, that is almost **10X** smaller and also faster than any other approach we are aware of. Our implementation of SCORAM will be made available online for people to build efficient ORAM-based secure computation protocols². Our SCORAM scheme and implementation are also an important

²The URL is not included here to preserve anonymity in the submission. We will make our code available upon reviewers’ request.

building block for realizing efficient oblivious data structures [?] in secure computation.

1.2 Related Work

Keller and Scholl [?] implemented secure oblivious data structures using both the SCSL ORAM [?] and the Path ORAM [?]. Their implementations use a small security parameter 2^{-20} and works in a pre-processing model; in contrast, we evaluate at security parameter 2^{-60} to be more realistic. At least two of our implementations achieve better efficiency parameters while also enjoying provable error bounds.

Gordon *et al.* [?] study the feasibility of constructing sublinear-time secure computation protocols for functions that can be computed in sublinear time on a random-access machine (RAM). They show generically how any function $f(\cdot, \cdot)$ that can be computed in time t and space s in the RAM model can be securely computed by a protocol that requires amortized time $O(t)\text{poly}(\log(s))$, requires one party to use space $O(\log(s))$ and requires the other party to use space $O(\text{spoly}(\log(s)))$. Note, their goal is to ensure that one party in the protocol requires very little space complexity. They report on an implementation of a binary-tree based SCORAM; using security parameter 2^{-13} , and $N = 2^{20}$, their ORAM scheme requires roughly 6.6M non-free gates and 12M total gates. To the best of our ability, we attempted to recreate their parameters: our implementation of the same required 3.3M non-free gates and 16m total gates. The implementation of our best scheme at a much higher security parameter is approximately 3 times smaller. See §6 for details.

Gentry *et al.* [?] optimize the binary tree ORAM for secure computation. We did not include the scheme in Table 1 because their scheme is subsumed both asymptotically and empirically by Path ORAM [?]. This can be observed with a simple back-of-the-envelope calculation as follows: Gentry *et al.* propose techniques to reduce the tree height by enlarging the bucket size. They then proposed a new eviction algorithm similar to Path ORAM’s eviction, except that they directly compute where a block should be dropped along the path—closest to leaf possible respecting invariant—and then drop it in that bucket. Naively implementing their eviction would result in $O(A^2)$ overhead where A is the total number of blocks on the path, i.e., $A = (\text{bucket size}) \cdot (\text{path length})$. Notice that Path ORAM also has $O(A^2)$ overhead for a naive eviction circuit, but Path ORAM’s A value is both asymptotically and empirically smaller than that of Gentry *et al.* In both schemes, we can have an asymptotically smaller eviction circuit with oblivious sort, but as we show, oblivious sort introduces such a large constant, that the practical performance is worse than that of the original binary-tree ORAM.

As far as we know, Lu and Ostrovsky [?] report the asymptotically best ORAM in the literature. Their 2-server design can perform a sequence of n reads or writes with $O(\log n)$ amortized overhead per access while using $O(n)$ storage for the servers and $O(1)$ client memory. These parameters meet the lower-bound for performance of a single-server ORAM and are superior to any known single-server ORAM. Thus, it appears to be a perfect candidate for ORAM in secure computation. Unfortunately, there are two serious practical bottlenecks to the implementation of LO.

To understand the first issue, recall that the LO ORAM

builds upon the KLO ORAM [?] which further builds upon³ the map-reduce based cuckoo-hashing based ORAM of Goodrich and Mitzenmacher [?]. In order to read or write at index x , the scheme iteratively queries a hierarchy of hash tables $H_k, H_{k+1}, \dots, H_L$ with either x or a dummy address t depending on whether x has been found or not. The security relies on the issuing of this dummy query once x has been found in order to maintain the invariant that every lookup is unique (i.e., there is never a lookup for the same address x at any two levels in the hierarchy). This means that read queries require *several sequential* executions of separate secure computation protocols; in contrast, tree-based ORAM designs require only 1 secure computation protocol to be run per read/write operation.

The second serious problem is a large overhead constant for cuckoo hashing. All cuckoo-hashing based ORAMs depend on an lemma proven by Goodrich and Mitzenmacher [?] which bounds the collision rate of a cuckoo-hashing structure via a combinatorial analysis of a graph G that is based on the cuckoo-hashing function. This lemma requires the cuckoo hash table size to be $m = \Omega(\log^7(N))$; this technique therefore only starts to beat the Linear scan SCORAM when $N > 2^{-37}$.

2. BACKGROUND: TREE-BASED ORAMS

Notation. We use N to denote the number of (real) data blocks in ORAM, D to denote the bit-length of a block in ORAM, Z to denote the capacity of each bucket in the ORAM tree, and λ to denote the ORAM’s statistical security parameter. When discussing binary trees of depth L in this paper, we say the *leaves* are at level 0 and the root is at level $L - 1$. Although unconventional, this simplifies the description of our algorithms.

2.1 Tree-based ORAM Construction

Shi *et al.* [?] proposed a new binary-tree based framework for constructing a class of ORAM schemes. Many recent efficient ORAM schemes [?, ?, ?] extend this construction, so we briefly review this framework below. The key difference between the schemes is the choice of *eviction* strategy.

Data organization. The server organizes N blocks into a binary tree of height $L = \log N$; each node of the tree is a *bucket* containing Z blocks. Each block is of the form:

$$\{\text{idx} || \text{label} || \text{data}\},$$

where **idx** is the index of a block, e.g., the (logical) address of desired block; **label** is a leaf identifier specifying the path on which the block resides; and **data** is the payload of the block.

The client stores a *position map*, mapping memory addresses to *leaf labels*. Position map storage can be reduced to $O(1)$ by recursively storing the position map in a smaller ORAM (see [?] for details). These leaf labels are assigned randomly and are reassigned as blocks are accessed. If we label the leaves from 0 to $N - 1$, then each label is associated with a path from the root to the corresponding leaf. Tree-based ORAMs maintain the *invariant* that a block marked **label** resides either on the path leading to the corresponding leaf node specified by **label**; or resides in the client’s stash.

³KLO point out a subtle security issue that affects almost all cuckoo-hashing based ORAM schemes, and explain how to fix it.

Operations. Tree-based ORAM have three main operations. Among these, the Eviction algorithm is the key difference between schemes.

- **ReadAndRemove:** Given an index idx , the client looks up the its label from the position map, and fetches all blocks on the path leading to label . The client finds the block idx (due to the main invariant) and removes it from the path.
- **Add:** The retrieved block is potentially updated, reencrypted and written back to the root bucket.
- **Eviction:** Percolate blocks towards leaves such that no bucket will overflow except with negligible probability. Various ORAMs use different eviction schemes which we explain below.

Recursion. Instead of storing the entire position map in the client’s local memory, the client can store it in a smaller ORAM on the server. In particular, this position map ORAM needs to store $N \log(N)$ -bit labels. By storing χ labels in one block, this ORAM only needs N/χ blocks. Finally, by applying recursion, the position map can be reduced to $O(1)$ size, after which the Linear scan ORAM can be used. See §5 for a discussion of how we set the recursion parameters. Unless otherwise noted, our discussion of the complexity of the eviction strategy applies to a single ORAM (i.e., not taking into consideration the recursion, which applies equally to all tree-based strategies).

2.2 Various Eviction Strategies

Original binary-tree ORAM. The original Binary Tree ORAM scheme [?] adopts random eviction: with every data access, pick two random buckets from each level, and evict one block from each selected bucket by placing the block into the correct child node (subject to the invariant). Shi *et al.* show that this eviction strategy requires a bucket size of $O(\log N)\omega(1)$ to avoid overflow except with negligible probability.

Path ORAM. Path ORAM adopts a greedy eviction strategy that works with a *stash* maintained in client storage. Blocks on the path P are first unioned with the stash. Each block in this set is then placed as close as possible to its destination leaf in path P subject to the path invariant. Stefanov *et al.* [?] show that this aggressive eviction strategy works with buckets that are only 4 blocks.

CLP ORAM. In the CLP ORAM [?] scheme, internal nodes of the binary tree have $O(\log \log N)$ blocks per bucket, and the stash is replaced by a queue that requires add, pop, and find operations. After path P is read, block x is removed and remapped, and then added to the queue. Next, the ORAM performs a “flush” operation by selecting a new leaf at random, reading its path into the ORAM, and then placing each block in this path as close as possible to the block’s destination leaf subject to the path invariant. This flush operation is performed according to a geometrically distributed random variable with expectation 2.

This scheme claims $\tilde{O}(\log^2(N))$ worst-case computational overhead, constant memory overhead, and CPU cache size that is poly $\log(N)$. Our empirical results confirm the claim in CLP that the data structure required to implement the queue is simpler than the one needed to implement the stash in the Path ORAM construction. However, as we show in the next section, the Path SCORAM and the SCORAM can be

```

1:  $P[0..L-1][Z]$  stores the block to be put back
2: for  $i$  from 0 to  $L-1$  do //from leaf to root
3:   for  $j$  from 1 to  $Z$  do
4:     for  $k$  from 1 to  $LZ + \text{stsize}$  do
5:       if  $\text{LCA}(A[k].\text{label}, P) \leq i$  then
6:          $P[i][j] := A[k]$ 
7:          $A[k] := \perp$ 

```

Figure 1: **Naive oblivious algorithm for Path ORAM’s Eviction.** Variable A denotes a buffer created by concatenating the stash and the path P read in Path ORAM. This algorithm writes as many blocks back to P as possible, packing them as close to the leaf as possible.

optimized to support an even simpler (and smaller) eviction algorithm.

2.3 SCORAM

A SCORAM is an ORAM scheme that is specifically designed for use in the Ostrovsky-Shoup framework for secure computation of RAM programs. Unlike the traditional ORAM scenario, in this framework, both the server memory and the client storage of the ORAM are secret-shared between the parties⁴.

Each *instruction* of a RAM program consists of an address to read from memory, an operation to perform, and an address to write back to memory. These three steps are accomplished via a secure computation protocol between the parties that takes as input (a) secret shares of the memory, (b) secret shares of the ORAM client state, (c) and secret shares of the program state.

The most significant component of this secure computation is typically the logic used to implement the read and write operations via the SCORAM design. In particular, the SCORAM design contributes most of the (a) inputs each party must supply to the secure computation, (b) and the logical gates of the secure computation protocol used to implement the three steps of the *instruction*. Among the components of the SCORAM design, the eviction procedures is the costliest. In the next section, we analyze the eviction procedures for several SCORAM designs.

3. FIRST ATTEMPT: PATH SCORAM

We first investigate the Path ORAM since has the least bandwidth overhead among all tree-based ORAMs. Naturally a good question is whether we can implement Path ORAM with a small circuit as well. since **ReadAndRemove** and **Add** algorithm are trivial to turn into $O(D \log N)\omega(1)$ -sized circuits, we focus our discussion on how to implement Path ORAM’s eviction algorithm in circuit.

Expressing circuits. In the remainder of the paper, we will need to describe circuit constructions. Since oblivious algorithms can be easily transformed into circuits preserving complexity, in this paper, we equate the notions of *oblivious algorithms* and *circuits*, and describe circuits using oblivious algorithms. When we sort lexicographically according to a

⁴We note that Gordon *et al.* propose an asymmetric division of the server and client state to enable one party in the secure computation protocol to have a sub linear number of input bits.

1. **Initialization.** For each $i \in \{1, 2, \dots, LZ\}$: let $A[i].\text{bucket} := \text{LCA}(A[i].\text{label}, P)$, let $A[i].\text{dummy}$ denote whether $A[i]$ is a dummy block.
2. **O-sort: real before dummy.** Oblivious sort based on the key bucket . After sorting, all real blocks come first, in bucket ascending order.
3. **Scan A to compute the final offset (from leaf) for each block.** We give the algorithm to calculate offset from bucket in Figure 4.
4. **Append dummy.** Append LZ dummy blocks at the end whose offset are $1, \dots, LZ$, respectively.
5. **O-sort: reorder based on offset.** Oblivious sort A by offset (where blocks with $\text{offset} = \perp$ are put at the end).
6. **Suppress unnecessary dummy.** Scan A to “eliminate” unnecessary dummy blocks. A dummy block is considered unnecessary if it is preceded by a real block with the same offset value. We “eliminate” this dummy block by setting its offset to “ $\perp$ ”.
7. **O-sort: fall into place.** Sort A by offset so that unnecessary dummies are moved to the end A .

Figure 2: The PATH SCORAM’s eviction algorithm.

key pair $(\text{key}_1, \text{key}_2)$, we mean first sort according to key_1 , and if there is a tie on key_1 , then we sort according to key_2 .

Notations. Suppose each of p_1 and p_2 represents a leaf or a root-to-leaf path. We use $\text{LCA}(p_1, p_2)$ to denote the level of the lowest common ancestor of the leaves, or equivalently the node where the two paths diverge when traversing from the root. In the remainder of the paper, we sometimes use the notation label and a path p interchangeably, since a path p is defined by the label of a leaf node.

Let A denote a buffer created by concatenating the stash concatenated with a path p (the data read path) in Path ORAM. Path ORAM’s eviction writes as many blocks from A to p as possible, and packs them as close to the leaf as possible.

Naive $O(D \log^2 N)$ eviction circuit. A naive way to turn Path ORAM’s eviction algorithm into a circuit would result in a $O(D \log^2 N) \omega(1)$ -sized circuit. We describe this naive method in Figure 1.

3.1 A New $\tilde{O}(D \log N)$ Eviction Circuit

The rearrangement problem. Path ORAM’s eviction can be recast as a rearrange problem: we would like to obliviously rearrange the entries in A (by pairwise swapping) with respect to the following conditions:

1. The blocks in $A[1..LZ]$ will be written back to the path P , where the block at $i = kZ + j$ (where $k = \lfloor \frac{i-1}{Z} \rfloor$ and $1 \leq j \leq Z$) will go to bucket k . We require that for each i , if $A[i]$ is a real block, then $\text{LCA}(A[i].\text{label}, P) \leq k$. If less than Z real blocks are assigned to a bucket, dummy blocks will be added to that bucket.
2. For the real blocks that cannot be written in the path, they will be stored in $A[LZ + 1..LZ + \text{stsize}]$. Hence, if $A[LZ + \text{stsize} + 1]$ contains a real block, this indicates stash overflow.

Circuit construction. First of all, we add three extra fields bucket , offset and dummy (used only in this improved eviction algorithm) to each entry $A[i]$. bucket takes values from $\{0, \dots, L-1\} \cup \{\perp\}$. For a dummy block, bucket is set to $\perp$. For a real block, $\text{bucket} := \text{LCA}(\text{label}, P)$, where label denotes the leaf label of the block. In other words, bucket is the lowest level (i.e., closest to leaf) where the block is allowed to reside on the path P . Recall that we assume $\text{bucket} = 0$ refers to the leaf level while $\text{bucket} = L-1$ refers to the root level. offset takes values from $\{1, \dots, LZ\} \cup \{\perp\}$. dummy indicates if a block is dummy.

The improved eviction algorithm is described in Figure 2. At the end of the algorithm, the first Z blocks of A can go to the leaf, path, and the next Z blocks should go to leaf but one level, and so forth.

In Appendix B, we also consider how to construct a low-depth circuit for our new eviction algorithm.

Examples. The most sophisticated part of the algorithm is Step 3 in Figure 2, i.e., computing the offset of blocks. This part of the algorithm is further explained in Figure 4. Below we give an example of this step. Assume that $Z = 3$. Then, a sequence of bucket values $[0 \ 0 \ 0 \ 1]$ should result in offset values $[1 \ 2 \ 3 \ 4 \ 5]$. Additionally, bucket values $[0 \ 1 \ 1 \ 1 \ 1]$ should result in offset values $[1 \ 4 \ 5 \ 6 \ 7]$.

An example of the full eviction circuit is given in Figure 3.

```

1: available := 1
2: for i from 1 to LZ + stsize do
3:   if available ≥ Z * A[i].bucket + 1 then
4:     A[i].offset := available
5:     available := available + 1
6:   else
7:     A[i].offset := Z * A[i].bucket + 1
8:     available := A[i].offset + 1

```

Figure 4: Computing offset from bucket .

THEOREM 1. *The PATH SCORAM with Eviction described above can be fully implemented (i.e., including all the recursive ORAMs used to store the position map) in $\tilde{O}(\log^3 N + D \log^2 N)$ boolean gates.*

PROOF. The circuit to implement the data level ORAM requires $O(D \log N \log \log N)$ gates due to the oblivious sorting operations in Figure 2. For recursion levels, we use $O(\log N)$ for each block, and with $O(\log N)$ levels of recursion, the total circuit size for position map recursion levels is $O(\log^3 N \log \log N)$. Therefore, the total circuit size across all levels is $\tilde{O}(\log^3 N + D \log N)$. $\square$

Findings. We implement our PATH SCORAM and compare its empirical performance with that of the Binary Tree ORAM and CLP ORAM. While PATH SCORAM is asymptotically superior to both Binary Tree ORAM and CLP ORAM in terms of circuit size, for practical ranges of N , empirical results suggest that both the Binary Tree ORAM and CLP ORAM perform better. Detailed empirical results are presented in Section 6.

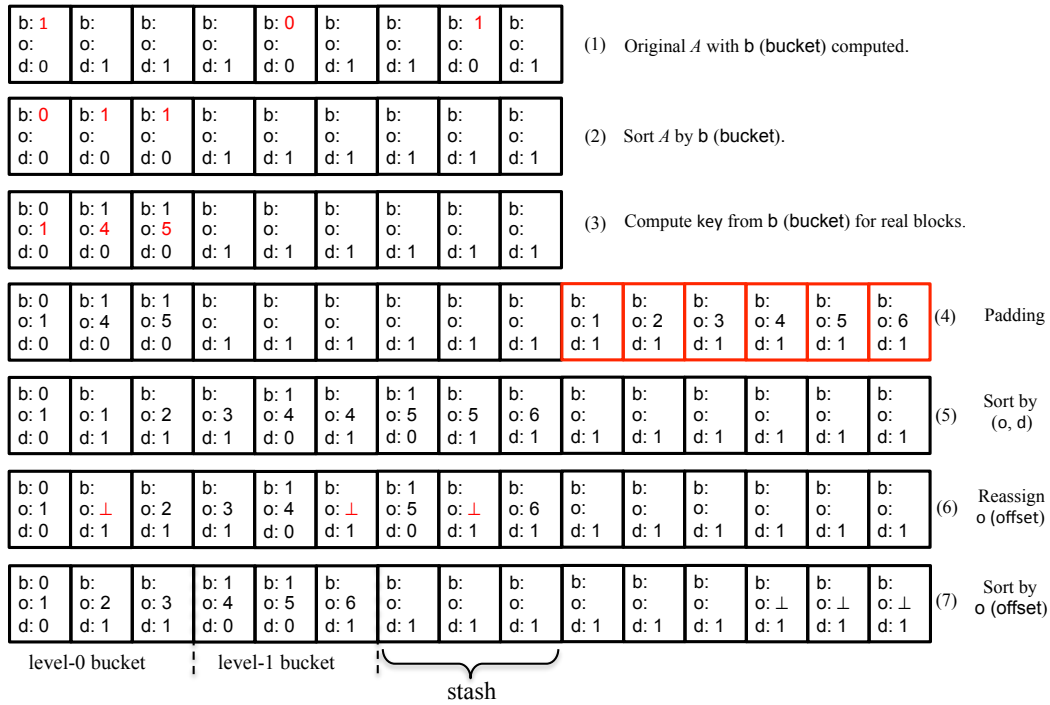


Figure 3: Running $\tilde{O}(D \log N)$ eviction algorithm over a toy example. Here we assume $L = 2, Z = 3$ and $\text{stsize} = 3$. b stands for the bucket field while k the offset field. $d=1$ indicates dummy blocks, where the contents of other fields are not applicable.

Upon closer examination, we realize that PATH SCORAM requires 3 oblivious sorts in its circuit construction, thus incurring a large constant. This motivated our observation that asymptotics is not representative of practical performance for the $\text{poly log } N$ range, since $\log N$ is so small (e.g., 20 to 30) for real-life ranges of N , that constants matter just as much as $\log N$ factors.

4. SECOND ATTEMPT: A NEW SCORAM

Because asymptotic evaluation does not properly characterize practical performance for our problem, we devise a new SCORAM scheme optimized for empirical performance. Our key idea is to have an effective eviction algorithm that can be implemented in small circuit. In this section, we focus on presentation of the algorithm, leaving optimal parameter choices to the next section.

4.1 Our New SCORAM Scheme

Our new compact SCORAM, which we call SCORAM, is another tree-based ORAM. `ReadAndRemove` and `Add` operate in the same way as the tree-based ORAMs, and we therefore focus the presentation on the Eviction algorithm.

Overall, Eviction will perform `flush()` (Algorithm 1) α times. In our implementation, we choose $\alpha = 4$ which we determine to be the optimal choice (see §5 for more details). Below we explain the intuition.

Greedy push pass (lines 7 to 10) We opt for a greedy push pass similar to that of the CLP ORAM, but avoid their *bucket overflows* handling strategy. First, a random path is selected for eviction with every data access. Next, for each bucket from root to the leaf-1 level on the selected path: pick a block that can be evicted deepest along this path,

and push it to the child bucket if there is room.

In the CLP ORAM, a bucket overflow occurs when more than half of the bucket capacity number of blocks can be evicted to one of its children. Overflow events are indicative of getting too crowded in some parts of the tree. Chung et al. handles this event by choosing a block to remove from the bucket, remapping its label, and putting it back into the stash. This is an expensive operation, since (1) removing (adding) blocks from a bucket (to the stash) requires a linear scan; (2) it needs to be done for every bucket on the eviction paths because we cannot reveal where the actual overflows happen; (3) updating the (recursively stored) position map is also very expensive. Our implementation suggests that for a RAM storing 1 million 32-bit entries, more than 85% of the computational cost are due to obviously handling overflows. Although some of the above overflow handling logic can be implemented asymptotically better using oblivious sorting, this actually worsens the empirical overhead due to the large constant factor associated with oblivious sorting.

Therefore, our new SCORAM avoids checking and handling overflows. A block will only be evicted if the child bucket is not full. Unfortunately, such a “greedy push pass” alone would make the eviction less effective, as it is more likely for a bucket to be full and the stash could grow unduly large. This motivates our idea of compensating with a “reverse dropping pass”.

Reverse dropping pass (lines 3 to 6). Pick a block that can be pushed deepest along the path, then put it into a bucket as deep in the path as possible. This can be implemented by scanning the eviction path in reverse order from leaf to root and placing the block into the first non-full bucket that satisfies the block’s path-invariant.

Algorithm 1 flush()

```
1: path := UniformRandom(0, ..., N - 1)
2: bucket[0, ..., L - 1] := array of buckets from leaf to root
3: B1 := the block in the stash with smallest LCA(path, B.label).
4: for i from 0 to L - 1 do (from leaf to root)
5:   if bucket[i] is not full and LCA(path, B.label) ≤ i and B1 has not been added already then
6:     Add B1 to bucket[i].
7: for i from L - 1 to 1 do (from root to leaf)
8:   B2 := the block in bucket[i] with smallest LCA(path, B2.label).
9:   if bucket[i - 1] is not full and LCA(path, B2.label) < i then
10:    Move B2 from bucket[i] to bucket[i - 1].
```

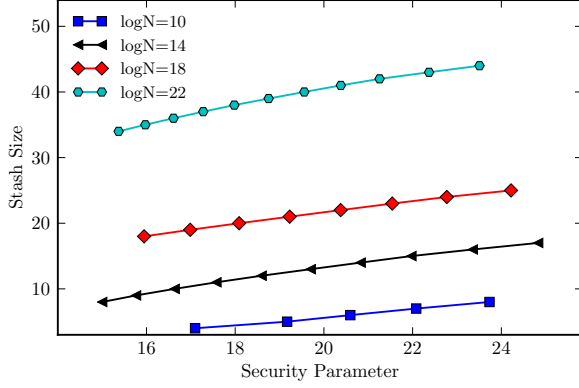


Figure 5: **scoram: stash size grows linear with security parameter.** Bucket size $Z = 6$, number of flushes $\alpha = 4$.

Security. Security of the scheme follows as per all other tree-based ORAMs; the server’s view observes accesses along random paths.

5. OPTIMIZATIONS

Our optimizations largely fall into two categories: (1) determining the best parameters for a particular SCORAM configuration (Section 5.1); (2) improving the circuit design for frequently used logic components (Section 5.2).

In general, ORAM schemes have several decisive parameters including bucket size, stash size (if it uses a stash) and recursion factor. Depending on the specific eviction algorithms employed, there are additional parameters describing the eviction process. For example, in our new SCORAM, we use α to denote the number times to call **flush**. These parameters are subtly inter-related. For instance, a larger bucket size allows for smaller α and smaller stash at the same security parameter.

5.1 Parameter Optimization

We now systematically explore this parameter space to determine heuristically good choices for our new SCORAM scheme.

Methodology. Some of our optimizations rely on simulations to count, for any particular ORAM setup, the number of ORAM failures (for estimating the security guarantee) and the total number of encryptions per memory access. We use simulations because all of the proofs that upper-

bound the failure probabilities are too conservative in their approximations.

For each ORAM, we first run 16 million ORAM accesses to warm up the ORAM, such that it enters a steady state, we then start collecting numbers to determine the security parameters associated with this setup (including e.g., various bucket and stash sizes). Since the time average is equal to ensemble average for regenerative processes [?], we simulate each ORAM setup for a single long run of 1 billion accesses to estimate the security parameter (instead of multiple runs). This allows us to estimate ORAM parameters that achieve up to 2^{-80} security. A similar approach was suggested by Stefanov *et al.* [?].

Number of flushes and bucket size. We have run simulations of our new ORAM that indicate a good choice of α is 4.

After α is fixed, we consider two strategies to configure bucket size: 1) a uniform bucket size everywhere; and 2) varying bucket sizes across levels. For the former, we empirically determine an optimal bucket size of 6. However, we observe that buckets at the middle and lower part of the path tend to be more congested. This motivates us to redistribute the bucket size across levels. For example, for a binary tree of 21 levels, we find that increasing the bucket size by 1 for the first 10 buckets from root and decreasing the bucket size by 1 for the last 10 bucket at leaf is a better distribution than evenly distributing buckets. Assuming 80-bit security, varying the bucket size in this way allows us to reduce the stash size from 66 to 50, resulting in a circuit of size of 4,094,832 gates versus 4,562,988 for the version of SCORAM reported in Table 1, i.e., roughly a 10% reduction.

Stash size. We plot the stash size versus security parameters with different $\log N$ in Figure 5. Each point (x, y) on the curve should be read as “with a stash size of more than y , ORAM failures were observed 2^{-x} fraction of the time”. the stash size grows linearly with security parameters, suggesting the failure probability decreases exponentially with the stash size. Note that unlike the PATH SCORAM, the stash size is super-linear in $\log N$.

Recursion factor. Here we study the choices needed to recursively implement the position-map in all tree-based SCORAM designs. We formulate the choice as an optimization problem as follows.

As before, let N denote the number of blocks (and thus the size of the position map), D denote the number of bit per blocks, and for our given scheme, let $f(N, D) = O(D \log^e N) \omega(1)$ denote the circuit size for one ORAM access *excluding* the cost of the position map lookup. Let $LS(N, D)$ denote

χ	ℓ	Total # gates	
		$N = 2^{20}$	$N = 2^{29}$
2	9	6,657,629	20,363,468
4	5	4,562,988	13,619,451
8	3	4,657,921	13,382,510
16	2	5,518,948	15,657,910
32	1	6,933,117	21,455,341
64	1	11,120,697	34,165,313

Table 2: **How to pick χ and the recursion level** This table shows how we concretely optimize recursion parameters needed to implement the position maps in the SCORAM. χ describes how many addresses are packed into each block, and ℓ describes the number of recursive steps before we use the linear-scan ORAM as the base case.

the circuit size of the linear scan ORAM.

In our top-level ORAM, we have $D_0 = 32$. When implementing the position map recursively with another ORAM, we must select the number of labels, χ , to pack into a block and the number of times, ℓ , to recurse before finally using the linear scan ORAM as the base case. Note, the ORAM for the i^{th} recursive level has $N_i = \frac{N}{\chi^i}$ blocks of size $D_i = \chi \log N_{i-1}$. Thus, the total cost of the SCORAM with recursion is

$$\sum_{i=0}^{\ell} f(N_i, D_i) + LS(N_{\ell+1}, D_{\ell+1})$$

For example, when $N = 2^{20}$, $\chi = 8$ and $\ell = 3$, we have total cost: $f(2^{20}, 32) + f(2^{17}, 160) + f(2^{14}, 136) + f(2^{11}, 112) + LS(2^8, 88)$. Since we can empirically determine f for any parameter setting, we can thus minimize the total cost. In our case, we limit χ to a power of 2 and use the same χ at each recursive level. See Table 2 for the results of this optimization. When $N = 2^{20}$, we use $\chi = 4$ and $\ell = 5$, and when $N = 2^{29}$, we use $\chi = 8$ and $\ell = 3$.

5.2 Backend-Independent Optimizations

Reducing gates. A highly frequently used logic is to determine if a block is dummy. We add a single bit field `isDummy` to each block indicating whether the block is `dummy`. This simple trick reduces the circuit size from $\log N$ AND gates to 1 AND gate. In addition, an important side benefit is that it enables efficient oblivious removal of a block from the bucket. We only need to set the `isDummy` field instead of resetting all bits of a block.

5.3 Backend-Dependent Optimizations

Fewer non-free gates Some secure computation protocols such as the garbled circuit and the GMW protocol support almost-free XOR gates. We make several circuit level optimizations especially exploiting this opportunity.

A common operation in some Eviction algorithms (e.g., CLP SCORAM, SCORAM) is, given a bucket B of Z blocks $b_1, \dots, b_Z$, to find the block b^* that can be pushed deepest along the path P without violating the path invariant. Intuitively, we can first calculate $\text{LCA}(b_i, \text{label}, P)$ for every i ; then compute b^* whose label is $\max_{1 \leq i \leq Z} (b_i)$. Since each label has $\log N$ bits, counting the number of leading zeros for every block requires $\ell + \log \ell$ AND gates and computing the

maximum for all Z blocks requires $(Z - 1) \log \ell$ AND gates. Therefore, it uses a total of $Z(\ell + \log \ell) + (Z - 1) \log \ell$ gates.

In our implementation, we use a circuit of size $(2Z - 1)L$ to compute exactly this function. The idea is to (1) use a linear scan to reset every bit of the label except the leading '1' bit (which costs a total of $Z\ell$ gates per bucket); (2) then use $Z - 1$ comparison circuit to find the minimal label, which belongs to the bucket to be flushed.

6. PERFORMANCE EVALUATION

6.1 Methodology and Metric

Our evaluation focuses on the following types of metrics:

1. *Cryptographic backend independent* metrics, such as gate count. This characterizes the performance of a SCORAM in general, relatively independent of the cryptographic backends.
2. *Cryptographic backend dependent* metrics, such as the number of encryptions, number of non-free gates, and bandwidth. These metrics characterize the performance of an ORAM scheme with a semi-honest Garbled Circuit backend in a manner that is independent of the specific hardware configuration or implementation artifacts.
3. *Implementation and machine dependent* metrics, such as runtime and breakdown of runtime. We will describe our specific hardware configuration and the specific Garbled Circuit implementation as the context of our results. We also discuss the interpretation of these results and project the performance had the experiments been run on a different hardware configuration or with a more optimized Garbled Circuit implementation.

For all ORAMs plotted, each payload data is chosen to be 32 bits. We set both the computational and statistical security parameters to be 80 bits.

6.2 Comparing ORAMs for secure computation

We reevaluate the state-of-the-art ORAM schemes over secure computation, and compare their performance with our new SCORAM. The metrics we considered include total gate count (Figure 6a), non-free gate count (Figure 6b), number of AES encryptions (Figure 6c), and input size of circuit (Figure 6d).

Table 3 summarizes the margin by which our SCORAM outperforms existing schemes. In particular, we show that across all metrics, we achieve $7.6\times$ to $9.8\times$ performance improvements comparing to the respective second best candidates.

As mentioned earlier, even though the PATH SCORAM is asymptotically better than Binary Tree ORAM, for practical ranges of N , Binary Tree ORAM outperforms PATH SCORAM due to lower constants in the asymptotic bound. This is shown in Figure 6.

The most popular ORAM scheme used previously is Binary Tree ORAM. Our newest scheme is 7 times smaller and requires 27x fewer inputs. Note that each input bit involves an oblivious transfer, which, even with OT extension, incurs 2 encryptions.

For the sake of reproducibility, we report all parameter settings that we used to run our experiments in Table 5 in the Appendix.

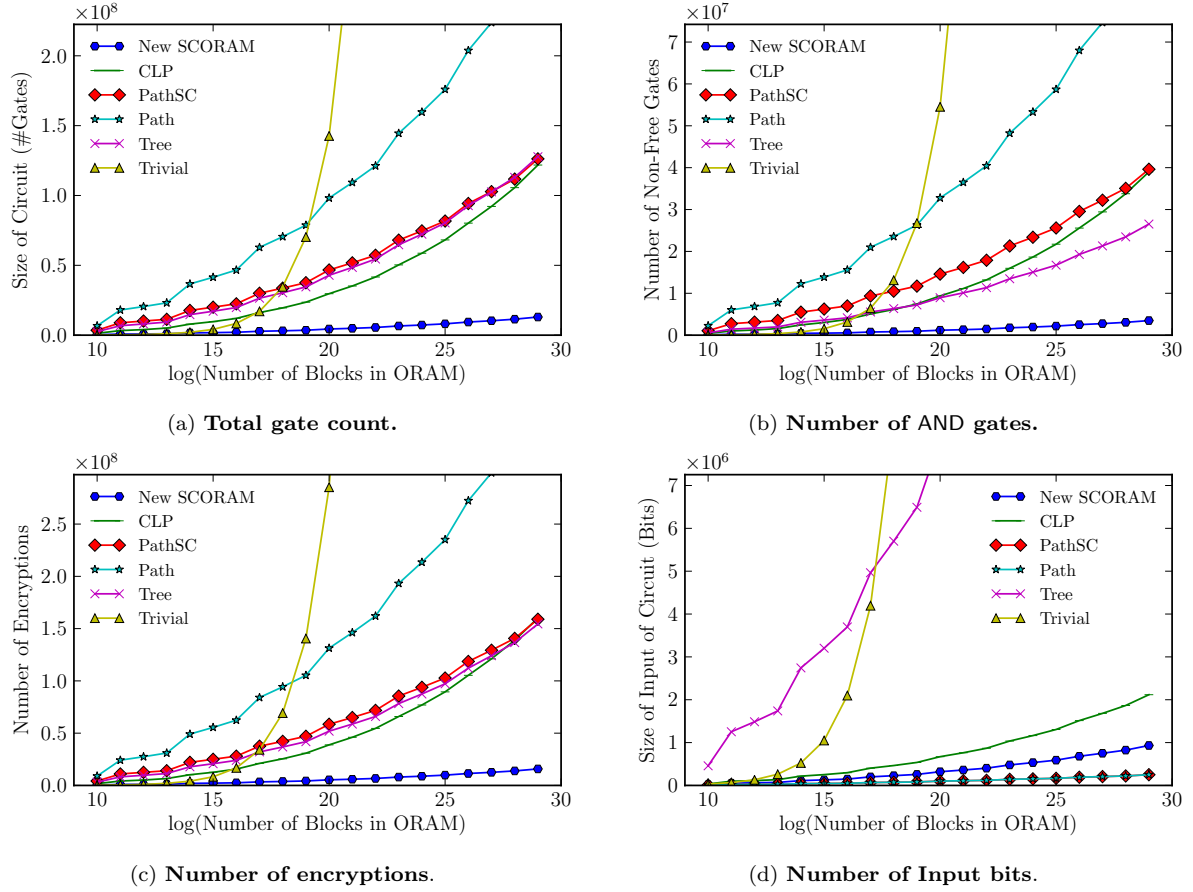


Figure 6: Comparison of various ORAMs. Payload bitlength = 32 bits, security parameter = 80.

6.3 Performance Profiling for SCORAM

We further investigate the performance breakdown of SCORAM scheme. Specifically, the cost can be broken down into I/O overhead and computation overhead. I/O overhead can be further broken down into transmission time (i.e., total # bytes transferred/network bandwidth), and synchronization overhead (i.e., all queuing delays in the JVM, OS, and on the network stack).

Our machine/implementation-dependent measurements are taken on a single server (Intel Xeon 2.13G Hz) running the circuit generator and evaluator as two independent processes (communicating through Java Socket). The memory usage ranges from 4–80 GB for both processes depending on the data size.

Interpreting the measurements. First, our implementation does not currently exploit the AES-NI instructions to speed-up garbling. We expect a noticeable speedup for the computation time when hardware AES is implemented.

Second, the Garbled Circuit backend we used is a Java-based implementation. Therefore, our timing measurements are subject to the artifacts of the Java-based implementation. There are two main sources of artifacts, memory garbage collector and I/O synchronization. In Figure 7, we note that a substantial portion of the time is due to I/O. Since the experiments are run on the same machine where network bandwidth is not a bottleneck, we infer majority of the I/O time we recorded are due to I/O synchronization.

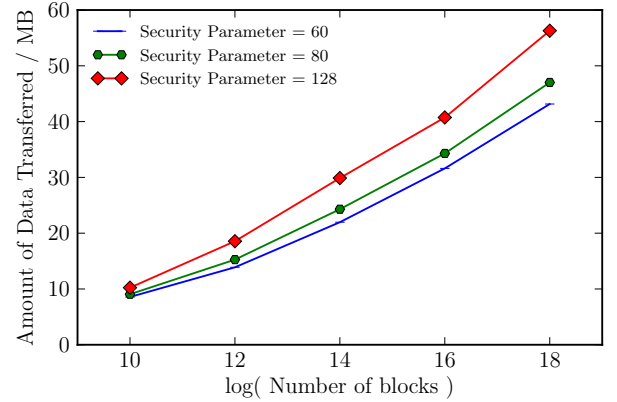


Figure 8: The amount of data transferred per access for SCORAM. Payload bitlength is 32 bit, 80-bit security parameter.

Note that for the circuit generator, we observe the computation time spent on OT and garbling is roughly the same. On the circuit evaluator side, the portion of I/O cost is larger because the evaluator indeed has less computational work to do (thanks to the garbled row reduction technique) while being stuck more often waiting for the generator.

Network transmission. Figure 8 plots the bandwidth

Metric	Second best	Performance gain of scoram	
		$N = 2^{20}$	$N = 2^{29}$
Gate count	CLP ORAM	6.7X	9.4X
Non-Free Gates	Binary Tree ORAM	7.6X	7.6X
Number of Encryptions	CLP ORAM	7.2X	—
	Binary Tree ORAM	—	9.8X

Table 3: **Performance comparison of scoram over the second best candidates.** (Data payload: 32 bits. Security parameter: = 80). Considering the number of encryptions, the second best is the CLP ORAM when $N = 2^{20}$, and the Binary Tree ORAM when $N = 2^{29}$.

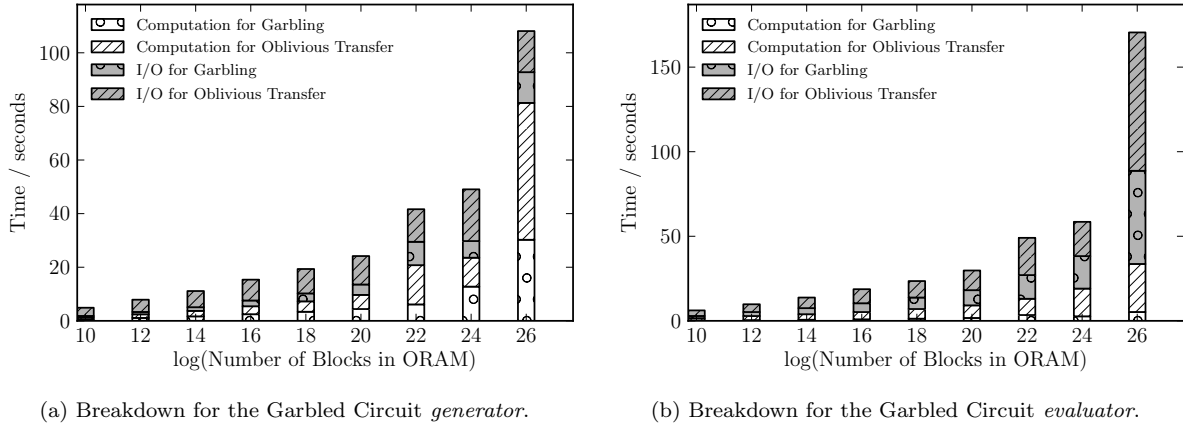


Figure 7: **Cost breakdown of SCORAM.** (Data payload: 32-bit, Security parameter: 80-bit)

consumption for our new SCORAM scheme, under different security parameters. Note that the bandwidth overhead is proportional to the total number of non-free gates in our garbled circuit system. The non-free-gate counts (Figure 6b) correlates with growth in the bandwidth.

Comparison with other ORAM implementations over secure computation We are aware of two other SCORAM implementations. Gordon *et al.* [?] report on an ORAM with $N = 2^{20}$ and $D = 512$. The paper does not clearly report the bucket size they use, but we assume the bucket size is 40 based on our interpretation of their paper. Their ORAM requires 11.9M gates and takes approximately 50 seconds for one operation. In contrast, our best implementation for their parameters requires 3,743,213 total gates, and our implementation at significantly higher security parameters runs in under 30 seconds.

The second ORAM implementation over secure computation is by Marcel Keller and Peter Scholl [?]. Their implementation was based on the SPDZ protocol, running in the *preprocessing* model (assuming sufficient CPU/Memory resource for preprocessing). Their flagship scheme was based on Path ORAM, with heuristic modifications (so the security is also based on simulation). They reported their protocol can securely compute an ORAM access in under 250 ms (not counting the expensive pre-computation) over a dataset of size 1 million (with 20-bit security). However, besides the differences in the crypto-backends, hardware, programming systems and execution model, many other important context (e.g., data payload size, gate counts, etc.) remain to be clarified for a fair comparison. Complementary to their work, we provide implementations for 5 quite

different ORAMs and it is possible to run the SPDZ crypto backend over our SCORAM implementations.

We also notice that Liu *et al.* [?] study RAM-based secure computation. However, their experiments were based on an program that only produces simulated performance numbers rather than a functional SCORAM.

Execution of the Garbled Circuit in the Honest-but-Curious model We also executed our SCORAM on a garbled circuit platform and measured the time spent on the different phases of the secure computation protocol. Figure 7b and Figure 7a shows the time for client and server respectively. Here we split the work by task into Garbling and Oblivious Transfer, which are further divided into Computation time and I/O time. As the plot indicates, if we have perfect communication channel, the overall time will be dominated by computation time, which corresponds to the white part of the bar. For $N = 2^{20}$, each SCORAM access requires roughly 30 seconds. While this is several times faster (at higher security parameters) than the best previous results, it indicates that secure computation in the RAM still incurs several *orders of magnitude* slowdown over plain implementations, and thus more research is needed.

7. CONCLUSION

As the key enabling primitive of RAM-based secure computation, the construction of efficient SCORAM remains a challenge. We have made the first step towards building practical SCORAMs by performing a thorough study of the performance of 5 state-of-art SCORAM constructions based on several metrics closely related to common instantiation

of secure computation protocols. We look forward to releasing our code to benefit the academic research in the related fields.

8. REFERENCES

- [1] M. A. H. Aseem Rastogi and M. Hicks. Wysteria: A programming language for generic, mixed-mode multiparty computations. *IEEE S & P*, 2014.
- [2] A. Ben-David, N. Nisan, and B. Pinkas. FairplayMP: A System for Secure Multi-party Computation. In *ACM Conference on Computer and Communications Security*, 2008.
- [3] D. Boneh, D. Mazieres, and R. A. Popa. Remote oblivious storage: Making oblivious RAM practical. <http://dSPACE.mit.edu/bitstream/handle/1721.1/62006/MIT-CSAIL-TR-2011-018.pdf>, 2011.
- [4] K.-M. Chung, Z. Liu, and R. Pass. Statistically-secure oram with $\tilde{O}(\log^2 n)$ overhead. *arXiv preprint arXiv:1307.3699*, 2013.
- [5] I. Damgård, S. Meldgaard, and J. B. Nielsen. Perfectly secure oblivious RAM without random oracles. In *TCC*, 2011.
- [6] Y. Ejgenberg, M. Farbstain, M. Levy, and Y. Lindell. SCAPI: The secure computation application programming interface. Cryptology ePrint Archive, Report 2012/629, 2012.
- [7] C. W. Fletcher, M. v. Dijk, and S. Devadas. A secure processor architecture for encrypted computation on untrusted programs. In *STC*, 2012.
- [8] C. Gentry, K. A. Goldman, S. Halevi, C. S. Jutla, M. Raykova, and D. Wichs. Optimizing ORAM and using it efficiently for secure computation. In *Privacy Enhancing Technologies Symposium (PETS)*, 2013.
- [9] O. Goldreich. Towards a theory of software protection and simulation by oblivious RAMs. In *ACM Symposium on Theory of Computing (STOC)*, 1987.
- [10] O. Goldreich and R. Ostrovsky. Software protection and simulation on oblivious RAMs. *J. ACM*, 1996.
- [11] M. T. Goodrich and M. Mitzenmacher. Privacy-preserving access of outsourced data via oblivious RAM simulation. In *ICALP*, 2011.
- [12] M. T. Goodrich, M. Mitzenmacher, O. Ohrimenko, and R. Tamassia. Oblivious RAM simulation with efficient worst-case access overhead. In *CCSW*, 2011.
- [13] M. T. Goodrich, M. Mitzenmacher, O. Ohrimenko, and R. Tamassia. Privacy-preserving group data access via stateless oblivious RAM simulation. In *SODA*, 2012.
- [14] S. D. Gordon, J. Katz, V. Kolesnikov, F. Krell, T. Malkin, M. Raykova, and Y. Vahlis. Secure two-party computation in sublinear (amortized) time. In *ACM conference on Computer and Communications Security*, pages 513–524, 2012.
- [15] M. Harchol-Balter. *Performance Modeling and Design of Computer Systems: Queueing Theory in Action*. Performance Modeling and Design of Computer Systems: Queueing Theory in Action. Cambridge University Press, 2013.
- [16] Y. Huang, D. Evans, J. Katz, and L. Malka. Faster Secure Two-Party Computation Using Garbled Circuits. In *USENIX Security Symposium*, 2011.
- [17] M. Keller and P. Scholl. Efficient, oblivious data structures for mpc. Cryptology ePrint Archive, Report 2014/137, 2014. <http://eprint.iacr.org/>.
- [18] B. Kreuter, B. Mood, A. Shelat, and K. Butler. PCF: A Portable Circuit Format for Scalable Two-Party Secure Computation. In *USENIX Security Symposium*, 2013.
- [19] B. Kreuter, A. Shelat, and C. hao Shen. Billion-Gate Secure Computation with Malicious Adversaries. In *USENIX Security Symposium*, 2012.
- [20] E. Kushilevitz, S. Lu, and R. Ostrovsky. On the (in)security of hash-based oblivious RAM and a new balancing scheme. In *SODA*, 2012.
- [21] C. Liu, Y. Huang, E. Shi, J. Katz, and M. Hicks. Automating efficient ram-model secure computation. *IEEE S & P*, 2014.
- [22] S. Lu and R. Ostrovsky. Distributed oblivious ram for secure two-party computation. In *Proceedings of the 10th Theory of Cryptography Conference on Theory of Cryptography, TCC'13*, pages 377–396, Berlin, Heidelberg, 2013. Springer-Verlag.
- [23] M. Maas, E. Love, E. Stefanov, M. Tiwari, E. Shi, K. Asanovic, J. Kubiatowicz, and D. Song. Phantom: Practical oblivious computation in a secure processor. In *CCS*, 2013.
- [24] P. MacKenzie, A. Oprea, and M. Reiter. Automatic Generation of Two-party Computations. In *ACM Conference on Computer and Communications Security*, 2003.
- [25] D. Malkhi, N. Nisan, B. Pinkas, and Y. Sella. Fairplay: A secure two-party computation system. In *USENIX Security*, 2004.
- [26] R. Ostrovsky. Efficient computation on oblivious RAMs. In *ACM Symposium on Theory of Computing (STOC)*, 1990.
- [27] R. Ostrovsky and V. Shoup. Private information storage (extended abstract). In *ACM Symposium on Theory of Computing (STOC)*, 1997.
- [28] B. Pinkas and T. Reinman. Oblivious RAM revisited. In *CRYPTO*, 2010.
- [29] E. Shi, T.-H. H. Chan, E. Stefanov, and M. Li. Oblivious RAM with $O((\log N)^3)$ worst-case cost. In *ASIACRYPT*, 2011.
- [30] E. Stefanov and E. Shi. Fastprp: Fast pseudo-random permutations for small domains. Cryptology ePrint Archive, 2012. <http://eprint.iacr.org/>.
- [31] E. Stefanov and E. Shi. Multi-cloud oblivious storage. In *ACM Conference on Computer and Communications Security (CCS)*, 2013.
- [32] E. Stefanov and E. Shi. Oblivstore: High performance oblivious cloud storage. In *IEEE Symposium on Security and Privacy (S & P)*, 2013.
- [33] E. Stefanov, M. van Dijk, E. Shi, C. Fletcher, L. Ren, X. Yu, and S. Devadas. Path ORAM: an extremely simple oblivious ram protocol. In *CCS*, 2013.
- [34] X. Wang, K. Nayak, C. Liu, E. Shi, E. Stefanov, and Y. Huang. Oblivious data structures. <http://eprint.iacr.org/2014/185.pdf>.
- [35] P. Williams and R. Sion. Usable PIR. In *Network and Distributed System Security Symposium (NDSS)*, 2008.
- [36] P. Williams and R. Sion. Round-optimal access

- privacy on outsourced storage. In *ACM Conference on Computer and Communications Security (CCS)*, 2012.
- [37] P. Williams, R. Sion, and B. Carbunar. Building castles out of mud: Practical access pattern privacy and correctness on untrusted storage. In *CCS*, 2008.

APPENDIX

A. NOTATIONS

Table 4: Some globally used notations.

Variable	Meaning
N	number of blocks in ORAM
D	payload bit-length
L	number of levels in binary tree
Z	capacity of each bucket
label	leaf label of a block
$\text{LCA}(p_1, p_2)$	lowest common ancestor of paths p_1 and p_2

B. A LOGARITHMIC DEPTH CIRCUIT

We describe how to construct a low-depth circuit for PATH SCORAM. The main steps of the algorithm are:

1. **Setting temporary offset fields.** As described before, we first sort the array A obliviously according to the field **bucket**. For each entry $A[i]$, if it is a dummy block, set its field $A[i].\text{offset} := \perp$. Otherwise, for $1 \leq i \leq LZ$, set $A[i].\text{offset} := Z \cdot A[i].\text{bucket} + 1$; recall that our final goal is to rearrange blocks such that the entry $A[i]$ should go to bucket $\lfloor \frac{i-1}{Z} \rfloor$, and so the **offset** value is the smallest index at which the block can reside in A . For $LZ + 1 \leq i \leq LZ + \text{stsize}$, the block $A[i]$ cannot be written back in the path P , and we set $A[i].\text{offset} := \text{st}$. Setting the **offset** fields takes circuit of size $O(S)$ and depth $O(1)$. For sorting the key values, we use the natural order for positive integers $\mathbb{Z}^+$, and use the convention $\mathbb{Z}^+ < \text{st} < \perp$. Observe that for $\text{stsize} + LZ + 1 \leq i \leq \text{stsize} + 2LZ$, $A[i]$ must contain a dummy block.
2. **Determining final offset of real blocks.** The blocks that are going to be written back in the path P can come from only $A[1..LZ]$, which is sorted according to the field **offset**, that currently indicates the lowest index at which the block can reside in A . The purpose of this step is to update the **offset** field to the final position of each real block within A before being written back to the path P . This can be achieved by linear scan as described in Figure 4 in Section 3.1. However, a naive implementation will incur a circuit depth of $\Theta(LZ)$. We shall describe a more careful implementation using divide and conquer with circuit depth of $O(\log LZ)$.

A naive way to implement linear scan will incur a circuit depth of $\Theta(LZ) = \Theta(\log N)$. We shall describe a more careful implementation such that the circuit depth is dominated by that for oblivious sorting. Recall the input of the problem is an array $A[1..m]$ that is sorted according to the **offset** field, which indicates the smallest integer that can be received by the entry. The problem is equivalent to increasing the **offset** field of each entry as little as possible such that the array A is still sorted according to **offset** and no two entries have the same **offset** value.

The high level idea is to use divide and conquer. The standard procedure is to first solve the problem recursively on $A[1..j]$ and $A[j+1..m]$, where $j = \lfloor \frac{m}{2} \rfloor$. Observe that the **offset** fields of entries in $A[1..j]$ do not have to be changed, and in order to achieve $O(1)$ circuit depth, we need to update

the entries $A[j+1..m]$ in parallel.

Observe that at this point, the next available integer for $A[j+1]$ is $p := A[j].\text{offset} + 1$. Hence, $r := \max\{p - A[j+1].\text{offset}, 0\}$ is the amount that we need to increase $A[j+1].\text{offset}$. The issue is whether we are able to deduce the increment readily for entries $A[i]$, for all $j+1 \leq i \leq m$.

For such an i , observe that $r_i := (A[i].\text{offset} - A[j+1].\text{offset}) - (i - j - 1)$ is the number of integers in $[A[j+1].\text{offset}..A[i].\text{offset}]$ that have not been assigned to any entry. Hence, the amount we need to increase $A[i].\text{offset}$ is $\max\{r - r_i, 0\}$. Hence, this step can be done in parallel for all $j+1 \leq i \leq m$. The whole procedure is achieved by calling $\text{UpdateOffset}(A[1..m])$, whose pseudocode is given in Algorithm 2. A standard analysis shows that this leads to a circuit of size $O(m \log m)$ and depth $O(\log m)$.

Algorithm 2 $\text{UpdateOffset}(A[a..b])$ using divide and conquer

```

1: if a=b then return;
2:  $j := \lfloor \frac{a+b}{2} \rfloor$ ;
3:  $\text{UpdateOffset}(A[a..j])$ ;
4:  $\text{UpdateOffset}(A[j+1..b])$ ;
5:  $r := \max\{A[j].\text{offset} + 1 - A[j+1].\text{offset}, 0\}$ ;
6:  $s := A[j+1].\text{offset}$ ;
7: for  $i$  from  $j+1$  to  $b$  in parallel do
8:    $A[i].\text{offset} := A[i].\text{offset} +$ 
      $\max\{r - (A[i].\text{offset} - s) + (i - j - 1), 0\}$ ;
9: return;
```

C. A DESCRIPTION OF ALL EXPERIMENTS

In this section, we describe every ORAM experiment that we report in this paper in order to facilitate reproducible experiments. The details is summarized in Figure 5

ORAM design	N	Other Parameters		Gates	Inputs
		Z	Stash, Evict, ℓ	(M)	(M)
Binary Tree ORAM	2^{20}	120	N/A, 2, 4	38.5	7.0
	2^{29}	120	N/A, 2, 8	127.7	24.2
CLP ORAM	2^{20}	4	120, 2, 4	29.4	0.7
	2^{29}	4	144, 2, 8	121.8	2.1
naive PATH ORAM	2^{20}	4	89, 1, 4	87.3	0.1
	2^{29}	4	89, 1, 8	278.1	0.3
PATH SCORAM	2^{20}	4	89, 1, 4	37.2	0.1
	2^{29}	4	89, 1, 8	111.7	0.3
SCORAM	2^{20}	6	88, 4, 4	4.4	0.3
	2^{29}	6	141, 4, 8	13.0	0.9

Table 5: **A listing of all parameters used in our experiments.** The parameters are set to achieve statistical security of 2^{-80} . Gates and Inputs are obtained with payload bitlength = 32bits; Cutoff Threshold is set at 2^{10}

Using SMT Solvers to Automate Design Tasks for Encryption and Signature Schemes*

Joseph A. Akinyele
Johns Hopkins University
Baltimore, MD, USA
akinyelj@cs.jhu.edu

Matthew Green
Johns Hopkins University
Baltimore, MD, USA
mgreen@cs.jhu.edu

Susan Hohenberger
Johns Hopkins University
Baltimore, MD, USA
susan@cs.jhu.edu

ABSTRACT

Cryptographic design tasks are primarily performed by hand today. Shifting more of this burden to computers could make the design process faster, more accurate and less expensive. In this work, we investigate tools for programmatically altering existing cryptographic constructions to reflect particular design goals. Our techniques enhance both security and efficiency with the assistance of advanced tools including Satisfiability Modulo Theories (SMT) solvers.

Specifically, we propose two complementary tools, AutoGroup and AutoStrong. AutoGroup converts a pairing-based encryption or signature scheme written in (simple) symmetric group notation into a specific instantiation in the more efficient, asymmetric setting. Some existing symmetric schemes have hundreds of possible asymmetric translations, and this tool allows the user to optimize the construction according to a variety of metrics, such as ciphertext size, key size or computation time. The AutoStrong tool focuses on the security of digital signature schemes by automatically converting an existentially unforgeable signature scheme into a *strongly* unforgeable one. The main technical challenge here is to automate the “partitioned” check, which allows a highly-efficient transformation.

These tools integrate with and complement the AutoBatch tool (ACM CCS 2012), but also push forward on the complexity of the automation tasks by harnessing the power of SMT solvers. Our experiments demonstrate that the two design tasks studied can be performed automatically in a matter of seconds.

1. INTRODUCTION

Cryptographic design is challenging, time consuming and mostly performed by hand. A natural question to ask is: to what extent can computers ease this burden? Which

*This work was partially supported by DARPA and the Air Force Research Laboratory (AFRL) under contract FA8750-11-2-0211. This document is a pre-print for DARPA and not for public distribution.

common design tasks can computers execute faster, more accurately or less expensively?

In particular, this work investigates tools for programmatically altering existing cryptographic constructions in order to enhance efficiency or security design goals. For instance, digital signatures, which are critical for authenticating data in a variety of settings, ranging from sensor networks to software updates, come in many possible variations based on efficiency, functionality or security. Unfortunately, it is often infeasible or tedious for humans to document each possible optimal variation for each application. It would be enormously valuable if there could be a small number of simple ways to present a scheme – as simple as possible to avoid human-error in the design and/or verification process – and then computers could securely provide any variation that may be required by practitioners.

A simple, motivating example (which we explore in this work) is the design of pairing-based signature schemes, which are often presented in a simple “symmetric” group setting that aids in exposition, but does not map to the specific pairing-based groups that maximize efficiency. Addressing this disconnect is ripe for an automated tool.

Summary of Our Contributions In this work, we explore two novel types of design problems for pairing-based cryptographic schemes. The first tool (AutoGroup) deals with efficiency, while the second (AutoStrong) deals with security. We illustrate how they interact in Figure 1. The tools take a Scheme Description Language (SDL) representation of a scheme (and optionally some user optimization constraints) and output an SDL representation of the altered scheme. This SDL output can be run through another tool or a Code Generator to produce C++ or Python software.

A contribution of this work is that we integrated our tools with the publicly-available source code for AutoBatch [3, 2] (ACM CCS 2012), a tool that automatically identifies a batch verification algorithm for a given signature scheme, therein weaving together a larger automation system. For instance, a practitioner could take any symmetric-pairing signature scheme from the literature, use AutoGroup to reduce its bandwidth in the asymmetric setting, use AutoBatch to reduce its verification time, and then automatically obtain a C++ implementation of the optimized construction. Our work appears unique in that we apply advanced tools, such as SMT solvers and Mathematica, to perform complex design tasks related to pairing-based schemes.

Automated Task 1: Optimize Efficiency of an Encryption or Signature Scheme via User Constraints.

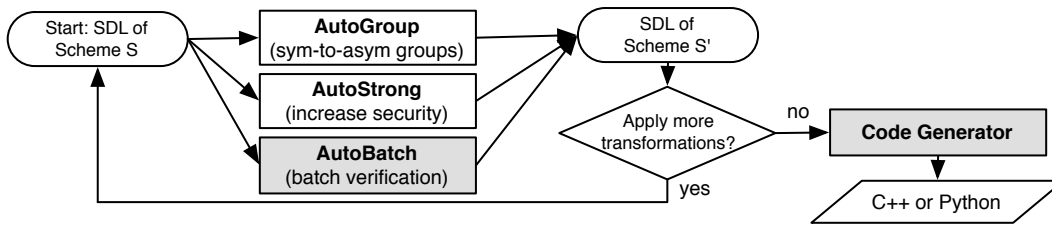


Figure 1: A high-level presentation of the new automated tools, AutoGroup and AutoStrong. They take as input a Scheme Description Language (SDL) representation of a cryptographic scheme and output an SDL representation of a transformation of the scheme, which can possibly be further transformed by another tool. These tools are compatible with the existing AutoBatch tool and Code Generator (shaded). An SDL input to the Code Generator produces a software implementation of the scheme in either C++ or Python.

Pairings are often studied because they can realize new functionalities, e.g., [18, 16], or offer low-bandwidth solutions, e.g., [20, 16]. Pairing (a.k.a., bilinear) groups consist of three groups $\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T$ with an efficient bilinear map $e : \mathbb{G}_1 \times \mathbb{G}_2 \rightarrow \mathbb{G}_T$. Many protocols are *presented* in a *symmetric* setting where $\mathbb{G}_1 = \mathbb{G}_2$ (or equivalently, there exists an efficient isomorphism from $\mathbb{G}_1$ to $\mathbb{G}_2$ or vice versa).

While symmetric groups simplify the description of new cryptographic schemes, the corresponding groups are rarely the most efficient setting for implementation [31]. The state of the art is to use *asymmetric* groups where $\mathbb{G}_1 \neq \mathbb{G}_2$ and no efficient isomorphism exists between the two. See for instance the work of Ramanna, Chatterjee and Sarkar [50] (PKC 2012) which translates the dual system encryption scheme of Waters [57] from the symmetric to a handful of asymmetric settings.

Such conversions currently require manual analysis (of all steps) – made difficult by the fact that certain operations such as group hash functions only operate in a single group. Moreover, in some cases, there are hundreds of possible symmetric to asymmetric translations, making it tedious to identify the optimal translation for a particular application.

We propose a tool called AutoGroup that automatically provides a “basic” translation from symmetric to asymmetric groups.¹ It employs an SMT solver to identify valid group assignments for all group elements and also accepts user constraints to optimize the efficiency of the scheme according to a variety of metrics, including signature/ciphertext size, signing/encryption time, and public parameter size. The tool is able to enumerate the full set of possible solutions (which may run to the hundreds), and can rapidly identify the most efficient solution.

Automated Task 2: Strengthen the Security of a Digital Signature Scheme. Most signature schemes are presented under the classic, existential unforgeability definition [34], wherein an adversary cannot produce a signature on a “new” message. However, *strong* unforgeability guarantees more – that the adversary cannot produce a “new”

¹By “basic”, we mean that it translates the scheme as written into the asymmetric setting, with minor optimizations performed, but does not attempt a re-imagining of the construction based on a stronger asymmetric complexity assumption. While the latter is sometimes possible, e.g., [50], it may not be required in some applications and the novel security analysis required places it beyond the current ability of our automation tools. See Section 3.3 for more.

signature even on a previously signed message. Strongly-unforgeable signatures are often used as a building block in signcryption [6], chosen-ciphertext secure encryption [27, 24] and group signatures [7, 17].

There are a number of general transformations from classic to strong security [54, 36, 53, 14, 55, 15], but also a highly-efficient transformation due to Boneh, Shen and Waters [21] that only applies to “partitioned” schemes. We propose a tool called AutoStrong that automatically decides whether a scheme is “partitioned” and then applies BSW if it is and a general transformation otherwise. The partitioned test is non-trivial, and our tool harnesses the power of both an SMT solver and Mathematica to make this determination. We are careful to err only on false negatives (which impact efficiency), but not false positives (which could compromise security.) Earlier works [14, 15] claimed that there were “very few” examples of partitioned schemes; however, our tool proved this was not the case by identifying valid partitions for most schemes we tested.

1.1 Related Work

Many exciting works have studied how to automate various cryptographic tasks. Automation has been introduced into the design process for various security protocols [39, 52, 49, 40, 37], optimizations to software implementations involving elliptic-curves [10] and bilinear-map functions [48], the batch verification of digital signature schemes [3], secure two-party computation [41, 42, 35], and zero-knowledge proofs [22, 8, 9, 5, 43].

Our current work is most closely related to the AutoBatch tool [3]. We borrow our tool-naming system from their paper and designed our tools so that they can integrate with the publicly-available source code of AutoBatch [2] to form a larger, more comprehensive solution. This work is different from AutoBatch in that it attacks new, more complicated design tasks and integrates external SMT solvers and Mathematica to find its solutions.

Prior work on automating the writing and verification of cryptographic proofs, such as the EasyCrypt work of Barthe et al. [13], are complimentary to but distinct from our effort. Their goal was automating the construction and verification of (game-based) cryptographic *proofs*. Our goal is automating the construction of cryptographic *schemes*. A system that combines both to automate the design of a scheme and then automate its security analysis would be optimal.

2. TOOLS USED

Our automations make use of three external tools. First, Z3 [25, 46] is a freely-available, state-of-the-art and highly efficient Satisfiability Modulo Theories (SMT) solver produced by Microsoft Research. SMT is a generalization of boolean satisfiability (SAT) solving, which determines whether assignments exist for boolean variables in a given logical formula that evaluates the formula to *true*. SMT solvers builds on SAT to support many rich first-order theories such as equality reasoning, arithmetic, and arrays. In practice, SMT solvers have been used to solve a number of constraint-satisfaction problems and are receiving increased attention in applications such as software verification, program analysis, and testing. Z3 in particular has been used as a core building block in API design tools such as Spec#/Boogie [11, 26] and in verifying C compilers such as VCC.

We leverage Z3 v4.3.1 to perform reasoning over statements involving arithmetic, quantifiers, and uninterpreted functions. We use Z3’s theories for equality reasoning combined with the decision procedures for linear arithmetic expressions and elimination of universal quantifiers (*e.g.*, $\forall x$) over linear arithmetic. Z3 includes support for uninterpreted (or *free*) functions which allow any interpretation consistent with the constraints over free functions and variables.

Second, we utilize the development platform provided by Wolfram Research’s Mathematica [59] (version 9), which allows us to simplify equations for several of our analytical techniques. We leverage Mathematica in our automation to validate that given cryptographic algorithms have certain mathematical properties. Finally, we utilize some of the publicly-available source code of the AutoBatch tool [2], including its Scheme Description Language (SDL) parser and its Code Generator, which translates an SDL representation to C++ or Python.

3. AUTOGROUP

In this section, we present and evaluate a tool, called AutoGroup, for automatically altering a cryptographic scheme’s algebraic setting to optimize for efficiency.

3.1 Background on Pairing Groups

Let $\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T$ be algebraic groups of prime order p .² We say that $e : \mathbb{G}_1 \times \mathbb{G}_2 \rightarrow \mathbb{G}_T$ is a pairing (a.k.a., bilinear map) if it is: efficiently-computable, (*bilinear*) for all $g \in \mathbb{G}_1$, $h \in \mathbb{G}_2$ and $a, b \leftarrow \mathbb{Z}_p$, $e(g^a, h^b) = e(g, h)^{ab}$; and (*non-degenerate*) if g generates $\mathbb{G}_1$ and h generates $\mathbb{G}_2$, then $e(g, h) \neq 1$. This is called the *asymmetric* setting. A specialized case is the *symmetric* setting, where $\mathbb{G}_1 = \mathbb{G}_2$.³

In practice, all candidate constructions for pairing groups are constructed such that $\mathbb{G}_1$ and $\mathbb{G}_2$ are groups of points on some elliptic curve E , and $\mathbb{G}_T$ is a subgroup of a multiplicative group over a related finite field. The group of points on E defined over $\mathbb{F}_p$ is written as $E(\mathbb{F}_p)$. Usually $\mathbb{G}_1$ is a subgroup of $E(\mathbb{F}_p)$, $\mathbb{G}_2$ is a subgroup of $E(\mathbb{F}_{p^k})$ where k is the embedding degree, and $\mathbb{G}_T$ is a subgroup of $\mathbb{F}_{p^k}^*$. In the symmetric case $\mathbb{G}_1 = \mathbb{G}_2$ is usually a subgroup of $E(\mathbb{F}_p)$.

The challenge in selecting pairing groups is to identify parameters such that the size of $\mathbb{G}_T$ provides acceptable se-

curity against the MOV attack [44]. Hence the size of p^k must be comparable to that of an RSA modulus to provide the same level of security – hence elements of $\mathbb{F}_{p^k}$ must be of size approximately 3,072 bits to provide security at the 128-bit symmetric equivalent level. The group order q must also be large enough to resist the Pollard- ρ attack on discrete logarithms, which means in this example $q \geq 256$.

Two common candidates for implementing pairing-based constructions are supersingular curves [30, 47] in which the embedding degree k is ≤ 6 and typically smaller (an example is $|p| = 1536$ for the 128-bit security level at $k = 2$), or ordinary curves such as MNT or Barreto-Naehrig (BN) [12]. In BN curves in particular, the embedding degree $k = 12$, thus $|p| = |q|$ can be as small as 256 bits at the 128-bit security level, with a corresponding speedup in field operations.

A challenge is that the recommended BN subgroups do not possess an efficiently-computable isomorphism from $\mathbb{G}_1$ to $\mathbb{G}_2$ or vice versa, which necessitates re-design of some symmetric cryptographic protocols. A related issue is that BN curves permit efficient hashing only into the group $\mathbb{G}_1$. This places strict restrictions on the set of valid group assignments we can use.

3.2 How AutoGroup Works

AutoGroup is a new tool for automatically translating a pairing-based encryption or signature scheme from the symmetric-pairing setting to the asymmetric-pairing setting. At a high-level, AutoGroup takes as input a representation of a cryptographic protocol (*e.g.*, signature or encryption scheme) written in a Domain-Specific Language called Scheme Description Language (SDL), along with a description of the optimizations desired by the user. These optimizations may describe a variety of factors, *e.g.*, requests to minimize computational cost, key size, or ciphertext/signature size. The tool outputs a new SDL of the scheme, that represents the optimal assignment of groups for the given constraints. The assignment of groups is non-trivial, as many schemes are additionally constrained by features of common asymmetric bilinear groups settings, most notably, restrictions on which groups admit efficient hashing. At a high level, AutoGroup works by reducing this constrained group assignment problem to a boolean satisfiability problem, applying an SMT solver, and processing the results. We next describe the steps of AutoGroup, as illustrated in Figure 2.

1. Extract Generator Representation. The first stage of the AutoGroup process involves parsing SDL to identify all base generators of $\mathbb{G}$ that are used in the scheme. For each generator $g \in \mathbb{G}$, AutoGroup creates a pair of generators $g_1 \in \mathbb{G}_1$ and $g_2 \in \mathbb{G}_2$. This causes an increase in the parameter size of the scheme, something that we must address in later steps.

We assume the Parser knows the basic structure of the scheme, and can identify the algorithm responsible for parameter generation. This allows us to parse the algorithm to observe which generators that are created. When AutoGroup detects the first generator, it marks this as the “base” generator of $\mathbb{G}$ and splits g into a pair $g_1 \in \mathbb{G}_1$ and $g_2 \in \mathbb{G}_2$. Every subsequent group element sampled by the scheme is defined in terms of the base generators. For example, if the setup algorithm next calls for “choosing a random generator h in $\mathbb{G}$ ”, then AutoGroup will select a random $t' \in \mathbb{Z}_p$ and compute new elements $h_1 = g_1^{t'}$ and $h_2 = g_2^{t'}$.

²Pairing groups may also have composite order, but we will be focusing on the more efficient prime order setting here.

³An alternative instantiation of the symmetric setting has $\mathbb{G}_1 \neq \mathbb{G}_2$ but admits an efficiently-computable isomorphism between the groups.

this variable during the encryption operation. The overall cost value then determines the cost of placing a value in one of the two groups – for size-related calculations, this roughly corresponds to the length of a group element’s representation, and for operation time it corresponds to the cost of a single group operation. By assigning these costs correctly, we are able to create a series of different weight functions that represent all of the different values that we would like to minimize (e.g., ciphertext size, parameter size, time).

If the user chooses to optimize for multiple criteria simultaneously, we must find a model that balances between all of these at the same time. This is not always possible. For example, some schemes admit solutions that favor a minimized secret key size or ciphertext size, but not both. In this case, we allow the user to determine which constraint to relax and thereby select the next best solution that satisfies their requirements.

7. Evaluate and Process the Solution. Once the application-specific solution is obtained from the solver, the next step is to apply the solution to produce an asymmetric scheme. As indicated earlier, we interpret the solution for each variable as $0 = \mathbb{G}_1$ and $1 = \mathbb{G}_2$. To apply the solution, we first pre-process each algorithm in SDL to determine how the pairing inputs are affected by each assignment. Consider a simplistic example: $e(A, B)$ where $A = g^a$ and $B = h^b$. Let us assume that the satisfying solution is that $A \in \mathbb{G}_1$ and $B \in \mathbb{G}_2$. Therefore, we would rewrite these two variables as $A = g_1^a$ and $B = h_2^b$ where $g_1 \in \mathbb{G}_1$ and $h_2 \in \mathbb{G}_2$. The program slice recorded for each pairing input in step (2) provides the necessary information to correctly rewrite the scheme in the asymmetric setting.

In addition to rewriting the scheme, AutoGroup performs several final optimizations. First, it removes any unused parameter values in the public and secret keys. For signature schemes, we try to optimize further by reducing the public parameters used per algorithm. In particular, we trace which variables in the public key are actually used during signing and verification. For elements that appear only in the signing (resp. decryption) algorithms, we split the public key into two: one is kept just for computing signatures (resp. decryption), and the other is given out for use in encryption/verification. Second, AutoGroup performs an additional efficiency check and attempts to optimize pairing product equations to use as few pairings as possible. This is due to the decoupling of pairings in earlier phases of translating the scheme to the asymmetric setting or perhaps, just a loose design by the original SDL designer. In either case, we apply pairing optimization techniques from previous work [3, 28] to provide this automatic efficiency check. Finally, AutoGroup outputs a new SDL of the modified scheme.

We do not offer the efficiency check of AutoGroup as a standalone tool for symmetric groups at present, because our experience inclines us to believe that most practitioners concerned with efficiency will want to work in asymmetric groups. However, our results herein also demonstrate that a simple tool of this sort is efficient and feasible.

3.3 Security Analysis of AutoGroup

Whether a scheme is translated by hand (as is done today [50]) or automatically (as in this work), a completely separate question applying to both is: is the resulting asym-

metric scheme secure? The answer is not immediately clear. Unlike the signature transformation that we automate in Section 4 that already has an established security proofs showing that the transformations preserve security, the theoretical underpinnings of symmetric-to-asymmetric translations are less explored. Here are some things we can say.

First, the original proof of security is under a symmetric pairing assumption, and thus can no longer immediately apply since the construction and assumption are changing their algebraic settings. This would seem to require the identification of a new complexity assumption together with a new proof of security. In many examples, e.g., [20], the new assumption and proof are only minor deviations from the original ones, e.g., where the CDH assumption in $\mathbb{G}$ (given (g, g^a, g^b) , compute g^{ab}) is converted in a straight-forward manner to the co-CDH assumption in $(\mathbb{G}_1, \mathbb{G}_2)$ (given (g_1, g_2, g_2^a) , compute g_1^a). However, there could be cases where a major change is required to the proof of security. For instance, in some asymmetric groups it is not possible to hash into $\mathbb{G}_2$, but in these groups there exists an isomorphism from $\mathbb{G}_2$ to $\mathbb{G}_1$. In other groups there is no such isomorphism, but it is possible to hash into $\mathbb{G}_2$. So if a scheme requires both for the security proof, that scheme may not be realizable in the asymmetric setting (see [31] for more).

In best practices today, a human first devises the new construction (based on their desired optimizations) and then the human works to identify the new assumption and proof. Our current work automates the first step in this process, and hopefully gives the human more time to spend on the second step. In this sense, our automation is arguably faster, and no less secure than what is done by hand today.

However, a more satisfactory solution requires a deeper theoretical study of symmetric-to-asymmetric pairing translations, which we feel is an important open problem, but which falls outside the scope of the current work. What can one prove about the preservation of security in symmetric-to-asymmetric translations? Is it necessary to dig into the proof of security? Or could one prove security of the asymmetric scheme solely on the assumption of security of the symmetric one? Will this work the same for encryption, signatures and other protocols? Do the rules by which translations are done (by hand or AutoGroup) need to change based on these findings? These questions remain open.

3.4 Experimental Evaluation of AutoGroup

To determine the effectiveness of our automation, we evaluate several encryption and signature schemes on a variety of optimization combinations supported by our tool. We summarize the results of our experiments on encryption schemes in Figure 3 and signature schemes in Figure 4.

System Configuration. All of our benchmarks were executed on a 2 x 2.66GHz 6-core Intel Xeon Mac Pro with 10GB RAM running Mac OS X 10.8.3 using only a single core of the Intel processor. Our implementation utilizes the MIRACL library (v5.5.4), Charm v0.43 [4] in C++ due to the efficiency gains over Python code, and Z3 SMT solver (v4.3.1). We based our implementations on the MIRACL library to fully compare each scheme’s performance using symmetric and asymmetric curves at the same security level.

Experiments and Results. To demonstrate the soundness of AutoGroup on encryption and signature schemes, we compare algorithm running times, key and ciphertext/signature

sizes between symmetric and asymmetric solutions. We tested AutoGroup on a variety of optimization combinations to extract different asymmetric solutions. In each test case, AutoGroup reports all the unique solutions, obtains the best solution for given user-specified constraints, and generates the executable code of the solution in a reasonable amount of time. AutoGroup execution time on each test case is reported in Figure 6.

4. AUTOSTRONG

In this section, we present and evaluate a tool, called AutoStrong, for automatically generating a strongly-unforgeable signature from an unforgeable signature scheme.

4.1 Background on Digital Signatures

A digital signature scheme is comprised of three algorithms: key generation, signing and verification. The classic (or “regular”) security definition for signatures, as formulated by Goldwasser, Micali and Rivest [34], is called *existential unforgeability with respect to chosen message attacks*, wherein any p.p.t. adversary, given a public key and the ability to adaptively ask for a signature on any message of its choosing, should not be able to output a signature/message pair that passes the verification equation and yet where the message is “new” (was not queried for a signature), with non-negligible probability.

An, Dodis and Rabin [6] formulated *strong unforgeability* where the adversary should not only be unable to generate a signature on a “new” message, but also be unable to generate a different signature for an already signed message. Strongly-unforgeable signatures have many applications including building signcryption [6], chosen-ciphertext secure encryption systems [27, 24] and group signatures [7, 17].

Partitioned Signatures In 2006, Boneh, Shen and Waters [21] connected these two security notions, by providing a general transformation that converts any *partitioned* (defined below) existentially unforgeable signature into a strongly unforgeable one.

DEFINITION 4.1 (PARTITIONED SIGNATURE [21]). *A signature scheme is partitioned if it satisfies two properties for all key pairs (pk, sk) :*

- **Property 1:** *The signing algorithm can be broken into two deterministic algorithms F_1 and F_2 so that a signature on a message m using secret key sk is computed as follows:*
 1. *Select a random r from a suitable randomness space.*
 2. *Set $\sigma_1 = F_1(m, r, sk)$ and $\sigma_2 = F_2(r, sk)$.*
 3. *Output the signature (σ_1, σ_2) .*
- **Property 2:** *Given m and σ_2 , there is at most one σ_1 such that (σ_1, σ_2) verifies as a valid signature on m under pk .*

As one example of a partitioned scheme, Boneh et al. partition DSS [45] as follows, where x is the secret key:

$$\begin{aligned} F_1(m, r, x) &= r^{-1}(m + xF_2(r, x)) \mod q \\ F_2(r, x) &= (g^r \mod p) \mod q \end{aligned}$$

Our empirical evidence shows that many discrete-log and pairing-based signatures in the literature are partitioned.

Interestingly, some prominent prior works [14, 15] claimed that there were “few” examples of partitioned schemes “beyond Waters [56]”, even though our automation discovered several examples existing prior to the publication of these works. We conjecture that it is not always easy for a human to detect a partition.

Chameleon Hashes The BSW transform uses a *chameleon hash* [38] function, which is characterized by the nonstandard property of being collision-resistant for the signer but collision tractable for the recipient. The chameleon hash is created by establishing public parameters and a secret trapdoor. The hash itself takes as input a message m and an auxiliary value s . There is an efficient algorithm that on input the trapdoor, any pair (m_1, s_1) and any additional message m_2 , finds a value s_2 such that $\text{ChamHash}(m_1, s_1) = \text{ChamHash}(m_2, s_2)$.

Boneh et al. [21] employ a specific hash function based on the hardness of finding discrete logarithms.⁴ Since pairing groups also require the DL problem to be hard, this chameleon hash does not add any new complexity assumptions. It works as follows in $\mathbb{G}$, where g generates $\mathbb{G}$ of order p . To setup, choose a random trapdoor $t \in \mathbb{Z}_p^*$ and compute $h = g^t$. The public parameters include the description of $\mathbb{G}$ together with g and h . The trapdoor t is kept secret. To hash on input $(m, s) \in \mathbb{Z}_p^2$, compute

$$\text{ChamHash}(m, s) = g^m h^s.$$

Later, given any pair m, s and any message m' , anyone with the trapdoor can compute a consistent value $s' \in \mathbb{Z}_p$ as

$$s' = (m - m')/t + s$$

such that $\text{ChamHash}(m, s) = \text{ChamHash}(m', s')$.

The BSW Transformation The transformation [21] is efficient and works as follows. Let $\Pi_p = (\text{Gen}_p, \text{Sign}_p, \text{Verify}_p)$ be a partitioned signature, where the signing algorithm is partitioned using functions F_1 and F_2 . Suppose the randomness for Sign_p is picked from some set R . Let $\parallel$ denote concatenation. BSW constructs a new scheme Π as:

Gen(1^λ): Select a group $\mathbb{G}$ with generator g of prime order p (with λ bits). Select a random $t \in \mathbb{Z}_p$ and compute $h = g^t$. Select a collision-resistant hash function $H_{cr} : \{0, 1\}^* \rightarrow \mathbb{Z}_p$. Run $\text{Gen}_p(1^\lambda)$ to obtain a key pair (pk_p, sk_p) . Set the keys for the new system as $pk = (pk_p, H_{cr}, \mathbb{G}, g, h, p)$ and $sk = (pk, sk_p, t)$.

Sign(sk, m): A signature on m is generated as follows:

1. Select a random $s \in \mathbb{Z}_p$ and a random $r \in R$.
2. Set $\sigma_2 = F_2(r, sk_p)$.
3. Compute $v = H_{cr}(m \parallel \sigma_2)$.
4. Compute the chameleon hash $m' = g^v h^s$.
5. Compute $\sigma_1 = F_1(m', r, sk_p)$ and output the signature $\sigma = (\sigma_1, \sigma_2, s)$.

⁴Indeed, we observe that substituting an arbitrary chameleon hash could break the transformation. Suppose $H(m, s)$ ignores the last bit of s (it is easy to construct such a hash assuming chameleon hashes exist.) Then the BSW transformation using this hash would result in a signature of the form (σ_1, σ_2, s) , which is clearly not strongly unforgeable, since the last bit can be flipped.

Encryption	Keygen [•]	Time Encrypt [•]	Decrypt [•]		Approx. Size Secret Key	Ciphertext		Num. Solutions
<i>ID-Based Enc.</i>								
BB04 [16] Sym. [†]	59.9ms	64.8ms	125.4ms		3072 bits	6144 bits		
Asym. (Min. CT)*	4.8ms	7.8ms	27.6ms		2048 bits	3584 bits		4
Gentry06 [32] Sym. [†]	39.9ms	176.2ms	67.8ms		3072 bits	7680 bits		
Asym. (Min. SK)*	1.4ms	41.0ms	19.1ms		512 bits	6400 bits		4
DSE09 [57] Sym. [†]	294.6ms	286.8ms	612.8ms		13824 bits	18432 bits		
Asym. (Min. SK/CT/Exp)*	12.6ms	19.2ms	128.0ms		5376 bits	8704 bits		256
<i>Broadcast Encryption</i>								
BGW05 [19] Sym. [§3.1] [†] (<i>n</i> = 100)	1992.2ms	119.6ms	136.9ms		19200 bytes	6144 bits		
Asym. (Min. SK)*	70.4ms	25.7ms	28.5ms		3200 bytes	5120 bits		4

*Average time measured over 100 test runs and the standard deviation in all test runs were within $\pm 1\%$ of the average.

Figure 3: AutoGroup on encryption schemes under various optimization options. We show running times and sizes for several schemes generated in C++ and compare symmetric to automatically generated asymmetric implementations at the same security levels (roughly equivalent with 3072 bit RSA). For IBE schemes, we measured with ID string length at 100 bytes. For BGW, n denotes the number of users in the system.

Verify(pk, m, σ): A signature $\sigma = (\sigma_1, \sigma_2, s)$ on a message m is verified as follows:

1. Compute $v = H_{cr}(m || \sigma_2)$.
2. Compute the chameleon hash $m' = g^v h^s$.
3. Output the result of $\text{Verify}_p(pk_p, m', (\sigma_1, \sigma_2))$.

THEOREM 4.2 (SECURITY OF BSW TRANSFORM [21]). *The signature scheme $\Pi = (\text{Gen}, \text{Sign}, \text{Verify})$ is strongly existentially unforgeable assuming the underlying scheme $\Pi_p = (\text{Gen}_p, \text{Sign}_p, \text{Verify}_p)$ is existentially unforgeable, H_{cr} is a collision-resistant hash function and the discrete logarithm assumption holds in $\mathbb{G}$.*

The Bellare-Shoup Transformation The BSW transformation [21], which only works for partitioned signatures, sparked significant research interest into finding a general transformation for *any* existentially unforgeable signature scheme. Various solutions were presented in [54, 36, 53, 14, 55, 15], as well as an observation in [14] that an *inefficient* transformation was implicit in [33].

We follow the work of Bellare and Shoup [14, 15], which is less efficient than BSW and, for our case, requires a stronger complexity assumption, but works on any signature. Their approach uses *two-tier* signatures, which are “weaker” than regular signatures as hybrids of regular and one-time schemes. In a two-tier scheme, a signer has a primary key pair and, each time it wants to sign, it generates a fresh secondary key pair and produces a signature as a function of the both secret keys and the message. Both public keys are required to verify the signature. Bellare and Shoup transform any regular signature scheme by signing the signature from this scheme with a strongly unforgeable two-tier scheme. They also show how to realize a strongly unforgeable two-tier signature scheme by applying the Fiat-Shamir [29] transformation to the Schnorr identification protocol [51], which requires a one-more discrete logarithm-type assumption.

The BS transformation works as follows. Let $\Pi_r = (\text{Gen}_r, \text{Sign}_r, \text{Verify}_r)$ be a regular signature scheme and let $\Pi_t = (\text{PGen}_t, \text{SGen}_t, \text{Sign}_t, \text{Verify}_t)$ be a two-tiered strongly unforgeable scheme. A new signature scheme Π is constructed as:

Gen(1^λ): Run $\text{Gen}_r(1^\lambda) \rightarrow (pk_r, sk_r)$ and $\text{PGen}_t(1^\lambda) \rightarrow (ppk, psk)$. Output the pair $\text{PK} = (pk_r, ppk)$ and $\text{SK} = (sk_r, psk)$.

Sign(SK, m): A signature on m is generated as follows:

1. Parse SK as (sk_r, psk) .
2. Run $\text{SGen}_t(1^\lambda) \rightarrow (spk, ssk)$.
3. Sign the message and secondary key as $\sigma_1 \leftarrow \text{Sign}_r(sk_r, (spk || m))$.
4. Sign the first signature as $\sigma_2 \leftarrow \text{Sign}_t(psk, ssk, \sigma_1)$.
5. Output the signature $\sigma = (\sigma_1, \sigma_2, spk)$.

Verify(PK, m, σ): A signature $\sigma = (\sigma_1, \sigma_2, spk)$ on a message m is verified as follows:

1. Parse PK as (pk_r, ppk) .
2. If $\text{Verify}_r(pk_r, (spk || m), \sigma_1) = 0$, then return 0.
3. If $\text{Verify}_t(ppk, spk, \sigma_1, \sigma_2) = 0$, then return 0.
4. Otherwise, return 1.

THEOREM 4.3 (SECURITY OF BS TRANSFORMATION [15]). *If the input scheme is existentially unforgeable, then the output signature is strongly existentially unforgeable assuming the strong unforgeability of the two-tier scheme.*

The Transformation used in AutoStrong For our purposes, we employ the following hybrid transformation combining BSW and Bellare-Shoup. On input a signature scheme, we automate the following procedure:

1. Identify a natural partition satisfying property 1 and test if it has property 2. (We allow false negatives, but not false positives. See Section 4.3.)
2. If a valid partition is found, apply the BSW transformation [21] (using SHA-256 and the DL-based chameleon hash above).
3. If a valid partition is not found, apply the Bellare-Shoup transformation [14, 15] (using the Schnorr Fiat-Shamir based two-tier scheme suggested in [15].)
4. Output the result.

The security of this transformation follows directly from the results of [21, 15] as stated in Theorems 4.2 and 4.3. *The most challenging technical part is step one: determining if a scheme is partitioned.*

Signature	Security	Time		Approx. Size		Num. Solutions
		Sign*	Verify*	Public Key	Signature	
CL04 [23] Sym. [†] Sym. [†] Asym. (Min. SIG)*	EU-CMA	169.8ms	316.6ms	3072 bits	4608 bits	2
	SU-CMA	192.0ms	387.8ms	4608 bits	6144 bits	
	SU-CMA	3.4ms	56.8ms	2048 bits	1024 bits	
BB Short [17] Sym. [†] Sym. [†] Asym. (Min. PK)*	EU-CMA	21.5ms	102.1ms	7680 bits	3072 bits	2
	SU-CMA	62.8ms	142.8ms	9216 bits	4608 bits	
	SU-CMA	5.0ms	18.3ms	3584 bits	1536 bits	
WATERS05 [56] Sym. [†] Sym. [†] Asym. (Min. SIG)*	EU-CMA	47.9ms	195.2ms	6144 bits	3072bits	8
	SU-CMA	88.7ms	236.4ms	7680 bits	4608 bits	
	SU-CMA	6.5ms	62.9ms	3328 bits	768 bits	
DSE09 [57] Sym. [†] Asym. (Min. PK/SIG)*	WU-CMA	258.5ms	896.8ms	23040 bits	13824 bits	256
	WU-CMA	13.6ms	129.2ms	6144 bits	5376 bits	
ACDK12 [1] Sym. [§5.3] [†] Asym. (Min. PK/SIG/Exp)*	RMA	346.4ms	1307ms	23040 bits	12288 bits	1024
	RMA	23.3ms	279.9ms	3584 bits	8192 bits	

*Average time measured over 100 test runs and the standard deviation in all test runs were within $\pm 1\%$ of the average.

Figure 4: We show the result of AutoGroup and AutoStrong on signature schemes. For CL, BB, and Waters (with length of identities, $\ell = 128$), we first apply AutoStrong to determine that the signature scheme is partitioned, then apply the BSW transform to obtain a strongly unforgeable signature in the symmetric setting. We then feed this as input to AutoGroup to realize an asymmetric variant under a given optimization. We also tested AutoStrong on the DSE signature variant and ACDK structure-preserving signature, even though these are not known to be existentially unforgeable. A partition was found for ACDK, but not DSE.

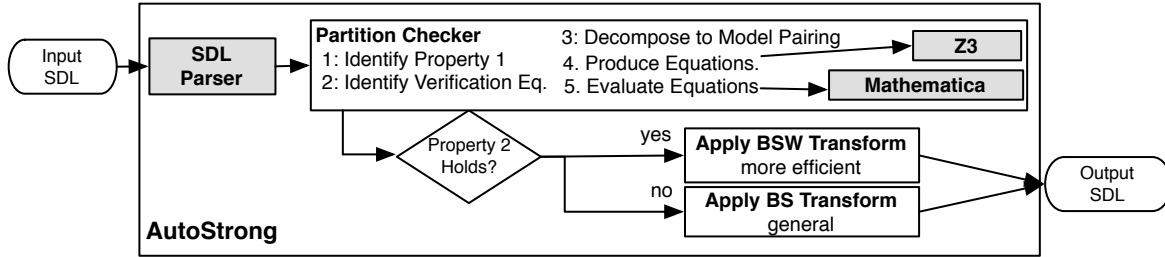


Figure 5: A high-level presentation of the tool AutoStrong, which uses external tools Z3, Mathematica and SDL Parser.

4.2 How AutoStrong Works

AutoStrong takes as input the SDL description of a digital signature scheme along with some metadata.⁵ At a high-level, it runs the transformation described at the end of the last section, where the most challenging step is testing whether a scheme is *partitioned* according to Definition 4.1.

We now describe each step involved in testing that Properties 1 and 2 are satisfied and how we utilize Z3 and Mathematica to prove such properties, as illustrated in Figure 5.

Identify Property 1. The first goal is to identify the variables in the signature that should be mapped to σ_1 or σ_2 according to Definition 4.1. We assume that the input signature scheme is existentially unforgeable.⁶ Given this assumption, our objective is to identify the portions of the signature that are computed based on the message and designate that component as σ_1 . All other variables in the signature that do not meet this criteria are designated as σ_2 . We determine that we have designated the correct variables for property 1 if and only if the variable mapping satisfy

⁵The user must specify the variables that denote message, signature, key material in a configuration file.

⁶We remark that we tested the partition checker for AutoStrong on schemes that are not existentially unforgeable to fully vet the checker (see Figure 4), but the resulting output in these cases may not be strongly unforgeable.

property 2. We test only the most “natural” division for property 1, which could result in a false negative, but this won’t impact the security, so our system allows it.

To illustrate each step, we will show how our tool identifies the partition in the CL signature scheme [23].

CL signatures [23]: Key generation consists of selecting a generator, $g \in \mathbb{G}$, then chooses $x \in \mathbb{Z}_q$ and $y \in \mathbb{Z}_q$. It sets $sk = (x, y)$ and $pk = (g, X = g^x, Y = g^y)$. The signature is computed as $\sigma = (a, b = a^y, c = a^{x+m \cdot x \cdot y})$ where a is chosen randomly in $\mathbb{G}$ and m denotes the message in $\mathbb{Z}_q$. Finally, the verification algorithm checks that $e(a, Y) = e(g, b)$ and $e(X, a) \cdot e(X, b)^m = e(g, c)$ holds.

Our logic would identify that c is dependent on the message, therefore, identifying that $\sigma_1 = c$ and $\sigma_2 = (a, b)$ which satisfies the definition of property 1. The next challenge is to determine whether property 2 holds given our identified mapping for σ_1 and σ_2 .

Prove Property 2. Proving that a scheme satisfies this property requires the ability to abstractly evaluate the verification equations on the input variables. We require this ability to automatically prove that there exists at most one σ_1 which verifies under a fixed σ_2 , m and pk for all possible inputs. To this end, the partition checker determines whether a σ'_1 exists such that $\sigma'_1 \neq \sigma_1$ and is a valid signature over the fixed variables. Finding such a σ'_1 means

Process	BB-IBE	Gentry	BGW	CL	BB Short Sig	Waters	DSE-Sig	ACDK
AutoGroup	0.33s	0.34s	0.54s	0.34s	0.31s	0.54s	4.17s	17.47s
AutoStrong	-	-	-	0.28s	0.27s	0.38s	3.25s	1.22s

Figure 6: Running time required by the AutoGroup and AutoStrong routines to process the schemes discussed in this work (averaged over 10 test runs). The running time for AutoGroup includes the running time for executing the Z3 SMT solver. The running time for AutoStrong also includes Z3 and Mathematica and the application of the BSW transformation. In all cases, the standard deviation in the results were within $\pm 3\%$ of the average. For AutoGroup, running times are correlated with the number of unique solutions found and the minimization of the weighted function using Z3. AutoStrong times are highly correlated with the complexity of the verification equations.

the signature is not partitioned. The checker determines whether it can find a solution or if *no* such solution exists. If no solutions exist, then the signature is indeed partitioned. Stated more precisely, does there exist a $\sigma'_1 \neq \sigma_1$ such that the following condition holds:

$$\text{Verify}(pk, m, (\sigma_1, \sigma_2)) = 1 \wedge \text{Verify}(pk, m, (\sigma'_1, \sigma_2)) = 1$$

At a high-level, our goal is to evaluate the pairing-based verification algorithms in a way that allows us to find a contradiction to the aforementioned condition. Recall that the bilinearity property of pairings states that $e(g^a, g^b) = e(g, g)^{ab}$ holds for all $a, b \in \mathbb{Z}_q$ where $g \in \mathbb{G}$. We observe that pairings can be modeled as an abstract function that performs multiplication in the exponent. Because the rules of multiplication and addition hold in the exponent, we can abstractly reduce pairings to basic integer arithmetic.

To accomplish this, we leverage Z3 to model the bilinearity property of pairings so that it is possible to automatically evaluate them. Our partition checker relies on Z3's uninterpreted functions and universal quantifiers to reduce pairing product equations to simpler equations over the exponents. However, this reduction alone is not sufficient to completely evaluate the verification equations as required for detecting a partitioned signature. To satisfy the property 2 condition, we also need a way to evaluate these equations on all possible inputs. Z3 was less suited for this task and instead, we employ the Mathematica scripting framework to evaluate such equations. Our solution consists of five steps:

Step 1: Decompose Verification Equations. To model pairings using an SMT solver, we encode the verification equations into a form that the solver can interpret. The first phase extracts the verification equations in SDL, then decomposes the equations in terms of the generators and exponents used. We leverage recent term rewriting extensions introduced in the SDL Parser by Akinyele et al. [3]. Their improvements keep track of how variables are computed in terms of the generators and exponents. With knowledge of how each variable is computed, we are able to fully decompose each equation in an automated fashion.

Our technique for modeling pairings in Z3 requires that decomposition of verification equations be guided by a few rules. First, generators must be rewritten in terms of a base generator, g , if the scheme is specified in the symmetric setting.⁷ For example, the random generator $a \in \mathbb{G}$ chosen in CL would be rewritten as $g^{a'}$. Second, hashing statements of the form $v = H(m)$ where $v \in \mathbb{G}$ are rewritten as g^v

⁷The same applies for asymmetric pairings except that we specify $\mathbb{G}_1$ generators in terms of a base generator g_1 and $\mathbb{G}_2$ in terms of g_2 .

where $v \in \mathbb{Z}_q$. Third, we do not decompose any variable designated as σ_1 for the purposes of determining whether a signature is partitioned. The intuition is that since σ'_1 variables are adversarially controlled we also treat σ_1 as a black box. Finally, whenever we encounter dot products, we require the user to provide an upper bound (if known) so that we can unroll the dot product then further apply our rules. When these reduction rules are automatically applied to the CL signature, we obtain the following equation:

$$e(a, Y) = e(g, b) \text{ becomes } e(g^{a'}, g^y) = e(g, (g^{a'})^y)$$

$$e(X, a) \cdot e(X, b)^m = e(g, c) \text{ becomes}$$

$$e(g^x, g^{a'}) \cdot e(g^x, (g^{a'})^y)^m = e(g, g^{c'})$$

Note that c' denotes the σ_1 for CL and is a free variable. All other variables that comprise m , pk , and σ_2 are fixed.

Step 2: Encode Rules for Evaluating Pairings. Once we have decomposed the verification equation as shown above, the next step is to encode the equations in terms that Z3 can understand. After the pairing equations are rewritten entirely using the base generator, we can model the behavior of pairings by simply focusing on the exponents. To capture the bilinearity of pairings, we rely on two features in Z3: uninterpreted functions and universal quantifiers. As mentioned earlier, uninterpreted functions enable one to abstractly model a function's behavior. Our model of a pairing is an uninterpreted function, E , that takes two integer variables and has a few mathematical properties. First, we define the multiplication rule as $\forall s, t : E(s, t) = s \cdot t$. Second, we define the addition rule as $\forall s, t, u : E(s+t, u) = s \cdot u + t \cdot u$.⁸ Third, we define pairing products in terms of multiplication with E and division as subtraction.

We note that these rules are straightforward and sufficient for evaluating pairings. Moreover, by defining exponents in terms of integers, Z3 can apply all the built-in simplification rules for multiplication and addition. As a result, the solver uses these rules to reduce any pairing-based verification equation into a simpler integer equation.

To automatically encode the equations, we first simplify the decomposed pairing equation as much as possible using previous techniques [3]. Then, we convert each pairing to the modeled pairing function, E and remove the base generators. Upon simplifying and encoding the decomposed CL equations, we obtain the following:

$$e(g^{a'}, g^y) = e(g, (g^{a'})^y) \text{ becomes } E(a', y) = E(1, a' \cdot y)$$

$$e(g^x, g^{a'}) \cdot e(g^x, (g^{a'})^y)^m = e(g, g^{c'}) \text{ becomes}$$

$$E(x, a') + E(x \cdot m, a' \cdot y) = E(1, c')$$

⁸Note that same rule applies if $E(s, t + u)$

Step 3: Execute SMT Solver. After encoding the pairing functions in terms of E , the goal is to employ the solver to evaluate it. We first specify our rules in the SMT solver then evaluate these rules on each input equation. The result is a simplified equation representation of the verification algorithm. For the above CL formulas, the solver determines that the first equation is *true* for all possible inputs because a' and y are fixed variables. For the second equation, the solver produces: $a' \cdot x + a' \cdot m \cdot x \cdot y = c'$.

Step 4: Evaluate equations. At this point, we have obtained the equation representation of the verification equation, we can now concretely express the conditions for property 2. That is, $c' \neq c'' \wedge$
 $a' \cdot x + a' \cdot x \cdot m \cdot y = c' \wedge a' \cdot x + a' \cdot x \cdot m \cdot y = c''$

We use Mathematica to prove that no such c'' exists assuming the verification condition is correct via the Mathematica Script API. In particular, we utilize the *FindInstance* function to mathematically find proof over non-zero real numbers then subsequently finding a solution over integers. If *no* such solution exists, the *FindInstance* will return such a statement and the result is interpreted as a fact that the signature scheme is partitioned. Otherwise, the scheme may not be partitionable.

During this step, we make an explicit assumption that the verification condition is mathematically correct. Suppose that this was not the case. In this scenario, our technique would also determine that it is not possible to find a σ_1' such that $\sigma_1' \neq \sigma_1$ and verifies over fixed variables. In reality, however, no σ_1 and σ_2 pair can ever produce a valid signature because the verification equation does not hold for *any* input. To limit the possibility of such scenarios, our partition checker offers a sanity check on the correctness of the input verification equations.

By relaxing the rule for decomposing the variables that are designated as σ_1 in step 1, we can evaluate the verification equation over all inputs using Mathematica. For the CL scheme, a full decomposition would produce the following equation in the exponent:

$$a' \cdot x + a' \cdot x \cdot m \cdot y = a' \cdot (x + x \cdot m \cdot y)$$

It is sufficient to leverage the *Simplify* function within Mathematica to evaluate that this holds for all possible inputs. Since Mathematica has built-in techniques for solving equations of this sort, it becomes trivial to show that the above equation is correct in all cases (due to the law of distribution). We subsequently inform the user on the output of this sanity check. This sanity check is useful for determining the correctness of SDL signature descriptions.

Step 5: Apply Transformation. Once the partition checker determines whether the signature is partitioned or not, we apply the efficient BSW transform if deemed partitioned or the less-efficient BS transform if not as described in Section 4.1. We elaborate further in the full version.

4.3 Security Analysis of AutoStrong

The theoretical security of the unforgeable-to-strongly-unforgeable transformations that we use in AutoStrong were previously established in [21, 14, 15], as discussed in Section 4.1. The security of the BSW transform only holds, however, if the input scheme is partitioned. Our partition test allows false negatives, but not false positives. That is, our algorithm may fail to identify a scheme as partitioned

even though it is, which results in a less efficient final scheme, but it will not falsely identify a scheme as partitioned when it is not, which would result in a security failure. To see why this claim holds, consider that the partition tester guesses a partition, Z3 to interpret the verification equation as a system of equations, and then Mathematica fixes the variables on one partition side and asks how many solutions there are for the free variables on the other side. If 0 or 1 are found, then the scheme meets the partitioned definition. If more than 1 is found, then it is not partitioned. If there is no answer (program crash or times out), then we consider it not partitioned. Thus, false negatives can occur, but not false positives (in theory). Proving that there are no software or hardware errors in AutoStrong, Z3, Mathematica or the underlying software and hardware on which they run is outside the scope of this project. However, we did verify AutoStrong’s outputs experimentally on a suite of test cases and no errors were found.

4.4 Experimental Evaluation of AutoStrong

In 2008 [15], Bellare and Shoup remarked that “unfortunately, there seem to be hardly any [partitioned signature] schemes”. Interestingly, our experimental results show that there are in fact many partitioned schemes, including a substantial number invented prior to 2008. We evaluated AutoStrong by testing it on a collection of signatures, including the bilinear Camenisch-Lysyanskaya signature [23], the Boneh-Boyen short signature [16], the Waters signature from CRYPTO 2005 [56], the Waters Dual-System (DSE) signature from CRYPTO 2009 [57], and a structure-preserving signature scheme due to Abe *et al.* [1].

Of the above signatures, all but one – the Waters DSE signature – were successfully partitioned, leading to strongly unforgeable signatures via the BSW transform.⁹ Figure 6 shows the time that it took our tool to identify the partitioning and output the revised signature equations. Figure 4 illustrates the performance and size of the resulting signatures, when evaluated on two different types of curve (using AutoGroup to calculate the group assignments).

5. CONCLUSION

We explored two new tasks in cryptographic automation. First, we presented a tool, AutoGroup, for automatically translating a symmetric pairing scheme into an asymmetric pairing scheme. The tool allows the user to choose from a variety of different optimization options. Second, we presented a tool, AutoStrong, for automatically altering a digital signature scheme to achieve *strong unforgeability* [6]. The tool automatically tests whether a scheme is “partitioned” according to a notion of Boneh *et al.* [21] and then applies a highly-efficient transformation if it is partitioned or a more general transformation otherwise. To perform some of these complex tasks, we integrated Microsoft’s SMT Solver Z3 and Mathematica into our tools. Our performance measurements indicated that these standard cryptographic design tasks can be quickly, accurately and cost-effectively performed by computers. We leave open the question of which other design tasks are well suited for SMT solvers.

⁹However, we note that the partitioning of the Abe *et al.* signature scheme destroys the structure-preserving property, making this result somewhat less interesting.

6. REFERENCES

- [1] Masayuki Abe, Melissa Chase, Bernardo David, Markulf Kohlweiss, Ryo Nishimaki, and Miyako Ohkubo. Constant-size structure-preserving signatures: Generic constructions and simple assumptions. Cryptology ePrint Archive, Report 2012/285, 2012. <http://eprint.iacr.org/>.
- [2] Joseph A. Akinyele, Matthew Green, Susan Hohenberger, and Matthew W. Pagano. AutoBatch Toolkit. <https://github.com/jhuisi/auto-tools>.
- [3] Joseph A. Akinyele, Matthew Green, Susan Hohenberger, and Matthew W. Pagano. Machine-generated algorithms, proofs and software for the batch verification of digital signature schemes. In *ACM CCS*, pages 474–487, 2012.
- [4] Joseph A. Akinyele, Matthew Green, and Avi Rubin. Charm: A framework for rapidly prototyping cryptosystems. Cryptology ePrint Archive, Report 2011/617, 2011. <http://eprint.iacr.org/>.
- [5] José Bacelar Almeida, Endre Bangerter, Manuel Barbosa, Stephan Krenn, Ahmad-Reza Sadeghi, and Thomas Schneider. A certifying compiler for zero-knowledge proofs of knowledge based on σ -protocols. In *ESORICS'10*, pages 151–167, 2010.
- [6] Jee Hea An, Yevgeniy Dodis, and Tal Rabin. On the security of joint signature and encryption. In *EUROCRYPT*, volume 2332, pages 83–107, 2002.
- [7] Giuseppe Ateniese, Jan Camenisch, Marc Joye, and Gene Tsudik. A practical and provably secure coalition-resistant group signature scheme. In *CRYPTO '00*, volume 1880, pages 255–270, 2000.
- [8] Michael Backes, Matteo Maffei, and Dominique Unruh. Zero-knowledge in the applied pi-calculus and automated verification of the direct anonymous attestation protocol. In *IEEE Symposium on Security and Privacy*, pages 202–215, 2008.
- [9] Endre Bangerter, Thomas Briner, Wilko Henecka, Stephan Krenn, Ahmad-Reza Sadeghi, and Thomas Schneider. Automatic generation of sigma-protocols. In *EuroPKI'09*, pages 67–82, 2009.
- [10] M. Barbosa, A. Moss, and D. Page. Compiler assisted elliptic curve cryptography. In *OTM Conferences (2)*, pages 1785–1802, 2007.
- [11] Mike Barnett, K. Rustan M. Leino, and Wolfram Schulte. The spec# programming system: An overview. pages 49–69. Springer, 2004.
- [12] Paulo S. L. M. Barreto and Michael Naehrig. Pairing-friendly elliptic curves of prime order. In *SAC*, volume 3897, pages 319–331, 2006. <http://cryptojedi.org/papers/\#pfcpo>.
- [13] Gilles Barthe, Benjamin Grégoire, Sylvain Heraud, and Santiago Zanella Béguelin. Computer-aided security proofs for the working cryptographer. In *CRYPTO*, pages 71–90, 2011.
- [14] Mihir Bellare and Sarah Shoup. Two-tier signatures, strongly unforgeable signatures, and fiat-shamir without random oracles. In *PKC*, pages 201–216, 2007.
- [15] Mihir Bellare and Sarah Shoup. Two-tier signatures from the fiat-shamir transform, with applications to strongly unforgeable and one-time signatures. *IET Information Security*, 2(2):47–63, 2008.
- [16] Dan Boneh and Xavier Boyen. Short signatures without random oracles. In *EUROCRYPT*, volume 3027, pages 382–400, 2004.
- [17] Dan Boneh, Xavier Boyen, and Hovav Shacham. Short group signatures. In *CRYPTO '04*, volume 3152 of LNCS, pages 45–55, 2004.
- [18] Dan Boneh and Matthew K. Franklin. Identity-based encryption from the Weil pairing. In *CRYPTO*, pages 213–229, 2001.
- [19] Dan Boneh, Craig Gentry, and Brent Waters. Collusion resistant broadcast encryption with short ciphertexts and private keys. In *CRYPTO'05*, pages 258–275, 2005.
- [20] Dan Boneh, Ben Lynn, and Hovav Shacham. Short signatures from the Weil pairing. In *ASIACRYPT '01*, volume 2248 of LNCS, pages 514–532, 2001.
- [21] Dan Boneh, Emily Shen, and Brent Waters. Strongly unforgeable signatures based on computational Diffie-Hellman. In *PKC*, pages 229–240, 2006.
- [22] J. Camenisch, M. Rohe, and A.R. Sadeghi. Sokrates - a compiler framework for zero- knowledge protocols. In *Proceedings of the Western European Workshop on Research in Cryptology, WEWoRC 2005*, 2005.
- [23] Jan Camenisch and Anna Lysyanskaya. Signature schemes and anonymous credentials from bilinear maps. In *CRYPTO*, volume 3152, pages 56–72, 2004.
- [24] Ran Canetti, Shai Halevi, and Jonathan Katz. Chosen-ciphertext security from identity-based encryption. In *EUROCRYPT*, pages 207–222, 2004.
- [25] Leonardo De Moura and Nikolaj Bjørner. Z3: an efficient smt solver. In *Proceedings of the Theory and practice of Software, TACAS'08/ETAPS'08*, pages 337–340, 2008.
- [26] Robert DeLine, K. Rustan, and M. Leino. Boogie pl: A typed procedural language for checking object-oriented programs. Technical Report MSR-TR-2005-70.
- [27] Danny Dolev, Cynthia Dwork, and Moni Naor. Nonmalleable cryptography. *SIAM J. Comput.*, 30(2):391–437, 2000.
- [28] Anna Lisa Ferrara, Matthew Green, Susan Hohenberger, and Michael Østergaard Pedersen. Practical short signature batch verification. In *CT-RSA*, volume 5473 of LNCS, pages 309–324, 2009.
- [29] Amos Fiat and Adi Shamir. How to prove yourself: Practical solutions to identification and signature problems. In *CRYPTO*, pages 186–194, 1986.
- [30] Steven D. Galbraith. Supersingular curves in cryptography. In *ASIACRYPT*, pages 495–513, 2001.
- [31] Steven D. Galbraith, Kenneth G. Paterson, and Nigel P. Smart. Pairings for cryptographers, 2006. Cryptology ePrint Archive: Report 2006/165.
- [32] Craig Gentry. Practical identity-based encryption without random oracles. In *EUROCRYPT*, pages 445–464, 2006.
- [33] Oded Goldreich. *The Foundations of Cryptography - Volume 2, Basic Applications*. Cambridge University Press, 2004.
- [34] S. Goldwasser, S. Micali, and R. Rivest. A digital signature scheme secure against adaptive chosen-message attacks. *SIAM J. Comp.*, 17(2), 1988.
- [35] Wilko Henecka, Stefan Kögl, Ahmad-Reza Sadeghi,

- Thomas Schneider, and Immo Wehrenberg. Tasty: tool for automating secure two-party computations. In *ACM CCS*, pages 451–462, 2010.
- [36] Qiong Huang, Duncan S. Wong, and Yiming Zhao. Generic transformation to strongly unforgeable signatures. In *ACNS*, pages 1–17, 2007.
- [37] Shinsaku Kiyomoto, Haruki Ota, and Toshiaki Tanaka. A security protocol compiler generating C source codes. In *ISA'08*, pages 20–25, 2008.
- [38] Hugo Krawczyk and Tal Rabin. Chameleon signatures. In *NDSS*, 2000.
- [39] Gavin Lowe. Casper: a compiler for the analysis of security protocols. *J. Comput. Secur.*, 6(1-2):53–84, 1998.
- [40] Stefan Lucks, Nico Schmoigl, and Emin Islam Tatli. Issues on designing a cryptographic compiler. In *WEWoRC*, pages 109–122, 2005.
- [41] Philip MacKenzie, Alina Oprea, and Michael K. Reiter. Automatic generation of two-party computations. In *ACM CCS*, pages 210–219, 2003.
- [42] Dahlia Malkhi, Noam Nisan, Benny Pinkas, and Yaron Sella. Fairplay – a secure two-party computation system. In *USENIX Security Symposium*, pages 287–302, 2004.
- [43] Sarah Meiklejohn, C. Chris Erway, Alptekin Küpcü, Theodora Hinkle, and Anna Lysyanskaya. ZKPD: a language-based system for efficient zero-knowledge proofs and electronic cash. In *USENIX Security'10*, pages 193–206, 2010.
- [44] A. Menezes, S. Vanstone, and T. Okamoto. Reducing elliptic curve logarithms to logarithms in a finite field. In *STOC*, pages 80–89, 1991.
- [45] Alfred Menezes, Paul C. van Oorschot, and Scott A. Vanstone. *Handbook of Applied Cryptography*. CRC Press, 1996.
- [46] Leonardo Moura and Grant Olney Passmore. The strategy challenge in SMT solving. In *Automated Reasoning and Mathematics*, volume 7788, pages 15–44. 2013.
- [47] Dan Page, Nigel Smart, and Fre Vercauteren. A comparison of MNT curves and supersingular curves. *Applicable Algebra in Eng, Com and Comp*, 17(5):379–392, 2006.
- [48] Luis J. Dominguez Perez and Michael Scott. Designing a code generator for pairing based cryptographic functions. In *Pairing'10*, pages 207–224, 2010.
- [49] Davide Pozza, Riccardo Sisto, and Luca Durante. Spi2java: Automatic cryptographic protocol java code generation from spi calculus. In *Advanced Information Networking and Applications*, pages 400–, 2004.
- [50] Somindu C. Ramanna, Sanjit Chatterjee, and Palash Sarkar. Variants of Waters' dual system primitives using asymmetric pairings - (extended abstract). In *Public Key Cryptography*, pages 298–315, 2012.
- [51] Claus-Peter Schnorr. Efficient signature generation by smart cards. *J. Cryptology*, 4(3):161–174, 1991.
- [52] Dawn Xiaodong Song, Adrian Perrig, and Doantam Phan. Agvi - automatic generation, verification, and implementation of security protocols. In *Computer Aided Verification*, pages 241–245, 2001.
- [53] Ron Steinfeld, Josef Pieprzyk, and Huaxiong Wang. How to strengthen any weakly unforgeable signature into a strongly unforgeable signature. In *CT-RSA*, pages 357–371, 2007.
- [54] Isamu Teranishi, Takuro Oyama, and Wakaha Ogata. General conversion for obtaining strongly existentially unforgeable signatures. In *INDOCRYPT*, pages 191–205, 2006.
- [55] Isamu Teranishi, Takuro Oyama, and Wakaha Ogata. General conversion for obtaining strongly existentially unforgeable signatures. *IEICE Transactions*, 91-A(1):94–106, 2008.
- [56] Brent Waters. Efficient identity-based encryption without random oracles. In *EUROCRYPT '05*, volume 3494 of LNCS, pages 320–329. Springer, 2005.
- [57] Brent Waters. Dual system encryption: Realizing fully secure ibe and hibe under simple assumptions. In *CRYPTO*, pages 619–636, 2009.
- [58] Brent Waters. Functional encryption for regular languages. In *CRYPTO*, volume 7417, pages 218–235, 2012.
- [59] Wolfram. Mathematica, version 9. <http://www.wolfram.com/mathematica/>.

GPU and CPU Parallelization of Honest-but-Curious Secure Two-Party Computation*

Nathaniel Husted
Dept. of Comp. Sc.
Indiana University
nhusted@cs.indiana.edu

Steven Myers
Dept. of Comp. Sc.
Indiana University
samymers@cs.indiana.edu

abhi shelat
Dept. of Comp. Sc.
University of Virginia
abhi@cs.virginia.edu

Paul Grubbs
Dept. of Comp. Sc.
Indiana University
paulgrub@umail.iu.edu

ABSTRACT

Recent work demonstrates the feasibility and practical use of secure two-party computation [5, 9, 15, 23]. In this work, we present the first Graphical Processing Unit (GPU)-optimized implementation of an optimized Yao's garbled-circuit protocol for two-party secure computation in the honest-but-curious and 1-bit-leaked malicious models. We implement nearly all of the modern protocol advancements, such as Free-XOR, Pipelining, and OT extension. Our implementation is the first allowing entire circuits to be generated concurrently, and makes use of a modification of the XOR technique so that circuit generation is optimized for implementation on SIMD architectures of GPUs. In our best cases we generate about 75 million gates per second and we exceed the state of the art performance metrics on modern CPU systems by a factor of about 200, and GPU systems by about a factor of 2.3. While many recent works on garbled circuits exploit the embarrassingly parallel nature of many tasks that are part of a secure computation protocol, we show that there are still various forms and levels of parallelization that may yet improve the performance of these protocols. In particular, we highlight that implementations on the SIMD architecture of modern GPUs require significantly different approaches than the general purpose MIMD architecture of multi-core CPUs, which again differ from the needs of parallelizing on compute clusters. Additionally, modifications to the security models for many common protocols have large effects on reasonable parallel architectures for implementation.

*This work is supported by Defense Advanced Research Projects Agency (DARPA) and the Air Force Research Laboratory (AFRL) under contract FA8750-11-2-0211. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the Defense Advanced Research Projects Agency or the US government. This material is based upon work supported by the National Science Foundation under Grant No. 1111149.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

ACSAC '13 Dec. 9-13, 2013, New Orleans, Louisiana USA
Copyright 2013 ACM 978-1-4503-2015-3/13/12 ...\$15.00
<http://dx.doi.org/10.1145/2523649.2523681>.

1. INTRODUCTION

A company may wish to offer a generic screening service which would let patients know if they are susceptible to disease based on the presence of different proprietary markers in their DNA. In such a scenario the company does not want to divulge its proprietary markers, and the consumer does not want to divulge their genetic information in fear that it will be exploited by the company. The above problem represents a specific case of *secure two-party computation*, in which there are two parties who wish to compute a function $f : (\{0, 1\}^m)^2 \rightarrow \{0, 1\}^m$, on respective inputs $x_0, x_1 \in \{0, 1\}^m$, with the guarantee that no party learns anything beyond what can be efficiently inferred from the output.

Cryptographers have studied this problem, and have suggested solutions in various security models. However, while theoretically interesting, it was historically believed that these protocols are too inefficient for practical implementation. Work by Malkhi et al. [18] gave the first implementation of Yao's garbled-circuit protocol and, while it could perform very modest computations in a reasonable amount of time, the result seemed to validate the belief that these protocols would not be practical. This has resulted in cryptographers pursuing specific protocols to solve specific instances of secure two-party computation. For example, specific algorithms for looking at the edit distance between two strings (e.g., for the genetic problem discussed above), with the goal of making practical algorithms that could be deployed. However, recent advances and improvements to Yao-based protocols and implementations (cf. [13, 21, 8]) have shattered the belief that general purpose solutions are too inefficient to be deployed in practice. These works have led to renewed interest in practical implementations of Yao's protocol. Research now focuses on determining which problems might be solved by efficiently engineered versions of Yao's garbled circuits. Any such engineering will make use of parallel processing, as Yao's circuit protocol (and its improvements), have a high level of inherent parallelism for both the honest-but-curious and malicious security models. Importantly, there are key differences in the available parallelism available in the two security models.

In this paper, we provide a high-performance parallel implementation of Yao's circuits in the *honest-but-curious (HbC)* and *1-bit-leaked malicious model (1BM)*. The implementation is optimized for parallel processing architectures with both multi-core CPUs and GPUs. We have implemented both circuit generation and evaluation on GPUs. Additionally, on multi-core CPUs we have implemented evaluation. GPUs have shown to provide more

GFLOPS per dollar and more GFLOPS per watt than leading x86 CPUs, and the gap in performance is expected to widen. Therefore, any compute intensive task that can naturally be parallelized, such as Yao’s garbled circuit technique, needs to be investigated in this model. Further, it has previously been noted that GPUs can potentially be used as cheap, “off-the-shelf” cryptographic co-processors, and work has been done showing their use for implementing both symmetric and asymmetric cryptographic primitives, as well as for cryptanalysis.

Frederiksen and Nielsen[5] have recently produced a somewhat optimized version of Yao’s protocol in the *malicious security model* for GPUs. However, differences in Yao’s protocol for the malicious model, as compared to the ones we consider, necessitate different architectural approaches for implementation. This is particularly poignant due to the the Single Instruction Multiple Data (SIMD) architecture of the GPU: *small changes in protocols can lead to large changes in the appropriate processing units for a GPU*. Thus, our work is important as it provides a new GPU work scheduling architecture optimized for the HbC and IBM security models, which have many practical deployment scenarios.

1.1 Our Contributions

We present the first modern implementation of Yao’s for the Honest-but-Curious (HbC) and One-bit Leak Malicious (IBM) security models. There are many settings in practice, where these security models are more than sufficient for use in secure computation. Due to the fact that protocols that satisfy these weaker security models are substantially less resource demanding, it permits for either a larger array of circuits to be evaluated, or for systems to potentially be much faster.

In this paper, *we first present a new method to generate garbled circuits with Free-XORs so that generation can be entirely parallelized on the GPU*. Prior works have parallelized the generation of “layers” of the circuit, but suffered from inherent data dependencies that prevented parallelizing the generation of the entire circuit due to the Free-XOR optimization technique. By default, garbled circuits are not designed to be optimal for GPUs and SIMD computation when using the Free-XOR technique because this technique creates dependencies in XOR gate chains. A principle contribution of this work is to fix this issue at the protocol level; we are the first implementation to do so and our experimental results validate the benefits of the approach. Our resulting GPU-based implementation provides significant improvements in circuit generation speed, compared to all previous constructions, both those using GPUs and those that do not. We provide a more detailed explanation of our system in Sec. 7 and additionally discuss implementation issues for the GPU and some simple optimizations to implementations of SHA1.

Next we see that for our security models the evaluation of circuits on a relatively simple CPU implementation outperforms our GPU implementation. Therefore, any system in which the GPU is going to be used to maximal capacity for evaluation is going to need to improve the ability to fully parallelize evaluation. Specifically, evaluation of a garbled circuit has an inherent data dependency between layers; one cannot evaluate layer i of the circuit without having first evaluated all the gates which it depends on layer $i - 1$. A strong deployment will need to have a finer understanding of dependency that allows for gates in higher levels to be evaluated (if possible), before all lower level gates are evaluated. Further, our experiments suggest a reasonable architecture for Yao circuits in the HbC and IBM models may also involve a hybrid approach: use the GPU for generation and verification and then splitting evaluation between the CPU and GPU.

Overall our performance results are better or comparable to the other state of the art implementations, with results varying on the facets of circuit generation or evaluation considered. We show we can generate gates using the GPU significantly faster than the only other GPU implementation of Yao’s [5], and on a per system basis we can generate and evaluate gates on CPUs faster than Kreuter et al. [15]. Specifically, on similar top-end hardware from 2013, our system can generate roughly 74 million gates/sec, whereas [5] achieves 21 million gates/sec and Kreuter et al. achieves 0.35 million gates/sec.

1.2 Roadmap

In Section 2 we present related work. Section 3 gives a brief overview of the different security models discussed in the paper, and of Yao’s protocol and its variants. In Sections 4 and 5 we briefly introduce and discuss some of the architectural issues in GPU development and multi-core CPU development respectively. In Section 6, we discuss how the different security models induce different architectural approaches to accommodate the different types of parallelism in the underlying protocols and the resources they use. In Section 7 we detail how our system works, and our modification to the Free-XOR technique that allows for faster circuit generation. In Section 8, we provide experimental results validating our claims. Section 9 gives conclusions and discusses our current directions with this work.

2. RELATED WORK

Malkhi et al.[18] describe the first secure multi-party scheme implementing garbled circuits in the Fairplay system. Their system uses a custom circuit definition language called SFDL compiled into a machine-readable representation language called SHDL.

The first paper to consider the feasibility in practice of these schemes was Pinkas et al. [22], who implemented the first Free-XOR scheme [13] and OT Extension, and introduced the notion of Garbled Row Reduction to save communication costs. They also considered all modern security models. Huang et al. [8] improved Pinkas’ et al.’s performance by utilizing a number of enhanced construction techniques for garbled circuits including Free-XOR, oblivious-transfer extension [11], the Naor-Pinkas OT protocol [20], and introducing the notion of pipelined gate generation and evaluation. The system still used serial gate generation and evaluation, but the authors showed the potential performance benefits of well-crafted circuits. Finally, the scheme was implemented in Java which is highly portable, but suffers from the need to be run through the virtual machine.

Pu et al. [23] gave the first implementation of Yao’s circuits using a GPU. However, their implementation only used the GPU as a cryptographic co-processor to calculate symmetric encryptions (i.e. 3DES) and elliptic curve operations. *They do not attempt to use the GPU to actually build or evaluate any of the garbled circuits*. The system does not implement any of the modern algorithmic advances in garbled circuits such as Free-XORs, Pipelining, OT-extension, etc. The implementation is in the HbC security model.

Kreuter et al. [15] presented the first garbled circuit protocol that is secure against malicious adversaries and can scale to handle circuits with several billion gates. They implement all of the modern efficiency improvements to Yao’s protocol, such as Free-XOR, Pipelining, OT-extension, etc. They also introduced a circuit compiler that translated C-like code into circuits and both the compiler and Yao’s system could scale to handle several billion gates.

Huang et al.[9]present the first efficient protocol and implementation of Yao’s in the IBM model suggested by Franklin and Mo-

hassel [19], but do not consider a parallelized implementation. They implement the Free-XOR technique, garbled row reduction and pipelining.

Frederiksen and Nielsen [5] have recently implemented Yao’s on the GPU in the malicious model. Unlike the work in [23], they use the GPU not only to compute cryptographic primitives efficiently, but to generate and evaluate the circuit. They also implement modern efficiency improvements such as Free-XOR, garbled gate row reduction, OT extension. They do not implement pipelining.

Our approach is similar to the latter two works in that we exploit the parallel nature of certain subproblems of garbled circuits. However, we target the HbC and IBM security models and thus the protocols we are implementing have less inherent parallelism. This is because the malicious model adds another layer of an inherently parallelizable protocol. Thus Kreuter et al. accelerate the cut-and-choose technique in the malicious model by giving each thread (i.e., processor) a circuit. They use a large compute clusters with hundreds of nodes to run their system. In particular, their MPI implementation is optimized for a *specific type of cluster*. Their code assumes the cluster’s scheduler allocates work at the granularity of processors. Modern, high-end super computing clusters schedule at the granularity of nodes thus will not work optimally with their code. Similarly, Frederiksen and Nielsen also generate each copy of a circuit’s gate in the cut-and-choose protocol on a separate GPU core.

In contrast, our system parallelizes the generation of circuits themselves, thus each core generates distinct gates of the circuit. Indeed, doing so requires changes to the protocol itself, but nonetheless, our systems are highly complementary, and a garbled circuit implementation that is a hybrid of our techniques could provide high performance. Further, because of the differences in the protocol, communication overhead in the HbC or IBM security model is approximately 2 orders of magnitude less than these two prior works. This means that circuit generation and evaluation times are of prime importance, as compared to communication overhead as Frederiksen and Nielsen observe for the malicious model.

While there are competing approaches for constructing two-party secure computation protocols, it appears that the Yao garbled circuits approach is currently one of the fore-runner’s in performance, although the recent SPDZ system of Damgard et al.[3] performs efficiently on some forms of circuit (such as, importantly, AES). Still, Major questions remain on how to optimize Yao’s garbled circuits for speed depending on different compute models, and different security models. Solutions have currently focused on four approaches: i) Implementation Optimizations (Parallelism, pipelining), ii) Security Model Compromises (Hybrid Model), iii) Construction Optimizations (Free-XOR technique, OT Extension), and iv) Compiler Optimizations (Maximize XOR gates, minimize gate counts). This work focuses on implementation optimizations using the current best security model optimizations, construction optimizations, and compiler optimizations from the state of the art work. Specifically, we focus on parallelizing the generation and evaluation of garbled circuits so as to perform well on *single machines with GPUs*.

3. BACKGROUND

We (briefly) review Yao’s garbled-circuit approach to secure computation [25], and the respective security models of interest: honest-but-curious (HbC), malicious, and one-bit leaked malicious (IBM).

Garbled circuits provide a way for two parties, holding inputs x and y respectively, to compute an arbitrary function f of their inputs without revealing anything to either party other than the result $f(x, y)$. At a high level, the idea behind the base protocol

is that one party—the *garbled-circuit generator*—prepares an “encrypted” version of a boolean circuit for f (the *garbled circuit*) and sends it, along with an “encryption” of its input (say, x), to the second party. This other party—the *garbled-circuit evaluator*—obtains some additional information from the garbled-circuit generator (this information depends on the evaluator’s input y), and then obviously computes the output value $f(x, y)$ without learning the values on any intermediate wires of the circuit.

Security Models.

In the *honest-but-curious* model, both parties execute a protocol correctly, but the parties are willing to try and extract any extra information they can from protocol execution in other external processes. Thus, informally, a secure protocol must ensure that no extra information other than the output can be extracted or deduced in polynomial time from the protocol transcripts. A secure protocol in the *malicious* model is one that ensures the same even when an adversary deviates arbitrarily from the protocols specification. The prior two models apply rather generically to many cryptographic protocols. The final model is the *one-bit-leaked malicious* model. A secure computation protocol is secure in this model, if it is secure against malicious adversaries with the relaxation that an arbitrary predicate of the private inputs can be leaked to the adversary during any execution. Formal definitions of each of these models can be found in [6, 9].

All three of the models have legitimate practical scenarios. For example, hospitals might, in order to preserve privacy as dictated by law, determine if they have patients in common using the honest-but-curious model. Whereas, its use to securely compute in the presence of an adversary, such as a nation’s intelligence bureau, would require malicious security. Companies might use secure computation in the one-bit-leaked malicious model, when the computation is not repeated frequently, and when no bit of the data is particularly valuable.

Yao’s Protocol for the Honest-but-Curious Setting.

Given a boolean circuit for f (pre-agreed upon by the parties), the circuit generator chooses two random cryptographic keys W_i^0, W_i^1 for each wire i of the circuit. (The semantics are that W_i^0 encodes a 0-bit on the i th wire, while W_i^1 encodes a 1-bit.) In addition, for each wire i he chooses a random permutation bit π_i , and assigns key W_i^b the label $\lambda_i^b = b \oplus \pi_i$. Next, for each binary gate g of the circuit, having input wires i, j and output wire k , the circuit generator computes a *garbled gate* consisting of the following four ciphertexts (in order):

$$\begin{aligned} & \text{Enc}_{W_i^{\pi_i}, W_j^{\pi_j}}^g \left(W_k^{g(\pi_i, \pi_j)} \parallel \lambda_k^{g(\pi_i, \pi_j)} \right) \\ & \text{Enc}_{W_i^{\pi_i}, W_j^{1 \oplus \pi_j}}^g \left(W_k^{g(\pi_i, 1 \oplus \pi_j)} \parallel \lambda_k^{g(\pi_i, 1 \oplus \pi_j)} \right) \\ & \text{Enc}_{W_i^{1 \oplus \pi_i}, W_j^{\pi_j}}^g \left(W_k^{g(1 \oplus \pi_i, \pi_j)} \parallel \lambda_k^{g(1 \oplus \pi_i, \pi_j)} \right) \\ & \text{Enc}_{W_i^{1 \oplus \pi_i}, W_j^{1 \oplus \pi_j}}^g \left(W_k^{g(1 \oplus \pi_i, 1 \oplus \pi_j)} \parallel \lambda_k^{g(1 \oplus \pi_i, 1 \oplus \pi_j)} \right), \end{aligned}$$

where $\text{Enc}_{W, W'}^g(m)$ denotes symmetric-key encryption of plaintext m using two keys W, W' . In our implementation, we define encryption as

$$\text{Enc}_{W, W'}^g(m) = H(g\|W\|W') \oplus m,$$

where H represents a cryptographic hash function (SHA1) whose output is truncated to the length of the given plaintext. The set of all the garbled gates constitutes the garbled circuit that is sent to the

evaluator. In addition, the circuit generator sends the permutation bit π_i for any output wire i of the circuit.

To evaluate the garbled circuit, the circuit evaluator must obtain keys for each input wire of the circuit; specifically, if the input bit on some input wire i is b_i , then the evaluator should be given the key $W_i^{b_i}$ along with the label $\lambda_i^{b_i}$. (Furthermore, for each input wire it should get *only* that key.) For input wires that correspond to the input of the circuit generator, x , this can easily be arranged by simply having the generator send the appropriate key/label for each such wire. The parties can use *oblivious transfer* [6] to allow the evaluator to obliviously learn the appropriate keys/labels for input wires that correspond to its own input, y .

Given keys/labels W_i, λ_i and W_j, λ_j associated with the input wires i, j of some gate g , the evaluator can compute a key/label for the output wire of that gate by decrypting the ciphertext in position (λ_i, λ_j) of the garbled table for g . Proceeding in this way through the entire (garbled) circuit in topological order, the evaluator can compute keys/labels for each output wire of the circuit. Using the permutation bits sent to him by the circuit generator, this means that the evaluator can determine the actual bit output of the circuit. If specified as part of the protocol, the evaluator can send that result back to the other party.

The crucial point for our purposes is that generation of all the garbled gates can be done in parallel once the wire keys and permutation bits have been chosen. Further details, along with a proof of security, can be found in [17].

Malicious Security Model: Cut-and-Choose.

There are several known ways to modify Yao's protocol to the malicious model, but the only one that has been implemented and deemed practical is the cut-and-choose approach. Here the circuit generator creates not one 'encrypted' circuit, but $\approx k$ 'encrypted' copies of the circuit (where k is a security parameter). The generator sends the k encrypted circuits (without the encryption of its input to the generator). The evaluator now chooses $\approx 60\%$ of the circuits to be revealed, at which point the generators gives all the information necessary to generate the circuit to the evaluator who then verifies that all the circuits are legitimate implementations of the correct circuit for f . If not, the evaluator quits. If so, the evaluator asks for the "encrypted" inputs to the remaining $\approx 40\%$ of the circuits, and computes the output. The evaluator takes as its output the majority output of the many evaluated circuits. The argument for the security of this protocols is beyond the scope of this paper but can be found in many places (cf. [24]).

One Bit Leaked Malicious Model.

In this model, the protocol is modified so that each party plays *both* the role of generator and evaluator. Each party generates a circuit and sends it to the other, which in turn evaluates it. After this is done, a specialized protocol *that is secure in the traditional malicious model* does a secure function evaluation to ensure that the outputs of both evaluated circuits are the same. A specialized and efficient protocol (both in computation and communication) for verifying the equality of outputs in the malicious security model is given by Huang et al. [9].

4. GPU COMPUTING AND CUDA

Modern GPUs are massively parallel computational devices, but differ from modern multi-core CPUs in significant aspects. In this section we provide a brief overview of their architecture. Communication to and from the GPU occurs over the system PCI bus,

which is substantially slower than the regular communication path between RAM and the CPU on a modern machine.

Anatomy of a CUDA GPU.

The smallest execution unit on a CUDA¹ GPU is called a streaming processor (SP), or CUDA core, which is capable of executing an independent thread. These cores are not equivalent to CPU cores but more equivalent to lanes on a vector processor. An SP has access to local memory and registers. Multiple SPs are combined to construct one Streaming Multiprocessor (SM). The number of SPs located in an SM, and complexity of the SM depend upon the GPU hardware generation. Every NVIDIA GPU has multiple SMs.

SMs are in charge of scheduling work to their SPs. SM's receive work in the form of thread blocks. Thread blocks can contain a number of threads defined by the programmer at run time. The SM then splits these thread blocks into groups of 32 threads called warps. Each warp is run on a set of 32 SPs. Each thread in a warp executes one common instruction at a time thus warps are akin to 32-wide vector processors. To ensure every thread is executing the exact same instruction, each thread in a warp must execute the exact same branch in program flow. If different threads need to run different branches, the GPU serializes them by having the appropriate threads execute the branch, while non-branching threads' processors sit idle. When such a divergence in execution occurs between threads in a warp it is termed *warp divergence*. *To get full efficiency from the GPU it is essential that programs be written so that they can be broken down into warps where all threads execute the same instruction*, and there is little to no conditional branching that does not affect all the threads in the same manner. This is the *Single Instruction Multiple Data* (SIMD) paradigm for parallel programming, where the same instruction is applied to multiple pieces of data at a time.

Also note that a given SM might be concurrently executing multiple warps via "hyper-threading". If one warp is waiting on memory access or some other condition, the SM may start to execute another warp. There is little to no cost in time for this context switching, however it does mean that local resources such as registers and local memory need to be shared between these warps.

Relative Speeds of Memory and CPU-GPU bandwidth.

GPUs must deal with latency issues caused by transferring data from the host machine to the GPU and vice-versa. Transfer rates between the host and GPU are $\approx 8\text{GB/s}$ over the PCI-E bus when the GPU is on a PCIe card. The transfer rates are orders of magnitude slower than the GPU's theoretical compute throughput. Memory transfer on-board the GPU is several orders of magnitude faster than the bus (e.g., global memory on a Tesla card transfers at 177.6 GB/s). Therefore, high performance requires minimizing memory transfers and the communication between the GPU and CPU, and maximizing the local computation performed on the GPU. Differing types of memory on the GPU, including global, shared, local (L1, L2 cache), and registers, also operate at varying speeds and thus create a memory hierarchy on the GPU mirroring that on a typical machine. Kernels must optimize the use of local registers and shared memory while dealing with the extremely limited resources of each. Register dependencies, such as when a read directly follows a write to the same register, can also increase latency. Thus we want to maximize register usage to increase speed, however any particular thread also wants to minimize usage so that a SM

¹By CUDA (Compute Unified Device Architecture) we mean an NVIDIA GPU that supports NVIDIA's CUDA programming environment. The most commonly developed for GPU.

can “hyper-thread” multiple warps, if any of them are latent due to memory fetches or other reasons. These conflicting goals make register usage a complicated trade-off in GPU programming.

5. MULTI-CORE CPU VS CLUSTERS

For our CPU based circuit evaluation system, we use local parallelism to take advantage of the multi-core environment found on modern day CPUs. This is in contrast to the different, if not complimentary, approach taken by Kreuter et al. [15], who take advantage of the parallelism available on multi-node compute clusters. We discuss the different technological approaches next.

MPI and OpenMP.

OpenMP and MPI (Message Passing Interface) are both competing and complementary standards for parallelization in High Performance Computing (HPC). OpenMPI is currently a dominant MPI implementation but is not related to OpenMP beyond being a parallelization technology.

MPI is a message passing technology that enables “scale-out” parallelism on multi-device compute clusters, such as super computing clusters. The developer defines an MPI process that is launched many times on many different compute nodes. These MPI processes are able to pass messages between one another when they need to share computation. It works best for large jobs on large systems, as each MPI process incurs large overhead. MPI requires that any machines running MPI code have an MPI implementation’s executables installed, for example, Open MPI’s libraries and executables.² OpenMPI is the technology used by Kreuter et al.[15].

In contrast, OpenMP is an HPC technology designed for “scale-up” parallelism on a single machine. It is a standard that is built in to compilers such as GCC and includes a small driver library. Developers use OpenMP to easily create many lightweight threads with minimal syntax compared to traditional POSIX thread implementations. Unlike POSIX threads, OpenMP is optimized for a data parallel programming paradigm and not a task parallel programming paradigm. Data parallelization is akin to the SIMD (Single Instruction Multiple Data) paradigm and task parallelization is akin to the MIMD (Multiple Instruction Multiple Data) paradigm. OpenMP is the technology we use for our parallel multi-core CPU evaluation scheme.

6. SECURITY MODEL INDUCED ARCHITECTURE TRADE-OFFS

While GPUs gain with massive parallelism, they lose in terms of algorithmic flexibility. Programmers must specify the logical allocation of their threads in terms of thread blocks, and these thread blocks affect physical GPU allocation. If this logical allocation is poor (e.g., setting thread blocks to have one thread) then poor performance follows, as many cores in a SM sit idle while a single core computes the thread. We compare the architectural approach of Frederiksen and Nielsen [5] and Kreuter et al. [15] given their malicious model implementations, and our approach in the HbC and IBM security models. Recall that in the cut-and-choose malicious implementation the generator must generate $\approx k$ circuits while the evaluator will evaluate some 40% of those circuits, and verify the remaining 60% to ensure they were properly constructed. In the HbC case the generator must generate only one circuit and the evaluator must evaluate only one circuit. In the IBM case, each party must generate and evaluate one circuit.

²<http://www.open-mpi.org/software/ompi/v1.6/>

Similarity between HbC and IBM.

Implementation details between HbC and IBM protocols are generally identical as the resources they need are very similar. The IBM protocol differs in only requiring one more circuit generation and one more circuit evaluation than that of the HbC protocol. Therefore, we address both protocols in our discussion as if they were the same model.

Communication Differences.

One immediate observation is that in the malicious model, reasonable values of k might vary between 60 and 120, and thus the number of circuits that need to be transferred between the two agents in the protocol 40% of this,³ compared to the one or two circuits that need to be transmitted our protocols. Frederiksen and Nielsen show that in their protocol with varying security parameters, that communication costs dominate, often by a factor of 3 to 4 times the generation or evaluation times. This is not by enough that one should expect them to dominate in the HbC or IBM scenario we implement. Further, recent advances in the cut-and-choose methodology by Lindell [16] and optimizations that Frederiksen and Nielsen did not implement [24], further reduce the communication burden.⁴ Finally, Frederiksen and Nielsen also increase the circuit-size to include a universal hash of the inputs, and there are alternate approaches that will not increase the circuit size which can be considered. Therefore, we can consider optimizations that disallow the garbled row-reduction methodology, and also slightly increase communication for the sake of efficiency.

Malicious Cut-and-Choose and the GPU.

Cut-and-choose protocols must generate k circuits at a time. Instead of having each thread represent a gate, in cut-and-choose each thread represents one of the k circuits and the thread block is used to represent an individual gate. Thus, each thread block contains k threads and each thread deals with the generation of a specific gate for each of the k circuits. One can then allocate the same number of thread blocks as there are gates in the circuit to the GPU. As each thread block will always have threads processing the same gate type, there is no fear of warp divergence. The only caveat is the thread block size, and thus the cut-and-choose security parameter, should be a multiple of 32 for optimal GPU allocation (again, the SMs allocate 32 threads at a time). Levels are evaluated in turn, but this is less problematic to performance, circuit widths are effectively multiplied by k , meaning the GPU spends little relative time idle waiting for a level to complete before the next is started. Evaluation occurs in a similar manner, but must use the level-by-level process that was used in semi-honest evaluation. For each level there will be a thread block for each gate in that circuit’s level and each thread block will contain k threads. We note this is exactly the approach taken by Frederiksen and Nielsen [5].

Honest-but-Curious and One-Bit Malicious.

The previous description of a successful approach to placing cut-and-choose on the GPU should make it clear why the same approach is inefficient for the semi-honest case (and similarly the IBM case). If only one circuit is being generated or evaluated, each thread block will contain only one thread, and only one thread will be allocated by the SMs on the GPU for each gate leaving 31/32

³It is known that *only* the circuits that are being evaluated need to be transferred as noted in [7], as it suffices to communicate a collision resistant hash of the verified circuits.

⁴In practice Frederiksen and Nielsen evaluate 50% of the circuits.

SPs in a warp dormant (meaning the vast majority of GPU cores are consistently going unused).

We describe our approach to implementing HbC and IBM on the GPU. For simplicity we will assume the circuit we are generating and evaluating fits in GPU memory, although our approach is not limited in this fashion. As we are only concerned with a single circuit, we will pass the whole circuit description to the GPU for generation and evaluation. In the case of generation, each thread represents a single gate in the circuit and the number of threads allocated are the number of gates in the circuit. The size of a thread block does not matter for the HbC or IBM case, although for efficiency it should be a multiple of 32 (the physical thread allocation count by the SM). We can handle XOR gates with one kernel and all other truth table gates with another kernel. The latter essentially always make a blank truth table, and the converts it to the appropriate type by changing each line of the table to describe the appropriate operator. However, to construct a truth table gate one needs to have knowledge of its input wires' labels. This too can seemingly be solved because we can pseudo-randomly generate a wire's labels based on the wire's identifier, but this conflicts with the Free-XOR technique (as described in the next section). We modify the Free-XOR technique at the cost of a small amount of extra communication, and allow all gates in the circuit to be generated in parallel, independent of the level on which they reside.

Evaluation needs to occur on a level-by-level basis to honor data dependencies between gates. Again, we would have two kernels for evaluating the two types of gates, truth table and XOR. Consider what happens when evaluating the end of a level: it is likely that many symmetric processors will sit idle waiting for the level to be completed, as there are no more gates on the current level, say $i - 1$, to evaluate, but they cannot commence processing the level i gates until the last few gates on level $i - 1$ are evaluated due to potential dependency issues. The narrower a level is, the larger the inefficiency if many of the GPU's cores need to lay latent why completing the level. Logic to check for dependencies is likely to cause divergence, and latency problems. Therefore, for circuits which are not wide, the relative amount of time that is spent by the cores being idle at the end of a level can be quite large. In the cut-and-choose approach, the fact that there are k copies of each circuit has the effect of essentially multiplying the width of each circuit by k , making the issue far less problematic. Of course, for particularly large and wide circuits, this should not cause much of an issue for the HbC or IBM implementations.

7. ARCHITECTURE & METHODOLOGY

Several optimizations of Yao's garbled-circuit protocol have been proposed, but it is not clear how all of them can be efficiently implemented in a massively parallel system. Here we discuss the major techniques and our approach to implementing them. As a starting point, we implement the folklore "single row evaluation" technique already described in the description of the Yao protocol in Section 3. This optimization, created by [18], decreases evaluation time on encrypted gates by roughly a factor of 4.

On the other hand, one popular technique for reducing the size of garbled tables by 1/4, called *Garbled-row reduction* [21], is not implemented as any such implementation would seem to slow execution on a SIMD parallelization.⁵ The benefit of this approach is

⁵There are two issues: i) wires in level $i + 1$ of the circuit will now depend on the gates in level i , making parallel generation of the circuit difficult; and ii) during evaluation, about a 1/4 of the cores evaluating encrypted gates would evaluate to the missing row, and require different code than is required for the remaining 3/4 of the cores (causing warp divergence).

that it reduces communication by 25%, but as discussed our security models prompts us to be less concerned about communication costs and more about gate generation and evaluation timings.

7.1 Selection of the Random-Oracle/Permutation Function

The Yao-garbled circuit technique relies on symmetric encryption. In most modern implementations the symmetric encryption is provided via a random oracle instantiated by a cryptographic hash (SHA1). Recent work by Bellare et al. [1] has also considered the use of a Random Permutation that can be instantiated with fixed key in a block-cipher such as AES. In our implementation, we choose the SHA1 function for this purpose. Jang et al. [12] showed that AES had substantially slower throughput than SHA1 on GPU architectures. We have not had the chance to consider an optimized version of AES with a fixed key, as suggested in [1], and this should be investigated. We experimentally tested BLAKE256, a SHA3 finalist, which also had a slower throughput on the GPU than SHA1, when both are given inputs that require the same number of rounds of Merkle-Damgard. Our implementation of SHA1 is a hand optimized version of the John the Ripper implementation of SHA1.⁶ In particular, we are able to reduce the number of rounds we typically need to calculate in SHA1 by judiciously choosing the values we hash. As others have done, we ensure that we never need to hash more than one block of data. However, *by ensuring that the prefix of the values we hash for a given circuit remain relatively constant, we can pre-compute the first few rounds of SHA1 for each query in generating or evaluating a circuit.* This allows us to knock off either 6 or 14 rounds (out of the 80 rounds) of each SHA1 call. We also note that to allow an SM to be able to "hyper-thread", we need to be miserly with our use of registers in this code. Profiling clearly shows that hand optimization of the SHA1 code is a worthwhile endeavor.

7.2 GPUs and Free-XOR

One main challenge in this work is to develop a version of the Free-XOR technique that is compatible with parallelization. We begin by describing the technique.

The *Free-XOR* technique [13] allows XOR gates in the circuit to be evaluated using only a bit-wise XOR operation instead of the standard garbled-gate evaluation. In this approach, the circuit generator chooses a global random offset R , and then ensures that the keys for every wire i in the circuit satisfy $W_i^0 \oplus W_i^1 = R$. This is usually done by choosing keys for each wire i in the circuit that is *not* the output of an XOR gate by sampling W_i^0 at random (as before) but then setting $W_i^1 = W_i^0 \oplus R$. For k an output wire of an XOR gate with input wires i, j , the evaluator sets $W_k^0 = W_i^0 \oplus W_j^0$ and $W_k^1 = W_k^0 \oplus R$. Note that if the circuit evaluator holds input-wire keys W_i, W_j associated with the input wires to an XOR gate, he can compute the corresponding key for the output wire k of that gate as $W_k = W_i \oplus W_j$. Thus, no garbled tables need to be prepared or sent for such gates.

We modify this technique to permit efficient parallel gate generation. In particular, note that when the circuit alternates between non-XOR gates and XOR gates in the 'encrypted' circuit, this creates a dependency amongst wire-labels that would require gates to be generated in leveled order. This is because the output wire labels from the first XOR gates on level $i - 1$ need to be computed for use as the input wires on level i . This creates a dependency regardless of whether or not the gates on level i are XOR or truth tables.

Our modification operates by first virtually generating the labels (Sec. 7.3) for all wires in the circuit, even if the wire is the output

⁶John the Ripper is a brute force password hashing software suite.

wire from a XOR gate. This *differs* from the original Free-XOR technique which *does not randomly generate labels for XOR gate output wires*. Then, after wire label generation, during the generation of XOR gate i , we calculate a label offset (V_i) and a p-bit offset (P_i) that is unique for XOR gate i in the circuit. In the original Free-XOR technique it is at this point where an XOR gate's output wire labels would be generated. Instead, we make use of V_i and P_i to modify our XOR gate to our previously generated wire label and p-bit. The V and P values for every XOR gate can be calculated in parallel on the GPU. Our scheme adds two bitwise XOR operations to XOR gate generation, but this increased overhead is minuscule compared to locating every XOR gate chain and then serially computing each gate in the chain. The calculation of our V and P values during generation are as follows:

1. For each XOR-Gate G_i with wires $W_c = \text{XOR}(W_a, W_b)$ where $W_a = [\langle k_a^0, p_a^0 \rangle, \langle k_a^1, p_a^1 \rangle]$, $W_b = [\langle k_b^0, p_b^0 \rangle, \langle k_b^1, p_b^1 \rangle]$, $W_c = [\langle k_c^0, p_c^0 \rangle, \langle k_c^1, p_c^1 \rangle]$, and tuple $[V_c, P_c]$:
 - (a) Set value $V_c = k_a^0 \oplus k_b^0 \oplus k_c^0$ for wire W_c
 - (b) Set value $P_c = p_a^0 \oplus p_b^0 \oplus p_c^0$ for wire W_c

The use of V and P during evaluation are as follows:

1. For each XOR-Gate G_i with wires $W_c = \text{XOR}(W_a, W_b)$ where $W_a = \langle k_a, p_a \rangle$, $W_b = \langle k_b, p_b \rangle$, and tuple $[V_c, P_c]$:
 - (a) Compute garbled output value $W_c = \langle k_c, p_c \rangle$ which is equal to $\langle k_a \oplus k_b \oplus V_c, p_a \oplus p_b \oplus P_c \rangle$

Communication Costs: As discussed, this modification increases the communication costs by adding an extra 'key' for each XOR gate in exchange for significant parallelized speed improvements. This modification is also incompatible with the Garbled Row Reduction (GRR) optimization [10], which performs more computation in exchange for less communication because GRR also induces a dependence on a circuit's wire labels.

7.3 GPU Wire Creation and Gate Generation

Our system implements garbled circuit generation on the GPU in parallel. We first note a method of virtually generating all label pairs and permutation bits $\langle k_a^0, p_a^0 \rangle, \langle k_b^1, p_b^1 \rangle$ for every wire in the circuit. Because of the memory hierarchy, one does not wish to generate all of the keys associated with labels initially, as the costs of moving and storing these keys in memory is substantial. Instead, we use wire indexes from the circuit description (which are much smaller than cryptographic keys), as inputs to a pseudo-random function generator (PRFG), which outputs the labels for a given wire. Specifically, we output $k_a^0 = F_s(a)$ and $k_a^1 = F_s(a) \oplus R$ for a PRFG F with a circuit specific seed s and the global Free-XOR offset R . The permutation bits are handled similarly. Note that wire-labels are not actually precomputed, but rather only virtually assigned, and computed when constructing the gates that are attached to a given label, for implementation optimization reasons.

7.4 GPU Evaluation

The entire circuit cannot be evaluated in parallel on the GPU. The gates in the circuit must be topologically sorted, and then evaluated. That is, the circuit must be broken into smaller sub circuits such that each subcircuit S has a start gate level G_s and end gate level G_e .

Our API starts GPU evaluation by iteratively transferring several consecutive levels of truth tables and XOR gates to the GPU's memory, and then evaluating them. The next grouping of levels can be asynchronously transferred to the GPU while the previous

grouping of levels is being evaluated. At each level a separate kernel must be used to evaluate XOR and truth table gates.

CPU Evaluation.

Besides GPU Evaluation of garbled circuits we also implemented system to evaluate garbled circuits on the host both serially and in parallel. Our parallel CPU implementation makes use of OpenMP threads (cf. Sec. 5). The CPU parallel and serial evaluation work using the similar process as GPU evaluation, but importantly CPU evaluation does not need to perform any data transfers. The CPU evaluation algorithm iterates over each circuit level in the same manner as GPU evaluation. Serial CPU evaluation will iterate serially over each gate in a level and evaluate it. The parallel CPU evaluation algorithm iterates over gates in a level *in parallel*, where they are divided up equally among all available threads on the machine. Thus if a level contains N gates and the machine has t threads, each thread will process $\frac{N}{t}$ gates. No balancing is done to try and ensure a consistent mix of XOR and truth table gates, as the overhead was deemed higher than the benefit. Given CPUs are MIMD processors there is no need to worry about divergent branches, thus parallel evaluation on the CPU is a more straightforward process.

8. RESULTS

In this section, we present several experiments that support the main claims of this paper and give data on GPU circuit generation, GPU evaluation, and multi-core CPU evaluation. Given that there are now several implementations in different security models, such as HbC, IBM, and malicious security, it is clear a critical metric to the performance of these systems is the number of gates one can generate and evaluate per core per second. In the HbC and IBM security model, gate generation and evaluation are key, as Huang et al.[9] also note. Frederiksen and Nielsen suggest that communication is the fundamental limiting factor in the malicious model, but we recall that there are now several communications improvements that will help to alleviate communication overhead, as discussed in Section 6, which suggest that these metrics should still be of some concern in the malicious model. We do not address communication or its latency here, as it is not in scope of our investigation on the use of different parallelizing technologies to implement efficient and practical circuit generation and evaluation. We discuss this in the Future Work section.

We show through experiment that circuit generation in the HbC and IBM security models can dramatically benefit from a combination of the fine-grained parallelization that has not been exploited in prior works and our modification of the Free-XOR technique. Further, it can easily be accommodated on SIMD-style architectures such as GPUs. This applies to creating individual circuits, and can be carried forward to many duplicates for cut-and-choose scenarios, although the extra communications costs, and the availability of other parallelization techniques in that model may make our approach for that security model less feasible: more experiments need to be performed.

Finally, we show that circuit evaluation is more difficult to parallelize for individual circuits, but can perform better in the cut-and-choose scenario of malicious security.

8.1 Explanation of our Experiments and Data

Producing fair comparisons between different garbled circuit systems is currently challenging. In a perfect world we would execute all systems on the same machine using the same circuit descriptor files, and provide results. Unfortunately, we do not have access to all of the systems and circuit description files necessary to do this. We were able to get the Frederiksen and Nielsen [5] system work-

ing on two of our GPU systems for a direct comparison of our performance to theirs. We do not have interchangeable circuit formats, but we can provide comparisons on a per gate basis. Interestingly, their system performs better on an older architectural generation of GPU card than it was designed to function, so we compare their best performance and ours on both generations of cards. Further, since their implementation is in the malicious model, we cannot simply compare execution times of their many cut-and-choose generations and evaluations with a single generation or evaluation on our system. We took different AES circuits and “copy and pasted” multiple independent copies into one file to simulate the workload needed to generate or evaluate many copies of the circuit in the cut-and-choose protocol, and then compare on a per gate basis.

In the case of Kreuter et al. [15], we have recently been able to support generating and evaluating circuits from their most recent compiler [14], allowing for comparison of generation between the two systems on identical circuits, but we have not yet been able to fully integrate their system with our pipelining code, so we can only compile those circuits for which the entire final circuit fits on the GPU at one time. Larger circuits that need to be broken up and pipelined onto the GPU cannot yet be directly compared. For this reason, the comparisons of our system halt in the experiments when circuit sizes reach the maximal that will fit on the graphics cards. We note that of our two systems, one system’s card has a newer architecture than the other, and so can support slightly larger circuits. Unfortunately, we did not have access to a version of Kreuter et al.’s system that would work on a non-clustered machine, so we could not provide bare-metal side by side comparisons. Therefore, in the case of Kreuter et al. [14], we take the results from their paper and compare them with the same circuits on our machines. All reported results from our experiments are the average of 100 runs. Experiments from Kreuter et al. [14] report average results from 50 runs.

Most prior work in the area benchmarks the time it takes to generate and evaluate various circuits. This process indirectly benchmarks the number of gates generated or evaluated per second. However, this is often run on systems with varying numbers of cores, and to a lesser extent varying speeds. We report results on the average number of gates generated or evaluated per second per core. We note this metric seems relatively stable, and thus we use it for a near apples-to-apples comparison. Table 1 has details for the comparison systems. We note that even though EC2 has multiple GPUs, only one is used in the results presented.⁷ EC2 is run on Amazon’s elastic compute infrastructure, and is running under a Xen hypervisor. Since we do not have direct access to the bare metal, we cannot determine how much overhead the Xen hypervisor entails, but Xen project benchmarks suggest, assuming appropriate kernel patches have been applied, a 0-30% performance decrease [2].

8.2 GPU Circuit Generation

We ran circuit generation on the EC2 and Tie systems (cf. Table 1). We first compare our results to those of Frederiksen and Nielsen [5] in Fig. 1a. We remind the reader that we compare their circuit generation times from experiments where they have similar, but not identical circuits, due to the need to simulate the cut-and-choose malicious protocol, and further, while we did have access to their circuit file, we could not execute it directly as we do not support their file description language in our system, and their binary file format was not conducive to easy translation. Thus, we show in Fig. 1a that under similar workloads our scheme outperforms theirs on the same hardware using the metric of gates generated per

System	CPU	Core/Thrd.	GHz	Ram (GB)	GPU
Kreuter et al. [15]	Xenon E5506	4	2.13	8	N/A
EC2	Xenon X5570	8/16	2.93	24	Tesla S2050
Tie	Xenon E5-2620	12/12	2	64	Tesla K20

GPU	Cores	SMs	GHz	Memory (GB)	Compute Capability
S2050 (EC2)	448	14	1.15	2.7	2.0
K20 (Tie)	2496	13	0.71	4.8	3.5

Table 1: Benchmark system descriptions. EC2 runs a Xen virtual machine.

second. Observe that we generate gates at about 2.3 times the rate on the Tie system compared to Frederiksen and Nielsen on the EC2 system. Observe that we generate gates at about 3 times the rate on the Tie system compared to Frederiksen and Nielsen. This is the benchmark system, as Frederiksen’ and Nielsen’s code is targeted at compute capability 3.X CUDA cards.

As the number of cores on systems can be highly variable, in Fig. 1b we calculate the average rate of gate generation per core for the two systems, to help with understanding performance on other GPU cards with varying numbers of cores. Note that in the benchmarks reported in Figs. 1a and 1b we have commented out any code in our system necessary to split large circuits into smaller sub-circuits so that they can fit onto the GPU, as Frederiksen and Nielsen have no such corresponding code as they simply assume the circuit will fit. Thus we are not penalized for computing overhead that the other system also does not compute.

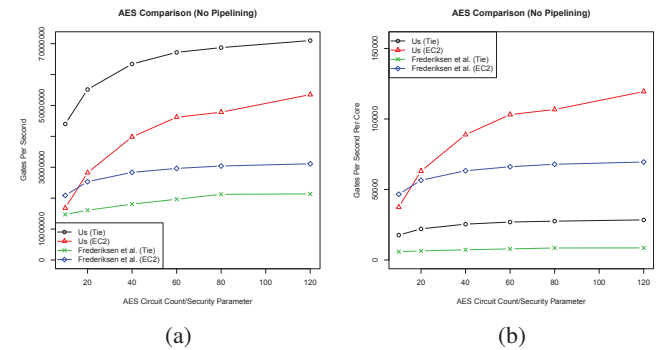


Figure 1: 1a) Circuit gate generation rates of [5] vs. our technique using fully parallelizable circuit generation. 1b) Gates generation rate per multi-processor on differing circuit sizes.

Next, we considered a number of different circuit sizes from both Kreuter et al.[14], and circuits that we have constructed. Given our support of PCF we can compare the same circuits as are tested by Kreuter et al.[14]. We see in Fig. 2, the absolute performance of our system versus that of Kreuter et al. in terms of Gates per sec, and then in Figs. 4a and 4b the relative performance per core. Note that performance per core is relatively stable across medium-to-large circuit sizes. Recall that our cores are substantially more abundant, and have lower cost and energy usage than those of Kreuter et al.

⁷We discuss multiple GPUs in Sec. 9 with respect to future work

Using the metric of gates per second we find our system, in the case of generation, provides significantly higher generation rates: approximately three orders of magnitude. Our system tops out at around 75 million gates per second, while Kreuter et al tops out at 0.35 million gates per second. We note that their system is built for cluster computing, and so they pay a significant overhead to support it.

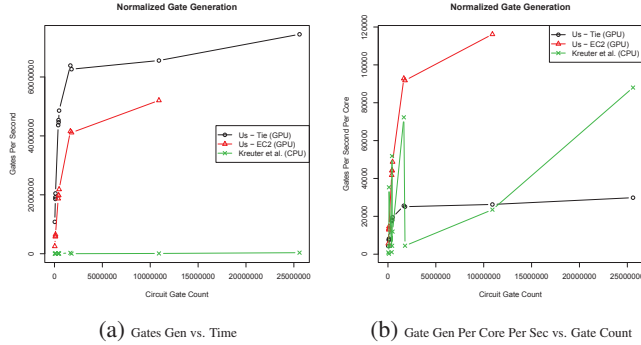


Figure 2: Gate Generation Times comparing to Kreuter et al.[14].

8.3 GPU Evaluation

While on generation we significantly outperform other systems, we only comparable performance to the CPU evaluation techniques of Kreuter et al. [15], and are slightly less efficient on a than the current implementation by Frederiksen and Nielsen [5]. Results are given in Fig. 3.

The advantage that the cut-and-choose protocol entails to parallel evaluation, especially on the SIMD architecture, makes it difficult for an HbC or IBM model protocol to remain competitive. Our evaluation problems seem to stem from two factors: i) It is difficult to keep the GPU fully engaged in processing, due to the limited width of any level of a circuit (recall level i of a circuit must be evaluated before level $i + 1$); and, ii) The lack of memory coalescence in our circuit evaluation data structure seems to impose harsh time penalties on our circuit evaluation times, due to poor memory read/write performance. Memory coalescence occurs on a GPU when all the threads in a warp access adjacent memory locations. Problem ii) is one we believe we can partially improve upon in future work, although we doubt it is possible to achieve the same levels as the cut-and-choose protocol permits (discussed below). Problem i) is inherently more problematic for the HbC and IBM security model protocols, as one can never have guarantees that there are k identical copies of each gate to evaluate, nor do we have the ability to naturally multiply the width of circuits by a factor of $\mathcal{O}(k)$. For naturally large circuits, there may be some hope.

Recall core utilization rates and memory coalescence are less of an issue for Frederiksen and Nielsen: not only are they in fact computing many copies of the AES circuit in the malicious model as we are, but their evaluation algorithm is guaranteed of this fact. This allows them several advantages when constructing kernels to evaluate their circuits. In particular, they can solve the two problems above. First, they can construct a kernel for evaluating each gate in a circuit, and they can evaluate gates from lowest level to the highest. As long as these kernels are scheduled in a leveled order—something easily done—the GPU need never sit with low usage while waiting on kernels to complete a level. Second, since

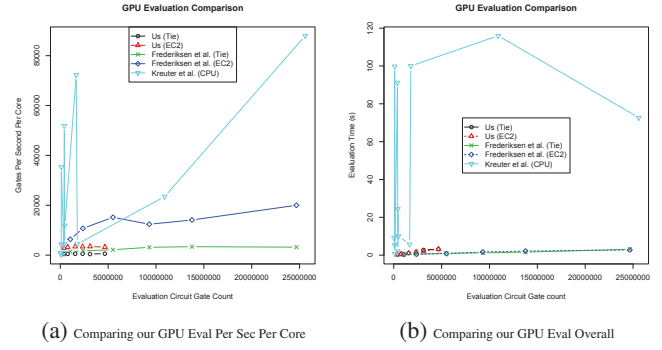


Figure 3: GPU Evaluation Times with comparison to Kreuter et al. [14], Frederiksen and Nielsen [5] and our GPU implementation.

the evaluation is guaranteed that it is executing multiple copies of an identical circuit, it is easier to setup kernels that i) avoid warp divergence, as warps will never process different gate types, and ii)coalesce circuit data in the GPU’s global memory, by simply storing each circuits data adjacent in memory. Note that both of these solutions depend on the GPU taking advantage of multiple identical copies of the same circuit executing.

We see that our GPU marginally outperforms Kreuter et al., suggesting that they are paying a heavy price for using MPI on a single machine (but of course, they are designed to run on large compute clusters, and huge circuits where such performance penalties should be amortized).

8.4 CPU Evaluation

Due in part to the seemingly structural problems of evaluation on a SIMD GPU, we implemented a multi-threaded CPU evaluation scheme in OpenMP. Results can be seen in Fig. 4. It is clear that a MIMD architecture, such as a multi-core CPU will not suffer from warp divergence or memory coalescing problems given their advanced memory controllers and internal logic. A lack of warp divergence removes the fear that large numbers of cores sit idle while a level is completed is less of a problem. Also, we do not need to create multiple distinct ‘kernels’ for different gate types, nor worry that different cores are evaluating different gates. Similarly, the fraction of cores that go unused while waiting for a level to complete, as a total fraction of compute power will be smaller.

While we continue to under perform Frederiksen and Nielsen, we improve over Kreuter et al, and show that their system is likely to benefit from the inclusion of threading within their nodes on the compute-cluster, as opposed to having all of the parallelism at the node level.

9. CONCLUSION, LESSONS LEARNED, AND FUTURE WORK

Given the ability of the GPU to generate large circuits (or large numbers of circuits) efficiently, and the CPUs better performance in evaluation, it seems that an implementation that aims to implement a cut-and-choose protocol, should do verification and generation on the GPU, and evaluation on the CPU in parallel. Similarly, IBM implementations should implement generation on the GPU, and evaluation on the CPU. With appropriate pipelining these would be done in parallel. The technique introduced which allows

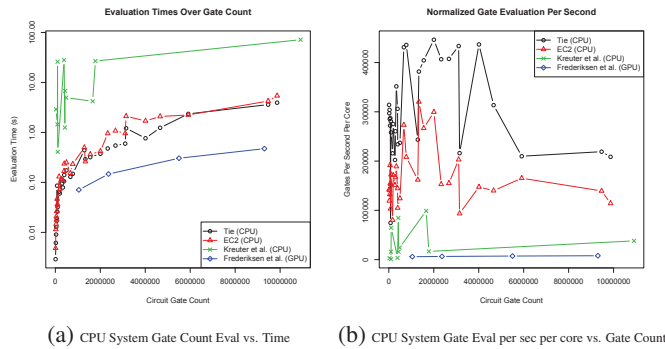


Figure 4: Our Evaluation Times as implemented on the CPU with comparison to Kreuter et al. [14] and Frederiksen and Nielsen [5].

XOR gates to be generated in parallel greatly helps in the circuit gate generation rate.

While we do not report the results here, we have initial work showing there is potential for multiple GPUs to be used in a single system to further speed generation and evaluation results, but a more careful implementation must be done that carefully splits work amongst the GPUs, and takes into consideration the single-bus bottleneck, or card-to-card memory transfer. We plan to pursue these directions as future work.

It is clear that in order for better performance comparisons to be made in the future, there needs to be a test-bank of standard circuits designed. They must be delineated in a standard file format that all future implementations can parse (although, they may further process in this format). Currently, each implementation in the field is rolling its own file format. The recent development of MPC Lounge aims to keep track of such circuits. Similarly, the SCAPi project by Ejgenberg et al. will help in providing a long term supported test environment [4].

10. ACKNOWLEDGEMENTS

The authors would like to thank the NSF and DARPA for funding, Jonathan Katz for discussion and aid on preliminary work, and Tore Frederiksen and Ben Kreuter for aid with their systems.

11. REFERENCES

- [1] BELLARE, M., HOANG, V. T., KEELVEEDHI, S., AND ROGAWAY, P. Efficient garbling from a fixed-key blockcipher. In *IEEE Symposium on Security and Privacy* (2013), IEEE Computer Society, pp. 478–492.
- [2] CAMPBELL, I. Baremetal vs. xen vs. kvm — redux. <http://blog.xen.org/index.php/2011/11/29/baremetal-vs-xen-vs-kvm-redux/> (Nov 2011).
- [3] DAMGÅRD, I., PASTRO, V., SMART, N. P., AND ZAKARIAS, S. Multiparty computation from somewhat homomorphic encryption. In *CRYPTO* (2012), R. Safavi-Naini and R. Canetti, Eds., vol. 7417 of *Lecture Notes in Computer Science*, Springer, pp. 643–662.
- [4] EJGENBERG, Y., FARBSTEN, M., LEVY, M., AND LINDELL, Y. Scapi: The secure computation application programming interface. *IACR Cryptology ePrint Archive 2012* (2012), 629.
- [5] FREDERIKSEN, T. K., AND NIELSEN, J. B. Fast and maliciously secure two-party computation using the gpu. Tech. rep., Cryptology ePrint Archive, Report 2013/046, 2013. <http://eprint.iacr.org>, 2012.
- [6] GOLDBREICH, O. *Foundations of Cryptography*, vol. 2: *Basic Applications*. Cambridge University Press, Cambridge, UK, 2004.
- [7] GOYAL, V., MOHASSEL, P., AND SMITH, A. Efficient two party and multi party computation against covert adversaries. In *EUROCRYPT* (2008), N. P. Smart, Ed., vol. 4965 of *Lecture Notes in Computer Science*, Springer, pp. 289–306.
- [8] HUANG, Y., EVANS, D., KATZ, J., AND MALKA, L. Faster secure two-party computation using garbled circuits. In *USENIX Security Symposium* (2011).
- [9] HUANG, Y., KATZ, J., AND EVANS, D. Quid-pro-quo-tocols: Strengthening semi-honest protocols with dual execution. In *Security and Privacy (SP), 2012 IEEE Symposium on* (2012), IEEE, pp. 272–284.
- [10] HUANG, Y., SHEN, C.-H., EVANS, D., KATZ, J., AND SHELAT, A. Efficient secure computation with garbled circuits. In *Information Systems Security*. Springer, 2011, pp. 28–48.
- [11] ISHAI, Y., KILIAN, J., NISSIM, K., AND PETRANK, E. Extending oblivious transfers efficiently. *Advances in Cryptology-CRYPTO 2003* (2003), 145–161.
- [12] JANG, K., HAN, S., HAN, S., MOON, S., AND PARK, K. Sslshader: cheap ssl acceleration with commodity processors. In *Proceedings of the 8th USENIX conference on Networked systems design and implementation* (2011), USENIX Association, pp. 1–1.
- [13] KOLESNIKOV, V., AND SCHNEIDER, T. Improved garbled circuit: Free XOR gates and applications. pp. 486–498.
- [14] KREUTER, B., MOOD, B., SHELAT, A., AND BUTLER, K. Pcf: A portable circuit format for scalable two-party secure computation. In *To Appear in USENIX Security 2013* (2013).
- [15] KREUTER, B., SHELAT, A., AND SHEN, C.-H. Billion-gate secure computation with malicious adversaries. In *Proceedings of the 21st USENIX conference on Security symposium, Security* (2012), vol. 12, pp. 14–14.
- [16] LINDELL, Y. Fast cut-and-choose based protocols for malicious and covert adversaries. In *CRYPTO (2)* (2013), R. Canetti and J. A. Garay, Eds., vol. 8043 of *Lecture Notes in Computer Science*, Springer, pp. 1–17.
- [17] LINDELL, Y., AND PINKAS, B. A proof of security of Yao’s protocol for two-party computation. 161–188.
- [18] MALKHI, D., NISAN, N., PINKAS, B., AND SELLA, Y. Fairplay—a secure two-party computation system. In *Proceedings of the 13th conference on USENIX Security Symposium-Volume 13* (2004), USENIX Association, pp. 20–20.
- [19] MOHASSEL, P., AND FRANKLIN, M. K. Efficiency tradeoffs for malicious two-party computation. In *Public Key Cryptography* (2006), M. Yung, Y. Dodis, A. Kiayias, and T. Malkin, Eds., vol. 3958 of *Lecture Notes in Computer Science*, Springer, pp. 458–473.
- [20] NAOR, M., AND PINKAS, B. Computationally secure oblivious transfer. *Journal of Cryptology* 18, 1 (2005), 1–35.
- [21] PINKAS, B., SCHNEIDER, T., SMART, N., AND WILLIAMS, S. Secure two-party computation is practical. pp. 250–267.
- [22] PINKAS, B., SCHNEIDER, T., SMART, N. P., AND WILLIAMS, S. C. Secure two-party computation is practical. In *Proceedings of the 15th International Conference on the Theory and Application of Cryptology and Information Security: Advances in Cryptology* (Berlin, Heidelberg, 2009), ASIACRYPT ’09, Springer-Verlag, pp. 250–267.
- [23] PU, S., DUAN, P., AND LIU, J.-C. Fastplay—a parallelization model and implementation of smc on cuda based gpu cluster architecture. Tech. rep., Cryptology ePrint Archive, Report 2011/097, 2011. <http://eprint.iacr.org>, 2011.
- [24] SHELAT, A., AND SHEN, C.-H. Two-output secure computation with malicious adversaries. In *EUROCRYPT* (2011), K. G. Paterson, Ed., vol. 6632 of *Lecture Notes in Computer Science*, Springer, pp. 386–405.
- [25] YAO, A. How to generate and exchange secrets. In *Foundations of Computer Science, 1986., 27th Annual Symposium on* (1986), IEEE, pp. 162–167.

Blackbox Construction of A More Than Non-Malleable CCA1 Encryption Scheme from Plaintext Awareness

Steven Myers
Indiana University
samyers@indiana.edu

Mona Sergi
University of Virginia
ms4bf@virginia.edu

abhi shelat
University of Virginia
abhi@virginia.edu

*

Abstract

We construct a Non-Malleable Chosen Ciphertext Attack (NM-CCA1) encryption scheme from any encryption scheme that is also plaintext aware and weakly simulatable. We believe this is the first construction of a NM-CCA1 scheme that follows strictly from encryption schemes with seemingly weaker or incomparable security definitions to NM-CCA1.

Previously, the statistical Plaintext Awareness #1 (PA1) notion was only known to imply CCA1. Our result is therefore novel because unlike the case of Chosen Plaintext Attack (CPA) and Chosen Ciphertext Attack (CCA2), it is unknown whether a CCA1 scheme can be transformed into an NM-CCA1 scheme. Additionally, we show both the Damgård Elgamal Scheme (DEG) [6] and the Cramer-Shoup Lite Scheme (CS-Lite) [5] are weakly simulatable under the DDH assumption. Since both are known to be statistical Plaintext Aware 1 (PA1) under the Diffie-Hellman Knowledge (DHK) assumption, they instantiate our scheme securely.

Furthermore, in response to a question posed by Matsuda and Matsuura [12], we define cNM-CCA1-security in which an NM-CCA1-adversary is permitted to ask a $c \geq 1$ number of parallel queries after

*This work is Sponsored by the NSF under grant 0939718, and under DARPA and AFRL. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the Defense Advanced Research Projects Agency or the US government. This research is sponsored by the Defense Advanced Research Projects Agency (DARPA) and the Air Force Research Laboratory (AFRL) under contract FA8750-11-C0080.

receiving the challenge ciphertext. We extend our construction to yield a cNM-CCA1 scheme for any constant c . All of our constructions are black-box.

Keywords: Public-Key Encryption, Plaintext-Awareness, Non-Malleability.

1 Introduction

Public-key encryption is one of the most commonly used cryptographic primitives in practice and theory. However, our community's understanding of its different security definitions and their relationships is still poor. Goldwasser and Micali [11] formalized the notion of computational security against passive eavesdroppers through the concept of semantic security or chosen plaintext (CPA) security. However, security against passive eavesdroppers is too weak to be used in modern applications, and thus stronger notions of security have been proposed and studied.

Naor and Yung [15] introduced the first strengthening of security by considering adversaries who have the ability to decrypt messages of their choice. In their notion, called Chosen Ciphertext Attacks #1 (CCA1), the adversary is not allowed to decrypt ciphertexts related to the ciphertext of interest. Later, a more comprehensive notion of adversarial decryption introduced by Simon and Rackoff [17] and termed CCA2 security became the gold standard requirement for decryption on the Internet. While it is clear that stronger security notions imply weaker ones, and thus CCA2-secure schemes imply CCA1 secure ones which in turn imply CPA secure ones, the converse directions are not known to be true. While it is the case that a CPA (resp. CCA1) secure encryption scheme need not be CCA1 (resp. CCA2) secure, it is not known if the existence of a CPA (resp. CCA1) secure scheme *implies* the existence of a CCA1 (resp. CCA2) scheme. These are considered some of the major open questions in cryptography.

The CPA and CCA1 security notions for encryption suffer another weakness which must also be addressed for public key encryption to function in modern settings. In particular, the CPA and CCA1 security definitions do not prevent an adversary who observes an encryption of the message m from producing an encryption of the message $f(m)$ for some function f (even though the value m remains private). The seminal work of Dolev, Dwork, and Naor [10] addressed this security issue by introducing the notion of non-malleable cryptographic primitives such as encryption schemes, commitment schemes, and zero-knowledge. Later, Pass, Shelat and Vaikuntanathan [16] strengthened the DDN definition

and presented a construction from CPA to non-malleable CPA encryption using non-blackbox use of the CPA encryption scheme. There have been many follow-up works that propose more efficient constructions of non-malleable primitives. A notable achievement in this line of research has been the construction of non-malleable CPA encryption from standard versions of encryption in a black-box manner [3].

In general, any progress on constructing public-key encryption schemes with stronger security properties from weaker ones is of great interest in furthering our understanding of public key encryption. Beyond theoretical importance, it is of practical value: when new cryptographic assumptions are shown to be sufficient for public-key encryption, it would be valuable to know that they are simultaneously sufficient for the strong forms we need for use in modern settings.

With this in mind, we consider the open question of whether an NM-CCA1 encryption scheme can be constructed from a CCA1 encryption scheme. We present a black-box construction of an NM-CCA1 encryption scheme from a subset of CCA1 encryption schemes which are also plaintext aware under multiple keys and weakly simulatable (we will formally define these concepts later). Intuitively, an encryption scheme is plaintext aware (called **sPA1** in [1]) if the only way that a ppt adversary can produce a valid ciphertext is to apply the (randomized) encryption algorithm to the public-key and a message. Notice that this definition does not imply non-malleability since there is no constraint on what an adversary can do *when given a valid ciphertext*. In fact, both plaintext-aware encryption schemes constructed in [1] are multiplicatively homomorphic, and thus clearly malleable. The weakly simulatable property in our construction is required for technical reasons and roughly corresponds to the ability to sample ciphertexts and pseudo-ciphertexts without knowing any underlying plaintext (if such a plaintext exists).

Note that there exist encryption schemes that satisfy security notions that “sit between” standard notions. One such example from Cramer et al. [4] consists of a black-box construction of a q -bounded CCA2 encryption scheme which is not NM-CPA-secure¹, but which satisfies a stronger security notion than CPA. In particular, as a generalization of NM-CPA, Matsuda and Matsuura [12] put forth the challenge of constructing encryption schemes that can handle more than one parallel query after revealing the challenge ciphertext. They write:

¹The [4] construction supports only q queries, whereas an NM-CPA adversary can submit more than q ciphertexts in its final parallel query.

“Since any (unbounded) CCA secure PKE construction from IND-CPA secure ones must first be secure against adversaries who make two or more parallel decryption queries, we believe that overcoming this barrier of *two parallel queries* is worth tackling.”

In this spirit, we define an extension over NM-CCA1, cNM-CCA1, in which the adversary can make c adaptive parallel decryption queries after seeing the challenge ciphertext, where each parallel decryption query can request that a polynomial number of ciphertexts (excluding the challenge ciphertext) be decrypted. Thus that NM-CCA1 is cNM-CCA1 where the parameter c is set to be one. Next, we show how to construct a cNM-CCA1 secure encryption scheme for an arbitrary constant c . Unfortunately, the size of the ciphertext in our cNM-CCA1 encryption scheme is multiplicatively polynomially bigger than the size of the ciphertext in a $(c - 1)$ NM-CCA1 encryption scheme and thus c must be a constant to obtain an efficient construction.

While our initial goal was to construct an NM-CCA1 scheme from a subset of the CCA1-secure schemes, a result by Bellare and Palacio [1] shows that any plain-text aware scheme that is CPA secure is also CCA1-secure, and thus formally all of our results follow from any CPA-secure that also has the necessary plaintext aware and simulatability properties. However, we show that the weak simulatability requirement implies CPA security, and therefore all of our results follow from any scheme which is weakly simulatable and plaintext aware.

About Knowledge Extraction Assumptions Our constructions rely on encryption schemes that are plaintext aware ($\mathbf{sPA}_{1\ell}$) in the multi-key setup and are weakly simulatable. In Theorem 5, we show that such encryption schemes exist under a suitable extension of the Diffie-Hellman Knowledge (DHK) assumption that was originally proposed by Damgård, and modified to permit interactive extractors by Bellare and Palacio [1]. Dent [9] has since shown that it is secure in the generic group model. Some critics of the DHK assumption have commented on its strength and observed that it is not efficiently falsifiable [14]. However, it is not our goal to argue whether or not it is an assumption which should be used in deployable systems. Instead we note it is seemingly a weaker assumption than the Random Oracle model, which is known to be incorrect in full generality, cf. [2] and is yet pervasively used in theory and practice. In contradistinction, we are not aware of any general security definitions that are non-

trivially weaker or incomparable to NM-CCA1 yet imply schemes which are NM-CCA1. Similarly, the gap between NM-CCA1 and CCA2 is poorly understood.

Techniques Both our NM-CCA1 and cNM-CCA1 constructions are based on the ideas of the nested encryption construction by Myers and Shelat in [13]. We first encrypt the message under one key (we refer to this ciphertext as the *inner layer*), and encrypt the resulting inner layer ciphertext repetitively under an additional k keys, where k is the security parameter (we refer to these k keys as the “outer keys”, and the ciphertexts they produce as the “outer layer”). During decryption, all the outer layer ciphertexts are decrypted, and it is verified that they all encode the same inner layer value. This idea is combined with the well-studied notion of non-duplicatable set selection (in this case of public-keys used to encrypt the outer-layer encryptions), such that anyone attempting to maul a ciphertext has to perform their own independent outer layer encryption. Intuitively, anyone that can encrypt to a consistent outer layer encryption under a new key must have knowledge of the underlying inner-layer.

On a more technical level, there are several challenges that need to be overcome. The technical difficulty in proving weaker public-key encryption security notions imply stronger security notions lies in the simulation of a decryption oracle. When beginning with a $\mathbf{sPA1}_\ell$ -secure encryption primitive, we can easily simulate the first phase decryption oracle in the NM-CCA1 security definition by using the plaintext extractor guaranteed by the $\mathbf{sPA1}_\ell$ security definition. However, we cannot simply use the extractor to simulate the decryption oracle *after* the adversary receives the challenge ciphertext in the NM-CCA1 security experiment. This is because the plaintext-aware security definition does not guarantee that an extractor works if the underlying randomness used to create the ciphertext by the challenger is not known to the challenger. Generally, an adversary that mauls an input ciphertext may not have access to this underlying randomness. To overcome this problem, we make use of the notion of weak simulatability.

Contributions To summarize, our contribution is twofold. Our work shows the first black-box construction of a non-malleable CCA1 encryption scheme in the standard model from a weaker encryption primitive. Secondly, for the first time, we show how to construct an encryption scheme that is not CCA2 secure but is secure against an adversary that can ask a

bounded number of polynomial-parallel queries after receiving the challenge ciphertext, satisfying a natural extension to the notion of NM-CCA1 security. This might be of independent interest since the development of constructions that satisfy stronger notions than non-malleable CCA1 security but do not satisfy CCA2 security can provide insight into the technical difficulties with understanding the relationship between CCA1 and CCA2. For example, prior to this work the authors did not believe it was clear that such schemes existed. At least one of the authors felt it was plausible that being able to provide multiple parallel queries after access to a challenge ciphertext was equivalent to providing an arbitrary polynomial number of parallel queries.

Finally, we note that none of our constructions

2 Notations and Definitions

We use $[n]$ to denote the set $\{1, 2, \dots, n\}$. We say a function $\mu : \mathbb{N} \rightarrow \mathbb{R}$ is negligible if for all polynomials p and all sufficiently large $n : \mu(n) \leq 1/p(n)$. Given two families of distributions $D_0 = \{D_{0,i}\}_{i \in \mathbb{N}}$ and $D_1 = \{D_{1,i}\}_{i \in \mathbb{N}}$, we denote that they are computationally indistinguishable by writing $D_0 \approx_c D_1$.

We use the standard definition for CPA/CCA1/CCA2 security, and a definition of non-malleability for CCA1 encryption schemes based on the non-malleability definition for CPA encryption schemes in [16]. In the NM-CCA1 game, the adversary is allowed to ask an unbounded number of decryption queries before seeing the challenge ciphertext, and one parallel query afterwards. A parallel decryption query is one that consists of unbounded number of ciphertexts, none of which will be decrypted until all the ciphertexts in the query are submitted.

To generalize NM-CCA1 security, we can define cNM-CCA1 security identically to NM-CCA1 except that the adversary can make $c \geq 1$ parallel queries after seeing the challenge ciphertext. For example, in the CCA2 security definition, the adversary may ask an unbounded number of queries before and after seeing the challenge ciphertext; thus, cNM-CCA1 is an intermediate notion that we study to understand public key encryption.

Definition 1 (cNM-CCA1). *For an integer $c \geq 0$, we say that a scheme $\Pi = (\text{nmg}, \text{nme}, \text{nmd})$ is cNM-CCA1 or (c)NME secure if for all ppt adversaries and distinguishers $\mathcal{A} = (\mathcal{A}_0, \dots, \mathcal{A}_c)$ and $\mathcal{D}$ respectively and for all polynomials p , we*

have that

$$\left\{ (c)\text{NME}_0(\Pi^{(c)}, \mathcal{A}, \mathcal{D}, k, p(k)) \right\}_k \approx_c \left\{ (c)\text{NME}_1(\Pi^{(c)}, \mathcal{A}, \mathcal{D}, k, p(k)) \right\}_k$$

where experiment $(c)\text{NME}$ is defined in Fig. 1.

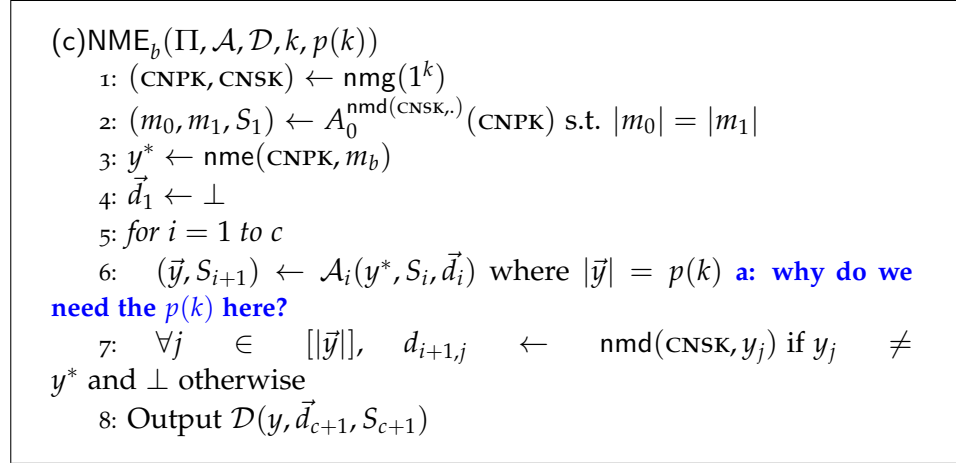


Fig. 1: THE $(c)\text{NME}$ EXPERIMENT FOR $c \geq 0$. An Adversary $\mathcal{A}$ gets c parallel queries to a decryption oracle.

2.1 Weakly Simulatable Encryption Scheme

Dent [8] introduced the notion of simulatability for an encryption scheme. Intuitively, an encryption scheme is simulatable if no attacker can distinguish valid ciphertexts from some family of pseudo-ciphertexts (which will include both valid encryptions and invalid encryptions). This family of pseudo-ciphertexts must be efficiently and publicly sampleable and somewhat invertible (given any pseudo-ciphertext, one can find a random looking string that generates it). In Dent's definition, a distinguisher is given a challenge "ciphertext" (i.e., either a legitimate ciphertext or a pseudo-ciphertext) and must classify it. The distinguisher has access to a decryption oracle to help it distinguish between pseudo-ciphertexts and legitimate ones, but it cannot query the oracle on the challenges that it is trying to distinguish. We introduce a weak notion of simulatability where the attacker is not given access to the decryption oracle.

Definition 2. (Weakly Simulatable Encryption Scheme) An asymmetric encryption scheme $\Pi = (\text{gen}, \text{enc}, \text{dec})$ is weakly simulatable if there exist two

poly-time algorithms (f, f^{-1}) , where f is deterministic and f^{-1} is probabilistic, such that for all $k \in \mathbb{N}$ there exists the polynomial function p where $l = p(k)$, and the following correctness properties hold for every pk in the range of gen :

1. For each $r \in \{0, 1\}^l$, assign $c \leftarrow f(pk, r)$ where $c \in \mathcal{C}$. The set $\mathcal{C}$ is the set of all “possible-ciphertext” strings that can be submitted to the decryption oracle (notice that members of $\mathcal{C}$ are both valid and invalid ciphertexts).
2. For each $c \in \mathcal{C}$, $f^{-1}(pk, c) \in \{0, 1\}^l$.
3. For each $c \in \mathcal{C}$, $f(pk, f^{-1}(pk, c)) = c$.
4. For every efficient distinguisher $\mathcal{A}$ and all k , $|\Pr[\text{DIST}_{\Pi}((f, f^{-1}), k, \mathcal{A}) = 1] - 1/2| \leq \mu(k)$, where μ is some negligible function and the DIST experiment is defined as follows:

$\text{DIST}_{\Pi}(k, (f, f^{-1}), \mathcal{A})$
 1: $(pk, sk) \leftarrow \text{gen}(1^k)$
 2: $(m, \sigma) \leftarrow \mathcal{A}(pk)$, where σ is state.
 3: $b \leftarrow \{0, 1\}$, $r \leftarrow \{0, 1\}^l$, $c \leftarrow \text{enc}_{pk}(m)$
 4: if $b = 0$, then $p = (r, f(pk, r))$
 5: else $p = (f^{-1}(pk, c), c)$.
 6: $b' \leftarrow \mathcal{A}(\sigma, p)$.
 7: Output 1 if $b = b'$.

When valid ciphertexts cannot be distinguished from pseudo-ciphertexts that need not encode messages, CPA security is immediate. The converse need not hold because ciphertexts might be hard to generate and invalid ciphertexts might be easily distinguishable from illegitimate ones (for example, they might contain a zero-knowledge proof of validity). Notice that the weak simulatability notion is not equivalent to the Invertible Sampling notion introduced in [7] since in this definition the plaintext is not needed to compute the pseudo-random string that generates the ciphertext.

Theorem 1. *If E is a weakly simulatable encryption scheme, then E is CPA secure.*

Proof. Let E be weakly simulatable using the efficiently computable functions (f, f^{-1}) . Let $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ be a CPA adversary, that breaks the CPA security of E with advantage ϵ . We will show that if E is a weakly simulatable encryption scheme, then ϵ should be negligible.

In Fig. 2 we present a distinguisher $\mathcal{B}_{\mathcal{A}} = (\mathcal{B}_{\mathcal{A},1}, \mathcal{B}_{\mathcal{A},2})$ that employs $\mathcal{A}$ internally and tries to break E ’s weak simulatability property. We analyze

$$\mathcal{B}_{\mathcal{A},1}(pk)$$

- 1: $(m_0, m_1, \sigma) \leftarrow \mathcal{A}_1(pk)$
- 2: $d \leftarrow \{0, 1\}$
- 3: $Output(m_d, \sigma' = (\sigma, d))$

$$\mathcal{B}_{\mathcal{A},2}(\sigma' = (\sigma, d), (r, c))$$

- 1: $d' \leftarrow \mathcal{A}_2(\sigma, c)$
- 2: If $d = d'$ output 0, otherwise output 1

Fig. 2: THE DISTINGUISHER $\mathcal{B}_{\mathcal{A}}$ USED IN THE DIST EXPERIMENT TO BREAK THE WEAK SIMULATABLE SECURITY OF $\mathbf{E}$.

the advantage of $\mathcal{B}_{\mathcal{A}}$ assuming $\mathcal{A}$ has advantage ϵ in breaking the CPA security of $\mathbf{E}$.

Assuming that $\mathcal{A}$ has advance ϵ in breaking the CPA security of $\mathbf{E}$ implies that:

$$\Pr_{\text{DIST}_{\mathbf{E}}(k, (f, f^{-1}), \mathcal{B}_{\mathcal{A}})}[d' = d \mid b = 0] > 1/2 + \epsilon(k) \quad (1)$$

Notice that $\Pr[d' = d \mid b = 0]$ is the probability that $\mathcal{A}$ guesses the encrypted message correctly when it is given a valid ciphertext. Also $\mathbf{E}$ is weakly simulatable if and only if for some negligible function ϵ'

$$|\Pr[b' = 0 \mid b = 0] - \Pr[b' = 0 \mid b = 1]| < \epsilon'(k) \quad (2)$$

Note that by definition in $\text{DIST}_{\mathbf{E}}(k, (f, f^{-1}), \mathcal{B}_{\mathcal{A}})$, $b' = 0$ if and only if $d' = d$. Hence $\Pr[b' = 0 \mid b = 0] = \Pr[d = d' \mid b = 0]$ and $\Pr[b' = 0 \mid b = 1] = \Pr[d = d' \mid b = 1]$. We have that

$$\Pr[d = d' \mid b = 1] = 1/2 \quad (3)$$

because whenever $b = 1$, the ciphertext c to the distinguisher $\mathcal{B}_{\mathcal{A},2}$ is independent of the bit d , and the distinguisher's probability guessing the random bit d is exactly $1/2$. We have that:

$$\begin{aligned} |\Pr[b' = 0 \mid b = 0] - \Pr[b' = 0 \mid b = 1]| &= |\Pr[d = d' \mid b = 0] - \Pr[d = d' \mid b = 1]| \\ &> 1/2 + \epsilon - 1/2 = \epsilon \end{aligned} \quad (4)$$

where the inequality follows from substituting the (in)equalities 1 & 3 respectively. We combine the inequalities 2 and 4:

$$\epsilon < |\Pr[b' = 0 \mid b = 0] - \Pr[b' = 0 \mid b = 1]| < \epsilon'(k)$$

$$\rightarrow \epsilon < \epsilon'(k)$$

Since ϵ is at most a negligible value in the security parameter, we conclude that $\mathcal{A}$ has at most negligible advantage in breaking the CPA security of E . Hence E is CPA secure. $\square$

Instantiating weak-simulatability Following the ideas of Dent [8], we show in Appendix A how the Damgård ElGamal (DEG) and CS-lite encryption schemes—summarized in Fig. 8 for convenience—can both be weakly simulatable when instantiated in the proper groups.

2.2 Plaintext Awareness For Multiple Key Setup

In Fig. 3 we present a slight generalization to the definition of $\mathbf{sPA1}$ by [1] in which multiple keys are given to the ciphertext creator and the extractor must be able to decrypt relative to any one of them.

$\mathbf{sPA1}_\ell(\Pi = (\text{gen}, \text{enc}, \text{dec}), \text{Crt}, \text{Ext}, k)$

- 1: Let $R[\text{Crt}], R[\text{Ext}]$ be randomly chosen bit strings for Crt and Ext
- 2: $((pk_i, sk_i))_{i \in [\ell(k)]} \leftarrow \text{gen}(1^k)$
- 3: $st \leftarrow ((pk_i)_{i \in [\ell(k)]}, R[\text{Crt}])$
- 4: $\text{Crt}^{\text{Ext}(st, \cdot)}((pk_i)_{i \in [\ell(k)]})$
- 5: Let $Q = \{(q_i = (pk_{j_i}, c_i), m_i)\}$ be the set of queries and responses made to Ext.
- 6: Return $\bigwedge_{i=1}^{|Q|} (m_i = \text{dec}_{sk_{j_i}}(c_i))$ (Note $a = b$ is a boolean)

Fig. 3: THE $\mathbf{sPA1}_\ell$ DEFINITION

Definition 3 ($\mathbf{sPA1}_\ell$ -Security). A public-key encryption scheme $\Pi = (\text{gen}, \text{enc}, \text{dec})$ is said to be $\mathbf{sPA1}_\ell$ secure, for a polynomial ℓ , if for each ppt ciphertext creator Crt, there exists a ppt extractor Ext and negligible function μ s.t. for all $k \in \mathbb{N}$: $\Pr[\mathbf{sPA1}_\ell(\Pi, \text{Crt}, \text{Ext}, k) = 0] \leq \mu(k)$, in which case the Ext is deemed successful for Crt. We define $\text{Adv}^{\mathbf{sPA1}_\ell}(E, \text{Crt}, \text{Ext}, k)$ to be $\Pr[\mathbf{sPA1}_\ell(E, \text{Crt}, \text{Ext}, k) = 0]$.

Notice that the $\mathbf{sPA1}$ definition is a special case of $\mathbf{sPA1}_\ell$ where $\ell(k) = 1$. In Fig. 3, Crt is a ciphertext creator. Ext is a stateful ppt algorithm called the *extractor* that takes as input its state information st and a ciphertext given by the ciphertext creator Crt . Ext will return the decryption of that ciphertext and an updated state st . Ext 's initial state is set to the public-keys pk_i and Crt 's random coins $R[\text{Crt}]$. The state gets updated by Ext as it answers each decryption query that Crt submits.

In Appendix B, we argue that Cramer-Shoup Lite (CS-Lite) and Damgård's ElGamal (DEG), described in Fig. 8, are $\mathbf{sPA1}_\ell$ secure based on a suitable modification of the Diffie-Hellman Knowledge definition that was originally proposed by Damgård, and then later modified to permit interactive extractors by Bellare and Palacio [1].

2.3 Why $\mathbf{sPA1}_\ell$ does not follow from $\mathbf{sPA1}$ security

It may seem that the $\mathbf{sPA1}_\ell$ definition should follow naturally from $\mathbf{sPA1}$ by composing extractors. The following toy example highlights the technical difficulty with natural composition. Let (g, e, d) be an $\mathbf{sPA1}$ -secure primitive, and define a new encryption scheme (g', e', d') which creates two pairs of keys from the original encryption scheme, and chooses one (at random) to use to encrypt during the encryption process. Formally, $g'(k) = (PK = (pk_0, pk_1), SK = (sk_0, sk_1))$, where (pk_b, sk_b) are an output of the b th invocation of $g(k)$. For encryption, $e'(PK, m)$ chooses a random coin $z \in \{0, 1\}$ and outputs $C = (z, e(pk_z, m))$; decryption is $d'(SK, C = (z, c))$ outputs $d(sk_z, c)$. One would expect that the resulting scheme is $\mathbf{sPA1}$ secure, but it is not clear that it is. In particular, one would think that for any ciphertext creator for the modified scheme, one could just use two extractors for the original scheme (one for each public-key) to simulate an extractor for the creator. However, this argument does not work, and we are not aware of any other methods for proving the equivalence. One issue is that if a creator switches between making encryptions under pk_b and pk_{1-b} , then at each switch we must incorporate the extractor in to the original ciphertext creator in order to construct a new extractor. The extractors must be continuously incorporated, because definitionally they have no ability to extract encryptions when the ciphertext creator has access to a decryption oracle other than the one simulated by the extractor.

More specifically, consider a ciphertext creator $\text{Crt} = (\text{Crt}_0, \text{Crt}_1, \dots, \text{Crt}_n)$ for the scheme (g', e', d') where Crt_i denotes the execution after the i th

query.² Let Crt switch between the public-key used to encrypt messages for each query, i.e. it encrypts its even queries with pk_0 and its odd queries with pk_1 . To make an extractor for Crt (without including the oracles in the definition, as we have done), we would first create Crt'_0 using the standard **sPA1** definition and the extractor for pk_0 that is guaranteed to exist for Crt_0 , call it Ext_0 , by embedding the extractor as a subroutine into Crt_0 . The running time of Crt'_0 is clearly the additive combination of the running time of Ext_0 and Crt_0 . One would then compose Crt'_0 with Crt_1 and use the **sPA1** definition to construct an extractor for $\text{Crt}_1 \circ \text{Crt}'_0$, called Ext_1 , which only queries decryptions for pk_1 . We could continue this inductively, but after a super-constant number of iterations, the running time of the resulting extractor would be super-polynomial.

Finally, we note that common additional definitional traits, like the notion of a history of computation, do not port readily to these extractability definitions. In essence, one needs to consider the possibility that a history string encodes a Turing Machine, which is then run by an extractor acting as a Universal Turing machine. The semantic effect of such a notion in the definition is to swap the order of quantifiers relating to the extractor, further strengthening the definition.

2.4 A Note On PA1^+

Dent [8] also investigated an augmented notion of plaintext awareness called PA1^+ in which he provides the ciphertext creator access to an oracle that produces random bits. The extractor receives the answers to any queries generated by the creator, but only at the time these queries are issued. The point of this oracle in the context of a plaintext awareness definition is to model the fact that the extractor might not receive all of the random coins used by the creator *at the beginning* of the experiment. Much in the spirit of “adaptive soundness” and “adaptive zero-knowledge”, this oracle requires the extractor to work even when it receives the random coins at the same time as the ciphertext creator. Therefore, the extractor potentially needs to be able to extract some ciphertexts independent of future randomness. This modification has implications when the notion of plaintext awareness is computational—as in the case of Dent’s work. However, in the case of statistical plaintext awareness, we argue that sPA1_ℓ security also implies sPA1_ℓ^+ security.

²We can assume the Crt_i outputs its state, which is then used as auxiliary information and passed as input to Crt_{i+1} .

Definition 4. Define the $\mathbf{sPA1}_\ell^+$ experiment in a similar way to the $\mathbf{sPA1}_\ell$ experiment. The only difference between the two is that during the $\mathbf{sPA1}_\ell^+$ experiment, the ciphertext creator has access to a random oracle $\mathcal{O}$ that takes no input, but returns independent uniform random strings upon each access. Any time the creator access the oracle, the oracle's response is forwarded to both the creator and extractor.

If an encryption scheme would be deemed $\mathbf{sPA1}_\ell$ secure, when we replace the $\mathbf{sPA1}_\ell$ experiment in the definition with the modified $\mathbf{sPA1}_\ell^+$ experiment, then the encryption scheme is said to be $\mathbf{sPA1}_\ell^+$ secure.

Lemma 1. If an encryption scheme Π is $\mathbf{sPA1}_\ell$ secure, then it is $\mathbf{sPA1}_\ell^+$ secure.

Proof. Notice that the only difference between $\mathbf{sPA1}$ and $\mathbf{sPA1}^+$ security definitions is that the latter makes use of an oracle $\mathcal{O}$ that returns random bits upon access that is not known in advance to either the adversary or the extractor. If the adversary Crt^+ does not access $\mathcal{O}$ during its execution then $\mathbf{sPA1}^+$ security holds since i) with no access to $\mathcal{O}$, $\mathbf{sPA1}^+$ and $\mathbf{sPA1}$ security are equivalent, ii) $\mathbf{sPA1}$ security holds (i.e. for any given adversary, there exists an extractor that can decrypt the queries correctly). Hence in what follows we only argue that $\mathbf{sPA1}^+$ security holds if the adversary accesses the oracle $\mathcal{O}$ at least once.

Let Crt^+ be an $\mathbf{sPA1}^+$ adversary. The intuition for this argument is that the answers that the Crt^+ receives from $\mathcal{O}$ can be interpreted as “the end of the random tape” for some $\mathbf{sPA1}$ adversary Crt . In other words, Crt runs Crt^+ internally and answers the queries to $\mathcal{O}$ by reading the end portion of its random tape. By properly formalizing this to handle polynomially many queries to $\mathcal{O}$, it is easy to see that Crt will make the same distribution of queries to its extractor that Crt^+ makes to its own extractor Ext^+ . By $\mathbf{sPA1}$ -security, there exists an Ext that works for the queries that Crt produces.

This observation provides a plausible model for how Ext^+ could work: it begins by sampling a random tape $R^+ \leftarrow (R|r'_1| \cdots |r'_n)$ for Crt (i.e., with randomly sampled answers r_i to $\mathcal{O}$ queries at the end of the tape). When it is asked queries to decrypt, it simply runs Ext (which must work properly) using this random tape. At some point, the first oracle query to $\mathcal{O}$ will be made and thus Ext^+ will receive a random string r_1 as the first oracle $\mathcal{O}$ query answer. At this point, Ext^+ updates the tape $R^+ \leftarrow (R|r_1|r'_2| \cdots |r'_n)$ with the correct answer, restarts its execution of Ext on this new tape by feeding it all of the same decryption queries that have been received up until this point. This results in a new state for the extractor that will

be used to answer future decryption queries. The remaining decryption queries and queries to $\mathcal{O}$ will be handled similarly.

One can observe that if ℓ queries to $\mathcal{O}$ are made, then Ext^+ must restart the execution of Ext ℓ times, and thus its running time will be a summation of ℓ running times of Ext . This summation will still be polynomial in the security parameter. Moreover, if Ext^+ fails to answer a decryption query properly, then it serves as a polynomial-time procedure that—by running Ext at most ℓ times—is able to produce a set of queries that breaks **sPA1**-security. In what follows, we present a more formal argument that shows how an **sPA1**⁺-adversary Crt^+ that succeeds in causing every Ext^+ to fail can be used, using the ideas above, to produce an **sPA1**-adversary Crt that violates **sPA1**-security.

Assume that the Crt^+ makes polynomially many queries using its random tape R , then accesses $\mathcal{O}$ once (and gets in return some random coins r_1) and asks one more query q that Ext^+ fails to decrypt correctly. Assume that $(R|r_1)$ is ℓ bits. We build an adversary Crt that simulates Crt^+ using the first ℓ bits of its random tape. Crt reads the first ℓ bits of its random tape, parses it as $(R|r_1)$ (call the rest of its random tape $r'_2|r'_3|\dots|r'_n$) and simulates Crt^+ on random coins R . Crt submits Crt^+ 's queries to its extractor and forwards back the answer to Crt^+ . When Crt^+ calls $\mathcal{O}$, Crt returns the r_1 portion of its tape. Then it continues running Crt^+ until it gets its next decryption query q , submits this query to its extractor, and halts. Notice the distribution of queries that Crt asks to its extractor is the same as Crt^+ . Also, Crt does a perfect simulation of the **sPA1**⁺ game for Crt^+ , so the query q is also distributed identically. Since Π is **sPA1** secure, there must be some extractor Ext such that $\Pr[\mathbf{sPA1}(\Pi, \text{Crt}, \text{Ext}, k) = 0]$ is negligible when Ext is run on input tape $(R|r_1|r'_2|r'_3|\dots|r'_n)$ as the random coins for Crt . Thus, the probability that Ext answers the query q correctly must be $1 - \epsilon(k)$ for some negligible function ϵ . However, notice that Ext^+ also answers q by running Ext on $(R|r_1|r'_2|r'_3|\dots|r'_n)$ and hence returns the same correct decryption of q to Crt^+ . This contradicts our assumption that Ext^+ decrypt q incorrectly. Similar argument can be made about any other query that Crt^+ makes to show that Ext^+ returns the right decryption for that query. Hence we conclude that Ext^+ always returns the right answer to all of the Crt^+ 's queries. $\square$

3 More Than Non-Malleable CCA1 Encryption Scheme

We show how to construct an encryption scheme that is cNM-CCA1 secure where c is a constant. The high level idea for constructing a cNM-CCA1 scheme is to add $c - 1$ layers of encryption atop the basic encryption of a message m , effectively redundantly re-encrypting the previous layer's ciphertext and forming a new layer of encryptions. Intuitively, each parallel query that the adversary asks can help it peel back the security of one of the layers of encryption in the challenge ciphertext, and therefore if a challenge ciphertext is composed of c layers, then the scheme can withstand c parallel queries.

3.1 The Construction

For the base case, we define $\text{NMGen}^{(0)} = \text{gen}$; $\text{NMEnc}^{+(0)}(pk, m, \text{SigVK}) = \text{enc}(pk, m)$; and $\text{NMDec}^{+(0)}(sk, c, \text{SigVK}) = \text{dec}(sk, c)$ where the weakly simulatable and $\mathbf{sPA1}_\ell$ secure encryption primitive $\mathbf{E} = (\text{gen}, \text{enc}, \text{dec})$ is the starting point for our work. Next, we recursively perform multiple redundant parallel encryptions of the last recursive steps output. In each of these steps, we use the standard practice of interpreting the bits of a freshly generated verification key for a one-time signature scheme to choose appropriate public keys with which to encrypt. The resulting set of ciphertexts is finally signed with the one-time signature's signing key to form the final encryption. Decryption proceeds as one might expect: first the signature is checked for validity, and next the encryption is recursively decrypted, where at each level it is ensured that the redundant parallel decryptions all encode the same underlying "message". The encryption scheme $\Pi^{(c)}$ parameterized by an integer $c > 0$ appears in Fig. 4.

3.2 Preliminary Notion

Before we present our security proof, we introduce an intermediate "tagged encryption" security game to simplify our proof. We call this notion (c)NME* security and it allows each ciphertext to have an associated tag used during both encryption and decryption. The challenge ciphertext is tagged with the vector $\vec{0}$, and the adversary can submit any query with a non-zero tag.

Along with this new definition, we present a natural analog of our original encryption scheme $\Pi^{*(c)}$ in Fig. 6. The difference is that the signature

$\text{NMGen}^{(c)}(1^k)$
 1: $(\text{NPK}^{(c-1)}, \text{NSK}^{(c-1)}) \leftarrow \text{NMGen}^{(c-1)}(1^k)$
 2: $\forall i \in [k]$ and $b \in \{0,1\}$, $(pk_i^b, sk_i^b) \leftarrow \text{gen}(1^k)$ s.t. pk_i^b encrypts the range of $\text{NMEnc}^{(c-1)}$
 3: Output $\text{NPK}^{(c)} = \{\text{NPK}^{(c-1)}, \{pk_i^b\}_{\substack{i \in [k] \\ b \in \{0,1\}}}\}$ and $\text{NSK}^{(c)} = \{\text{NSK}^{(c-1)}, \{sk_i^b\}_{\substack{i \in [k] \\ b \in \{0,1\}}}\}$
 $\text{NMEnc}^{(c)}(\text{NPK}^{(c)}, m)$
 1: $(\text{SigSK}, \text{SigVK}) \leftarrow \text{GenKey}(1^k)$
 2: $c \leftarrow \text{NMEnc}^{(c)}(\text{NPK}^{(c)}, m, \text{SigVK})$
 3: $\sigma \leftarrow \text{Sign}_{\text{SigSK}}(c)$
 4: Output $C = (c, \text{SigVK}, \sigma)$
 $\text{NMEnc}^{(c)}(\text{NPK}^{(c)}, m, \text{SigVK})$
 1: Parse $\text{NPK}^{(c)}$ into $(\text{NPK}^{(c-1)}, \vec{pk} = \{pk_1^b, \dots, pk_k^b\}_{b \in \{0,1\}})$
 2: Let SigVK_i be the i th bit of SigVK .
 3: $c_0'' \leftarrow \text{NMEnc}^{(c-1)}(\text{NPK}^{(c-1)}, m, \text{SigVK})$
 4: $c_i' \leftarrow \text{enc}_{pk_i^{\text{SigVK}_i}}(c_0'')$; $\forall i \in [k]$
 5: Output $c = \vec{c}'$
 $\text{NMDec}^{(c)}(\text{NSK}^{(c)}, C)$
 1: Parse C as $(c, \text{SigVK}, \sigma)$ and let SigVK_i be the i th bit of SigVK .
 2: **if** $\text{Verify}_{\text{SigVK}}(\sigma, \vec{c}) = 0$ **then** Output $\perp$
 3: Output $\text{NMDec}^{(c)}(\text{NSK}^{(c)}, c, \text{SigVK})$
 $\text{NMDec}^{(c)}(\text{NSK}^{(c)}, c, \text{SigVK})$
 1: Parse $\text{NSK}^{(c)}$ into $(\text{NSK}^{(c-1)}, \vec{sk} = \{sk_1^b, \dots, sk_k^b\}_{b \in \{0,1\}})$
 2: $\forall i \in [k]$, compute $c_i' \leftarrow \text{dec}_{sk_i^{\text{SigVK}_i}}(c_i)$
 3: **if** $\exists i \in [k]$ s.t. $c_1' \neq c_i'$ **then** Output $\perp$
 4: Output $\text{NMDec}^{(c-1)}(\text{NSK}^{(c-1)}, c_1', \text{SigVK})$

Fig. 4: THE cNM-CCA1 ENCRYPTION SCHEME $\Pi^{(c)}$

scheme used for unduplicatable set selection is replaced by the k -bit tag α .


```

(c)NMEb*(Π*,  $\mathcal{A}$ ,  $\mathcal{D}$ ,  $k$ ,  $p(k)$ )
  1: (NPK, CNSK)  $\leftarrow$  NMGen*(1k)
  2: (m0, m1, S1)  $\leftarrow$  A0NMDec*(NSK,·)(CNPk) s.t. |m0| = |m1|
  The oracle NMDec*(NSK, Y = (y, α)) returns ⊥ if α = 0k
  3: Y*  $\leftarrow$  NMEnc*(NPK, mb, α = 0k)
  4:  $\vec{d}_1 \leftarrow (\perp)$ 
  5: for i = 1 to c
  6:   ( $\vec{Y}$ , Si+1)  $\leftarrow$   $\mathcal{A}_i$ (Y*, Si,  $\vec{d}_i$ ) where | $\vec{Y}$ | = p(k)
  7:   If α ≠ 0k, di+1,j  $\leftarrow$  NMDec*(NSK, Yj = (yj, α))
  8:   Else di+1,j  $\leftarrow$  ⊥;  $\forall j \in [|\vec{Y}|]$ 
  9: Output  $\mathcal{D}(Y^*, \vec{d}_{c+1}, S_{c+1})$ 

```

Fig. 5: THE (c)NME^{*} EXPERIMENT FOR $c \geq 0$

```

NMGen*(c)(1k)
  1: Defined as NMGen(c)(1k) in Fig. 4
NMEnc*(c)(NPK(c), m, α ∈ {0, 1}k)
  2: Return (NMEnc†(c)(NPK(c), m, α), α) where NMEnc†(c) is de-
  fined in Fig. 4,
NMDec*(c)(NSK(c), Y = (y, α ∈ {0, 1}k))
  3: Defined as NMDec†(c)(NSK(c), y, α) in Fig. 4

```

Fig. 6: THE ENCRYPTION SCHEME $\Pi^{*(c)} = (\text{NMGen}^{*(c)}, \text{NMEnc}^{*(c)}, \text{NMDec}^{*(c)})$

As the next lemma shows, there is no difference between these security games; for every adversary in the tagged security game, there exists an equivalently succesful adversary for the (c)NME game.

Lemma 2. *For any ppt adversary $\mathcal{A}$, integer $c > 0$, polynomial p and security parameter k , there exists an adversary $\mathcal{B}$ s.t.*

$$\left\{ (c)\text{NME}_b(\Pi^{(c)}, \mathcal{A}, \mathcal{D}, k, p(k)) \right\}_k \equiv \left\{ (c)\text{NME}_b^*(\Pi^{*(c)}, \mathcal{B}, \mathcal{D}, k, p(k)) \right\}_k$$

Proof. We build a (c)NME^{*} adversary $\mathcal{B}$ that interacts with the (c)NME^{*} experiment by simulating the (c)NME experiment for $\mathcal{A}$. $\mathcal{B}$ receives pk as in Line 2 of the experiment (Fig. 5) and proceeds to generate a pair

of signature keys $(\text{skSig}^*, \text{vkSig}^*) \leftarrow \text{GenKey}(1^k)$. $\mathcal{B}$ then sets $pk' \leftarrow \text{ReArrange}(pk, \text{vkSig}^*)$ where the function ReArrange is presented in Fig. 7. Intuitively this function rearranges the keys in pk and pk' so that $\mathcal{B}$ can sign a challenge ciphertext that it will eventually receive using skSig^* and then produce an encryption according to the $\Pi^{(c)}$ scheme using only the keys in pk . $\mathcal{B}$ runs $\mathcal{A}_0$ on pk' .

```

ReArrange( $pk, \text{vkSig}^*$ )
1: Parse  $pk$  as  $(pk_0, \{pk_i^b\}_{i \in [c \cdot k], b \in \{0,1\}})$ 
2: for  $i \in [0..c-1]$ 
3:   for  $j \in [k]$ 
4:     if  $\text{vkSig}_j^* = 1$  then swap the values  $pk_{i \cdot k + j}^0$  and  $pk_{i \cdot k + j}^1$ 
5:   endFor
6: endFor
7: Return  $pk = (pk_0, \{pk_i^b\}_{i \in [0..ck], b \in \{0,1\}})$ 

```

Fig. 7: THE DEFINITION FOR THE REARRANGE FUNCTION

Whenever $\mathcal{A}_0$ asks a query $Y = (\vec{y}, \sigma, \text{vkSig})$, $\mathcal{B}$ does the followings: $\mathcal{B}$ returns $\perp$ to $\mathcal{A}_0$ as the answer to the query if either $\text{vkSig} = \text{vkSig}^*$ or $\text{Verify}_{\text{vkSig}}(\sigma, \vec{y}) = 0$. Otherwise $\mathcal{B}$ sets $\alpha \leftarrow \oplus(\text{vkSig}^*, \text{vkSig})$ where $\oplus$ is the bitwise XOR function on two vectors of the same length. Intuitively, this finds the right α that shows under which subset of pk keys the vector $\vec{y}$ is encrypted.

$\mathcal{B}$ then asks $(\vec{y}, \alpha)$ to its oracle and forwards the answer to its simulation of $\mathcal{A}_0$. Eventually $\mathcal{A}_0$ returns (m_0, m_1, S_1) and halts. $\mathcal{B}$ outputs (m_0, m_1) to the environment (i.e., its experiment) and receives a challenge ciphertext $(\vec{y}^*, 0^k)$. $\mathcal{B}$ computes $\sigma^* \leftarrow \text{Sign}_{\text{skSig}}(\vec{y}^*)$, sets $Y^* \leftarrow (\vec{y}^*, \sigma^*, \text{vkSig}^*)$, sets $\vec{d}_1 \leftarrow \perp$ and does the following for all $q = 1$ to c :

$\mathcal{B}$ simulates $\mathcal{A}_q$ on the input $(Y^*, S_q, \vec{d}_q)$ and receives in return a vector of ciphertexts $\vec{Y}$ and the state information S_{q+1} . $\mathcal{B}$ computes the decryption to each of Y_i 's in the same way that it computed the decryption to the CCA1 queries with the difference that it asks all of the queries in the same parallel query at once from the environment (instead of asking sequentially). Call the vector of decryption of the queries $\vec{d}_{q+1}$.

Eventually $\mathcal{B}$ outputs $\vec{d}_{c+1}$ and S_{c+1} and halts. $\square$

3.3 Main Theorem

The heart of our main theorem relies on Lemma 3 (introduced shortly) which, informally, shows that for any ppt adversary $\mathcal{A}$, there exists a ppt adversary $\mathcal{B}$ such that

$$\left\{ (c)\text{NME}_b^*(\Pi^{*(c)}, \mathcal{A}, \mathcal{D}, k, p(k)) \right\}_{b,k} \approx_c \left\{ (c-1)\text{NME}_b^*(\Pi^{*(c-1)}, \mathcal{B}, \mathcal{D}, k, p(k)) \right\}_{b,k}.$$

Assuming Lemma 3, we state and give the proof our main theorem below. We then formally state and prove Lemma 3.

Theorem 2. *If the encryption scheme $E = (\text{gen}, \text{enc}, \text{dec})$ is weakly simulatable and sPA1_ℓ secure, then for any integer $c > 0$, construction $\Pi^{(c)} = (\text{NMG}^{(c)}, \text{NMEnc}^{(c)}, \text{NMDec}^{(c)})$ in Fig. 4 is $(c)\text{NME}$ secure.*

Proof. By applying Lemma 3 c times, we have that

$$\left\{ (c)\text{NME}_0^*(\Pi^{*(c)}, \mathcal{A}, \mathcal{D}, k, p(k)) \right\}_k \stackrel{\text{Lemma 3}}{\approx_c} \cdots \stackrel{\text{Lemma 3}}{\approx_c} \left\{ (0)\text{NME}_0^*(\Pi^{*(0)}, \mathcal{B}, \mathcal{D}, k, p(k)) \right\}_k$$

In Claim 1 (below), we show that construction $\Pi^{*(0)}$ is $(0)\text{NME}^*$ -secure, and thus it follows that

$$\left\{ (c)\text{NME}_0^*(\Pi^{*(c)}, \mathcal{A}, \mathcal{D}, k, p(k)) \right\}_k \approx_c \left\{ (0)\text{NME}_1^*(\Pi^{*(0)}, \mathcal{B}, \mathcal{D}, k, p(k)) \right\}_k$$

Applying Lemma 3 again on the right hand side, it follows that

$$\left\{ (c)\text{NME}_0^*(\Pi^{*(c)}, \mathcal{A}, \mathcal{D}, k, p(k)) \right\}_k \approx_c \left\{ (c)\text{NME}_1^*(\Pi^{*(c)}, \mathcal{A}, \mathcal{D}, k, p(k)) \right\}_k$$

Finally applying Lemma 2 to show equivalence between $(c)\text{NME}$ and $(c)\text{NME}^*$ completes the theorem. $\square$

Claim 1. *If the encryption scheme $E = (\text{gen}, \text{enc}, \text{dec})$ is weakly simulatable and sPA1_ℓ secure, then for all ppt adversaries and distinguishers $\mathcal{A}$ and $\mathcal{D}$ respectively and for all polynomials p :*

$$\left\{ (0)\text{NME}_0^*(\Pi^{*(0)}, \mathcal{A}, \mathcal{D}, k, p(k)) \right\}_k \approx_c \left\{ (0)\text{NME}_1^*(\Pi^{*(0)}, \mathcal{A}, \mathcal{D}, k, p(k)) \right\}_k$$

where the experiment $(0)\text{NME}^*$ is defined in Fig. 5 and the encryption scheme $\Pi^{*(0)} = (\text{NMG}^{*(0)}, \text{NMEnc}^{*(0)}, \text{NMDec}^{*(0)})$ is defined in Fig. 6.

Proof. By definition, notice that $\text{NMEnc}^{*(0)} = \text{NMEnc}^{+(0)}(pk, m, \alpha) = \text{enc}(pk, m)$; in fact, $\Pi^{*(0)}$ is equivalent to **E**. Second, the $(0)\text{NME}^*$ experiment has no parallel queries after the challenge has been submitted, and is therefore (roughly) equivalent to the CCA1-security game. By assumption, **E** is sPA1_ℓ -secure and therefore CCA1-secure, which completes the claim. $\square$

It remains to prove the key technical lemma; we first present a high-level overview of the proof. Our goal is to show how to simulate an Adversary $\mathcal{A}$ that makes c parallel decryption queries when given a c -layered challenge ciphertext, with an adversary $\mathcal{B}$ that only has access to $c - 1$ parallel decryption queries, and a $c - 1$ layered challenge ciphertext. It is easy to see how we can simulate the extra layer of the challenge ciphertext, $\mathcal{B}$ can simply generate its own keys and add an extra layer to its challenge ciphertext to simulate $\mathcal{A}$. The question remains how to simulate the extra parallel decryption that $\mathcal{A}$ has access to. It may seem, on first glance, to follow immediately for the sPA1_ℓ security of the underlying encryption scheme, because the whole purpose of an extractor is to simulate a decryption oracle. However, $\mathcal{A}$ is fed a challenge ciphertext which it did not produce (and thus there is no extraction guarantee), and $\mathcal{A}$ might create its parallel decryption queries based on the challenge ciphertext, in which case there is no a priori reason to believe that $\text{Ext}_\mathcal{A}$ will be able to “decrypt” properly when used to decrypt the “extra” c^{th} parallel decryption query.

To solve this issue, we use the non-duplicatable set selection to ensure that there is a new public-key with respect to which the adversary must have generated part of the ciphertext (and not just mauled part of the challenge ciphertext); we can then be assured that the extractor will work on this portion of the challenge ciphertext. However, this by itself does not allow us to simulate the consistency check in the decryption algorithm that ensures that all of the encryptions at a given level are of the same message. For the outer layer of ciphertexts that need to be decrypted, we have the corresponding secret-keys since $\mathcal{B}$ generated the corresponding public-keys. The inner-layers are another matter entirely. In order to argue this, we use the sPA1_ℓ^+ security of the underlying encryption scheme in conjunction with the fact that it is weakly simulatable. In essence this means that the extractor cannot tell the difference between when the outer layer of the challenge ciphertext is legitimate encryptions and when they were instead created on demand using the simulator with randomness provided via an oracle. However, in the latter case, by the definition of sPA1_ℓ^+ security, the extractor must function.

There is one last subtlety, which is that due to technical requirements of the proof, we actually do not necessarily have access to the secret-keys for the outer layer of the encryptions of $\mathcal{A}$ when we need it, and therefore cannot perform the outer layer consistency check via the extractor. It suffices for this check to be done in a hybrid experiment using the actual secret-key (independent of where it comes from). However, we need to ensure that the responses from these consistency checks do not affect the viability of finding a suitable extractor. Here, the fact that we have $p(k)$ parallel decryption queries to simulate, as opposed to $p(k)$ adaptive queries, is used. Essentially, we consider a system which decrypts all of the parallel queries via the extractor, ignoring the initial consistency checks.

Lemma 3. *For any integer $c > 0$, any ppt adversary $\mathcal{A}$, polynomial p and security parameter k , there exists a ppt adversary $\mathcal{B}$ such that*

$$\left\{ (c)\text{NME}_b^*(\Pi^{*(c)}, \mathcal{A}, \mathcal{D}, k, p(k)) \right\}_{b,k} \approx_c \left\{ (c-1)\text{NME}_b^*(\Pi^{*(c-1)}, \mathcal{B}, \mathcal{D}, k, p(k)) \right\}_{b,k}$$

Proof. Consider the following hybrid experiment:

Experiment $\text{H}_b^*(\Pi^{*(c)}, \mathcal{A}, \mathcal{D}, k, p(k))$ proceeds similarly to $(c)\text{NME}_b^*$ with the difference that the former experiment handles the decryption of all ciphertexts up to the second parallel query differently. After all of the public keys have been generated, initialize $st \leftarrow (\{pk_i\}_{i \in [2ck+1]}, R_{\mathcal{A}})$ where $R_{\mathcal{A}}$ are the random coins that will be used to run $\mathcal{A}$. For all CCA1 queries that $\mathcal{A}$ makes (i.e., queries that are made before the challenge ciphertext is produced), everytime that NMDec^* calls the decryption function dec on y_i , the experiment calls $\text{Ext}_{\mathcal{A}}(st, y_i, \cdot)$ with the appropriate pk as the third argument. After $\mathcal{A}$ receives the challenge ciphertext, the *first* parallel query $\{d_i\}$ is decrypted using NMDecAlt defined below (without loss of generality, assume that $d_i = (\vec{C}, \sigma, \text{vkSig})$). The remaining $(c-1)$ parallel queries are decrypted as per $(c)\text{NME}^*$.

$\text{NMDecAlt}(d_i = (\vec{C}, \alpha))$:

- 1: If $\alpha = 0^k$ output $\perp$, else let i' be the first index at which $\alpha_{i'} \neq 0$.
- 2: For $i \in [k]$, do $C'_i \leftarrow \text{dec}_{sk_i^{\alpha_i}}(C_i)$
- 3: Call $Y^{(c-1)} \leftarrow \text{Ext}_{\mathcal{A}}(st, C_{i'}, pk_{i'}^{\alpha_{i'}})$
(notice that $Y^{(c-1)} = (y_1^{(c-1)}, \dots, y_k^{(c-1)})$ is a vector).
- 4: $m \leftarrow \text{ExtractAll}(pk, Y^{(c-1)}, (c-1), \alpha)$
- 5: If $\exists j$ s.t. $C'_1 \neq C'_j$, return $\perp$. Else return m

$\text{ExtractAll}(pk, Y = (y_1^c, \dots, y_k^c), c, \alpha)$:

```

1: for  $i = c - 1$  to  $0$ 
2:   for  $j = 1$  to  $k$ 
3:      $y_j^i \leftarrow \text{Ext}_{\mathcal{A}}(st, y_j^{(i+1)}, pk_{(i+1) \cdot k + j}^{\alpha_j})$ 
4:   if  $\exists d \in [k]$  s.t.  $y_1^i \neq y_d^i$  return  $\perp$ 
5:    $m \leftarrow \text{Ext}_{\mathcal{A}}(st, y_1^0, pk_0)$ 
6: Return  $m$ 

```

Intuitively ExtractAll submits the inner layer of the query $Y^{(c-1)}$ to the extractor to be decrypted under the appropriate keys until it reaches the innermost layer containing message m .

To define the function extractor $\text{Ext}_{\mathcal{A}}$ used in NMDecAlt above, we first define an $\mathbf{sPA1}_{2ck+1}^+$ ciphertext creator $\text{Crt}_{\mathcal{A}}$ (which makes calls to an extractor oracle) that roughly mimics the queries made by adversary $\mathcal{A}$ in the H_b experiment. We define $\text{Crt}_{\mathcal{A}}$ as follows:

1. $\text{Crt}_{\mathcal{A}}$ receives $2ck + 1$ public-keys $pk = (pk_0, \{pk_i^b\}_{i \in [1 \dots ck], b \in \{0,1\}})$ from the $\mathbf{sPA1}_{2ck+1}^+$ experiment. $\text{Crt}_{\mathcal{A}}$ reads its random tape as $R_{\mathcal{A}}$ and runs $\mathcal{A}_0(pk)$ using tape $R_{\mathcal{A}}$.
2. Whenever $\text{Crt}_{\mathcal{A}}$ receives a query $(\{y_i\}_{i \in [k]}, \alpha)$ from $\mathcal{A}_0$, it returns $\perp$ if $\alpha = 0^k$. Otherwise, $\text{Crt}_{\mathcal{A}}$ submits each $(y_i, pk_i^{\alpha_i})$ to its extractor. If all of the queries do not decrypt to the same value, $\text{Crt}_{\mathcal{A}}$ returns $\perp$ to $\mathcal{A}_0$ as the answer to that query. Call the decrypted value $Y^{(c-1)}$ and notice that it should be a vector of ciphertexts encrypted under k public keys in pk . Next $\text{Crt}_{\mathcal{A}}$ calls $m \leftarrow \text{ExtractAll}(pk, \alpha, Y^{(c-1)}, c - 1)$. $\text{Crt}_{\mathcal{A}}$ returns m to $\mathcal{A}_0$ as the answer to the query. Eventually $\mathcal{A}_0$ returns (m_0, m_1, St) and halts.
3. $\text{Crt}_{\mathcal{A}}$ accesses its oracle $\mathcal{O}$ to generate k blocks of ℓ random bits $(x_1, \dots, x_k)$ and then computes the vector $\vec{y} = (f(pk_{k(c-1)+1}^0, x_1), \dots, f(pk_{k(c-1)+k}^0, x_k))$. $\text{Crt}_{\mathcal{A}}$ then runs $\mathcal{A}_1(y^*, St)$ where $y^* = (\vec{y}, 0^k)$ and St is the state information returned by $\mathcal{A}_0$.
4. $\mathcal{A}_1$ returns a vector of ciphertexts $\vec{Y}$ and the state information S and halts. For each query $Y_j = (\{y_i\}_{i \in [k]}, \alpha)$, $\text{Crt}_{\mathcal{A}}$ executes steps 1, 3, and 4 of procedure NMDecAlt to decrypt the message. After each ciphertext in the first parallel query has been decrypted in this way, $\text{Crt}_{\mathcal{A}}$ halts.

The $\mathbf{sPA1}_\ell^+$ security of $\mathbf{E}$ implies there exists an extractor $\text{Ext}_\mathcal{A}$ whose answers to the decryption queries submitted by $\text{Crt}_\mathcal{A}$ are indistinguishable from their true decryptions. We have now defined $\text{Ext}_\mathcal{A}$ used in NMDecAlt . Notice that $\text{Crt}_\mathcal{A}$ does not exactly simulate $\mathcal{A}$'s view in H_b ; we will argue below why $\text{Ext}_\mathcal{A}$ continues to work properly when it is used in H_b .

Claim 2. For $b \in \{0, 1\}$, $\{(c)\text{NME}_b^* \left(\Pi^{(c)}, \mathcal{A}, \mathcal{D}, k, p(k) \right)\}_{k \in \mathbb{N}} \approx_c \{\text{H}_b \left(\Pi^{(c)}, \mathcal{A}, \mathcal{D}, k, p(k) \right)\}_{k \in \mathbb{N}}$

Proof. Experiments $(c)\text{NME}_b^*$ and H_b differ only if $\text{Ext}_\mathcal{A}$ answers with an incorrect decryption in the latter experiment. The assumption on the $\mathbf{sPA1}_\ell$ -security of $\mathbf{E}$ implies:

$$\Pr[\mathbf{sPA1}_\ell^+(\Pi^{*(c)}, \text{Crt}_\mathcal{A}, \text{Ext}_\mathcal{A}, k) = 0] \leq \mu_1(k) \quad (5)$$

for some negligible μ_1 and therefore $\text{Ext}_\mathcal{A}$ correctly answers all of the queries issued by $\text{Crt}_\mathcal{A}$ with very high probability (recall that the $\mathbf{sPA1}$ random variable being 0 corresponds to an incorrect decryption event). As mentioned, $\text{Crt}_\mathcal{A}$ does not exactly mimic $\mathcal{A}$'s view in H_b and so it is not obvious that $\text{Ext}_\mathcal{A}$ answers correctly in H_b . The two notable differences are that (i) $\text{Crt}_\mathcal{A}$ uses the weak simulatability of the base encryption scheme to create the challenge ciphertext instead of using enc to produce the challenge, and (ii) $\text{Crt}_\mathcal{A}$ does not perform a consistency check that all $C'_1 = C'_j$ before decrypting the inner ciphertext but instead uses the extractor on the outer layer at a position in which α differs from 0^k and then uses the ExtractAll method on the resulting inner ciphertext.

In order to handle the first difference, we analyze $\Pr[\mathbf{sPA1}_\ell^{++}(\Pi^{*(c)}, \text{Crt}_\mathcal{A}, \text{Ext}_\mathcal{A}, k) = 0]$ where $\mathbf{sPA1}_\ell^{++}$ is an experiment identical to $\mathbf{sPA1}_\ell^+$ with two differences:

1. First, a random bit d is selected and fixed for the remainder of the game.
2. The oracle $\mathcal{O}$ returns random bits as follows: when $\text{Crt}_\mathcal{A}$ accesses the oracle $\mathcal{O}$ for the i^{th} time, instead of $r \in \{0, 1\}^l$, $\mathcal{O}$ returns $f^{-1}(pk_{k(c-1)+i}^0, \text{enc}_{pk_{k(c-1)+i}^0}(m_d))$.

We argue that $\text{Ext}_\mathcal{A}$ answers all queries correctly in these two games must be negligibly close by the weak-simulatability property:

Claim 3. $|\Pr[\mathbf{sPA1}_\ell^+(\Pi^{*(c)}, \text{Crt}_\mathcal{A}, \text{Ext}_\mathcal{A}, k) = 0] - \Pr[\mathbf{sPA1}_\ell^{++}(\Pi^{*(c)}, \text{Crt}_\mathcal{A}, \text{Ext}_\mathcal{A}, k) = 0]| < \mu(k)$.

Proof. Consider the weak-simulatability adversary $\mathcal{B}$ defined as follows:

(Recall that in the first step, the weak-simulatability challenger samples k pairs of keys $(pk_i, sk_i) \leftarrow \text{gen}(1^k)$ for $i \in [k]$ and a random bit b .) The attacker $\mathcal{B}$ receives k public keys which we call $\{pk_{k(c-1)+i}^0\}_{i \in [k]}$ from the environment. $\mathcal{B}$ then samples another $k + 2(c-1)k + 1$ random keys using the gen algorithm and fresh random coins to build the public key $pk = \{pk_0, (pk_i^b, sk_i^b)_{i \in [ck], b \in \{0,1\}}\}$ (notice that $\{pk_{k(c-1)+i}^0\}_{i \in [k]}$ in pk are received from the environment and the rest are generated randomly). $\mathcal{B}$ samples random coins $R_{\mathcal{A}}$ for $\mathcal{A}$ and runs step (2) of the description of $\text{Crt}_{\mathcal{A}}$ using $\text{Ext}_{\mathcal{A}}$. Eventually $\mathcal{A}$ writes (m_0, m_1) to its write-only tape. $\mathcal{B}$ randomly chooses $d \in \{0,1\}$ and stores $c'_d = \text{NMEnc}^{+(c-1)}(pk', m_d, 0^k)$ where $pk' = \{pk_0, (pk_i^b, sk_i^b)_{i \in [(c-1)k], b \in \{0,1\}}\}$ (notice that pk' is the public key for the inner layer of ciphertexts encrypted under pk for the $\Pi^{*(c)}$ encryption scheme and c'_d is the inner layer for an encryption of m_d under pk). For ease of notation, we refer to $\{pk_{k(c-1)+i}^0\}_{i \in [k]}$ as $\vec{pk}''$ (these are the public keys for the outer layer of the challenge ciphertext). Next the challenger samples $r_i \in \{0,1\}^l$ for $1 \leq i \leq k$ and returns $\{y_i = (r_i, f(pk_i'', r_i))\}_{i \in [k]}$ if $b = 0$, and $\{y_i = (f(pk_i'', c_i = \text{enc}_{pk_i''}(c'_d)), c_i)\}_{i \in [k]}$ if $b = 1$. $\mathcal{B}$ then simulates step (3) of $\text{Crt}_{\mathcal{A}}$ by running $\mathcal{A}_1(y^*, St)$ where $y^* = (\vec{y}, 0^k)$ and St is the state information returned by $\mathcal{A}_0$. $\mathcal{A}_1$ returns a vector of ciphertexts $\vec{Y}$ and the state information S and halts. $\mathcal{B}$ runs step (4) of $\text{Crt}_{\mathcal{A}}$ on $\vec{Y}$. Finally attacker $\mathcal{B}$ outputs $b' = 0$ if all the queries made to $\text{Ext}_{\mathcal{A}}$ so far were answered correctly and $b' = 1$ otherwise. This check can be done by using the secret keys for the spots in pk that are generated by $\mathcal{B}$ (all of them except $\{pk_{k(c-1)+i}^0\}_{i \in [k]}$ which is received from the environment) because after $\mathcal{A}$ returns (m_0, m_1) , the only queries that it asks to its extractor are with respect to ciphertexts encrypted under the mentioned keys in pk .

The case $b = 0$ corresponds to experiment $\mathbf{sPA1}_{\ell}^+(\Pi^{*(c)}, \text{Crt}_{\mathcal{A}}, \text{Ext}_{\mathcal{A}}, k)$ and the case $b = 1$ corresponds to $\mathbf{sPA1}_{\ell}^{++}(\Pi^{*(c)}, \text{Crt}_{\mathcal{A}}, \text{Ext}_{\mathcal{A}}, k)$. For convenience, in the following equations, we abbreviate the two experiments as $\mathbf{sPA1}_{\ell}^+$ and $\mathbf{sPA1}_{\ell}^{++}$ respectively. It follows that:

$$\begin{aligned} \Pr[\text{DIST}_{\mathbf{E}'}((f, f^{-1}), k, \mathcal{B}) = 1] &= \Pr[b = 0] \cdot \Pr[\mathbf{sPA1}_{\ell}^+ = 0] + \Pr[b = 1] \cdot \Pr[\mathbf{sPA1}_{\ell}^{++} = 1] \\ &= (1/2) \Pr[\mathbf{sPA1}_{\ell}^+ = 0] + (1/2)(1 - \Pr[\mathbf{sPA1}_{\ell}^{++} = 0]) \\ &= 1/2 + 1/2(\Pr[\mathbf{sPA1}_{\ell}^+ = 0] - \Pr[\mathbf{sPA1}_{\ell}^{++} = 0]) \end{aligned}$$

Since the weak-simulatability property of $\mathbf{E}'$ implies that $|\Pr[\text{DIST}_{\mathbf{E}'}((f, f^{-1}), k, \mathcal{B}) =$

$1] - 1/2| \leq \mu(k)$ for some negligible function μ , it must then follow that

$$|\Pr[\mathbf{sPA}_{\ell}^+ = 0] - \Pr[\mathbf{sPA}_{\ell}^{++} = 0]| \leq 2\mu(k)$$

which completes the proof of the claim. $\square$

Combining (5) with Claim 2 implies that $\Pr[\mathbf{sPA}_{\ell}^{++} = 0] \leq \mu_2(k)$ for another negligible function μ_2 . Moreover, by Bayes rule, we can conclude that there exists another negligible function μ_3 such that both $\Pr[\mathbf{sPA}_{\ell}^{++} = 0 \mid d = 0] \leq \mu_3(k)$ and $\Pr[\mathbf{sPA}_{\ell}^{++} = 0 \mid d = 1] \leq \mu_3(k)$; i.e., the possibility for incorrect extraction results remains negligible no matter which of m_0 or m_1 is used in the $\mathbf{sPA}_{\ell}^{++}$ experiment.

In order to handle (ii), we observe that $\text{Ext}_{\mathcal{A}}$ only receives queries generated by the first parallel query in both $\mathbf{sPA}_{\ell}^{++}$ and experiment H_b . From the beginning of both experiments and up to the point of the challenge ciphertext generation, the initial state st and the distribution of queries fed to $\text{Ext}_{\mathcal{A}}$ in H_b is identical to those in experiment $\mathbf{sPA}_{\ell}^{++}$. (This explains why it is necessary for experiment H_b to make “dummy” calls to $\text{Ext}_{\mathcal{A}}$ for every call to dec during the decryption of the CCA1 queries.)

When the ciphertext is generated, since $f(f^{-1}(c)) = c$, that challenge ciphertext in experiments H_b and $\mathbf{sPA}_{\ell}^{++}$ conditioned on $d = b$ will also be identically distributed, and this implies that the parallel query that $\mathcal{A}_1$ issues will also be identically distributed. Once this parallel query has been fixed, the queries that are sent to $\text{Ext}_{\mathcal{A}}$ are also fixed in both experiments. By inspection, again because $\text{NMDec}^{*(c)*}$ issues dummy queries to $\text{Ext}_{\mathcal{A}}$ during the decryption of the outer layer, it follows that the query distribution will be identical, and the claim follows. Thus, the fact that $\text{Crt}_{\mathcal{A}}$ does not perform the same consistency check is irrelevant since the same distribution of queries is fed to the extractor in both experiments.

It then follows that with high probability, all of the responses from $\text{Ext}_{\mathcal{A}}$ in H_b coincide with the true decryption, and therefore $(c)\text{NME}_b^*$ and H_b also output the same value which concludes the Claim. $\square$

Claim 4. *For any ppt adversary $\mathcal{A}$, polynomial p and security parameter k , there exists an adversary $\mathcal{B}$ such that*

$$H_b(\Pi^{*(c)}, \mathcal{A}, \mathcal{D}, k, p(k)) \equiv (c-1)\text{NME}_b^*(\Pi^{*(c-1)}, \mathcal{B}, \mathcal{D}, k, p(k))$$

Proof. We build the $(c-1)\text{NME}^*$ adversary $\mathcal{B}$ as follows: $\mathcal{B}$ receives the public-key $\text{NPK}^{*(c-1)} = (pk'_0, \vec{pk}' = \{pk'_i\}_{i \in [(c-1)k], b \in \{0,1\}})$ from the environment and generates another $2k$ keys as $(pk''_b, sk''_b) \leftarrow \text{gen}(1^k)$ for $i \in [k]$

and $b \in \{0, 1\}$. Let $pk = \{pk'_0, \vec{pk}', \vec{pk}''\}$. $\mathcal{B}$ generates random coins $R_{\mathcal{A}}$ and initializes $st \leftarrow (\{pk_i\}_{i \in [2ck+1]}, R_{\mathcal{A}})$. $\mathcal{B}$ runs $\mathcal{A}_0(pk; R_{\mathcal{A}})$.

For any CCA1 query $Y = (\vec{y}, \alpha)$ that $\mathcal{A}_0$ submits, $\mathcal{B}$ runs NMDecAlt using $\vec{sk}''$ to decrypt the outer layer. Eventually $\mathcal{A}_0$ returns (m_0, m_1) and the state information S_1 and halts. $\mathcal{B}$ then forwards (m_0, m_1) to the environment and receives a challenge ciphertext $Y' = (\vec{y}', 0^k)$ from the environment. $\mathcal{B}$ computes $\{y_i^* \leftarrow \text{enc}(pk_i'', \vec{y}')\}_{i \in [k]}$ and sets $Y^* = (\vec{y}^*, 0^k)$. $\mathcal{B}$ also computes $\{r_i \leftarrow f^{-1}(pk_i'', y_i^*)\}_{i \in [k]}$ and forwards $\{r_i\}_{i \in [k]}$ to $\text{Ext}_{\mathcal{A}}$ and runs $\mathcal{A}_1$ on the challenge ciphertext Y^* and the state information S_1 . Eventually $\mathcal{A}_1$ returns a vector of queries (the first parallel query) $\vec{Y}$ and the state information S_2 and halts. $\mathcal{B}$ submits each Y_i to the extractor and receives a value which we call m' from it. $\mathcal{B}$ then uses $\vec{sk}''$ to check that all the ciphertexts in the outer layer of Y_i decrypts to the same value. If so, it sets m' as the decryption of that query otherwise it sets $\perp$ as the decryption of that query. We refer to the resulting vector of decryptions to $\vec{Y}$ as $\vec{d}_1$.

For all $q = 2$ to c , $\mathcal{B}$ runs $\mathcal{A}_q$ on Y^* , S_2 and $\vec{d}_q$ and gets in return a vector of ciphertexts $\vec{Y}$ and the state information S_{q+1} . $\mathcal{B}$ decrypts $Y_i = (\vec{y}, \alpha)$ as follows: the decryption is $\perp$ if $\alpha = 0^k$. Otherwise $\mathcal{B}$ uses $\vec{sk}''$ to check all the ciphertexts in the outer layer of Y_i decrypts to the same value y_0 . If not, the decryption to this query will be $\perp$. Otherwise $\mathcal{B}$ sets $Y'_i = (y_0, \alpha)$ and moves to the next query. After processing all queries, $\mathcal{B}$ submits $\vec{Y}'$ to the environment and gets in return a vector of decryptions to the $\vec{Y}'$. Using these answers and the results from the checks, $\mathcal{B}$ sets $\vec{d}_i$ (which is the vector of the decryption to $\vec{Y}_i$). Eventually $\mathcal{B}$ outputs $\vec{d}_{c+1}$ and the state information S_{c+1} and halts.

$\mathcal{B}$ performs a perfect simulation of the $H_b(\Pi^{*(c)}, \mathcal{A}, \mathcal{D}, k, p(k))$ experiment, and thus the claim follows. $\square$

Combining Claim 2 and Claim 4 completes the proof of Lemma 3. $\square$

3.4 Remarks about the proof

In our construction, we use the k -bit outer-most signature SigVK to pick the unduplicatable set for each of the c layers of encryption. Not only is this choice an efficiency improvement in that only one signature key is needed (instead of c), it is also a critical feature of our proof. This point

is used in Claim 4. Adversary $\mathcal{B}$ must not submit a 0-tag query to its $(c-1)$ NME challenger; but if each layer could use a different α tag, then $\mathcal{A}$ might select 0 as the tag for the $(c-1)$ layer and therefore prevent $\mathcal{B}$ from submitting it to its oracle.

4 Conclusions

We have shown both the first construction of a non-malleable CCA1 encryption scheme from a seemingly weaker primitive, and that it is possible to realize cNM-CCA1 schemes without achieving full CCA2 security. All of our constructions are black-box, although based on hardness assumptions that are not efficiently falsifiable. Major open questions in the area are clearly if CPA security implies CCA1 security, CCA1 security implies CCA2, or the transitive closure. Progress on any of these questions, with either black-box or white-box constructions (or impossibility results), would be of foundational importance to the field.

4.1 Acknowledgements

We thank the NSF (grant 0939718), DARPA and AFRL (joint contract FA8750-11-2-0211) for their gracious support.

References

- [1] Mihir Bellare and Adriana Palacio. Towards plaintext-aware public-key encryption without random oracles. In Pil Joong Lee, editor, *ASIACRYPT*, volume 3329 of *Lecture Notes in Computer Science*, pages 48–62. Springer, 2004.
- [2] Ran Canetti, Oded Goldreich, and Shai Halevi. The random oracle methodology, revisited. *J. ACM*, 51(4):557–594, July 2004.
- [3] Seung Geol Choi, Dana Dachman-Soled, Tal Malkin, and Hoeteck Wee. Black-box construction of a non-malleable encryption scheme from any semantically secure one. In Ran Canetti, editor, *TCC*, volume 4948 of *Lecture Notes in Computer Science*, pages 427–444. Springer, 2008.
- [4] Ronald Cramer, Goichiro Hanaoka, Dennis Hofheinz, Hideki Imai, Eike Kiltz, Rafael Pass, Abhi Shelat, and Vinod Vaikuntanathan.

- Bounded cca2-secure encryption. In Kaoru Kurosawa, editor, *ASIACRYPT*, volume 4833 of *Lecture Notes in Computer Science*, pages 502–518. Springer, 2007.
- [5] Ronald Cramer and Victor Shoup. Design and analysis of practical public-key encryption schemes secure against adaptive chosen ciphertext attack. *SIAM J. Comput.*, 33(1):167–226, 2003.
 - [6] Ivan Damgård. Towards practical public key systems secure against chosen ciphertext attacks. In Joan Feigenbaum, editor, *CRYPTO*, volume 576 of *Lecture Notes in Computer Science*, pages 445–456. Springer, 1991.
 - [7] Ivan Damgård and Jesper Buus Nielsen. Improved non-committing encryption schemes based on a general complexity assumption. In Mihir Bellare, editor, *CRYPTO*, volume 1880 of *Lecture Notes in Computer Science*, pages 432–450. Springer, 2000.
 - [8] Alexander W. Dent. The cramer-shoup encryption scheme is plaintext aware in the standard model. In Serge Vaudenay, editor, *EUROCRYPT*, volume 4004 of *Lecture Notes in Computer Science*, pages 289–307. Springer, 2006.
 - [9] Alexander W. Dent. The hardness of the dhk problem in the generic group model. *IACR Cryptology ePrint Archive*, 2006:156, 2006.
 - [10] Danny Dolev, Cynthia Dwork, and Moni Naor. Nonmalleable cryptography. *SIAM J. Comput.*, 30(2):391–437, 2000.
 - [11] Shafi Goldwasser and Silvio Micali. Probabilistic encryption. *Journal of computer and system sciences*, 28(2):270–299, 1984.
 - [12] Takahiro Matsuda and Kanta Matsuura. Parallel decryption queries in bounded chosen ciphertext attacks. In Dario Catalano, Nelly Fazio, Rosario Gennaro, and Antonio Nicolosi, editors, *Public Key Cryptography*, volume 6571 of *Lecture Notes in Computer Science*, pages 246–264. Springer, 2011.
 - [13] Steven Myers and Abhi Shelat. Bit encryption is complete. In *FOCS*, pages 607–616. IEEE Computer Society, 2009.
 - [14] Moni Naor. On cryptographic assumptions and challenges. In Dan Boneh, editor, *CRYPTO*, volume 2729 of *Lecture Notes in Computer Science*, pages 96–109. Springer, 2003.

- [15] Moni Naor and Moti Yung. Public-key cryptosystems provably secure against chosen cypher-text attack. In *STOC'90*, pages 427–437, 1990.
- [16] Rafael Pass, Abhi Shelat, and Vinod Vaikuntanathan. Construction of a non-malleable encryption scheme from any semantically secure one. In Cynthia Dwork, editor, *CRYPTO*, volume 4117 of *Lecture Notes in Computer Science*, pages 271–289. Springer, 2006.
- [17] C. Rackoff and D. R. Simon. Non-interactive zero-knowledge proof of knowledge and chosen ciphertext attack. In *CRYPTO'91*, pages 433–444, 1991.

A Instantiating weak-simulatability

Algorithm 1: DEG $\mathcal{G}(1^k)$ 1: $(p, q, g) \leftarrow G(1^k)$ 2: $x_1 \leftarrow \mathbb{Z}_q; X_1 \leftarrow g^{x_1} \bmod p$ 3: $x_2 \leftarrow \mathbb{Z}_q; X_2 \leftarrow g^{x_2} \bmod p$ 4: Return $(pk = (p, q, g, X_1, X_2),$ $sk = (p, q, g, x_1, x_2))$ $\mathcal{E}(pk, M)$ 1: $y \leftarrow \mathbb{Z}_q; Y \leftarrow g^y \bmod p$ 2: $W \leftarrow X_1^y; V \leftarrow X_2^y \bmod p$ 3: $U \leftarrow V \cdot M \bmod p$ 4: Return $C = (Y, W, U)$ $\mathcal{D}(sk, C)$ 1: if $W \neq Y^{x_1} \bmod p$ then Return $\perp$ 2: Return $M \leftarrow U \cdot Y^{-x_2} \bmod p$	Algorithm 2: CS-Lite $\mathcal{G}(1^k)$ 1: $(p, q, g_1) \leftarrow G(1^k); g_2 \leftarrow G_q \setminus \{1\}$ 2: $x_1 \leftarrow \mathbb{Z}_q; x_2 \leftarrow \mathbb{Z}_q; z \leftarrow \mathbb{Z}_q$ 3: $X \leftarrow g_1^{x_1} \cdot g_2^{x_2} \bmod p; Z \leftarrow g_1^z \bmod p$ 4: Return $(pk = (p, q, g_1, g_2, X, Z),$ $sk = (p, q, g_1, g_2, x_1, x_2, z))$ $\mathcal{E}(pk, M)$ 1: $r \leftarrow \mathbb{Z}_q$ 2: $R_1 \leftarrow g_1^r \bmod p; R_2 \leftarrow g_2^r \bmod p$ 3: $E \leftarrow Z^r \cdot M \bmod p; V \leftarrow X^r \bmod p$ 4: Return $C = (R_1, R_2, E, V)$ $\mathcal{D}(sk, C)$ 1: if $V \neq R_1^{x_1} \cdot R_2^{x_2} \bmod p$ then Return $\perp$ 2: Return $M \leftarrow E \cdot R_1^{-z} \bmod p$

Fig. 8: THE ENCRYPTION SCHEMES DEG AND CS-LITE

Definition 5. (Simulatable Group) [8] A family of groups $\{G_k\}_{k \in \mathbb{N}}$ is simulatable if there exist two poly-time functions (h, h^{-1}) and a polynomial ℓ , such that the following correctness requirements are met.

1. $\forall k, \forall r \in \{0, 1\}^{\ell(k)}, h(r) \in G_k.$
2. h^{-1} is probabilistic. $\forall k, \forall \alpha \in G_k, h^{-1}(\alpha) \in \{0, 1\}^{\ell(k)}.$

3. $\forall k, h(h^{-1}(\alpha)) = \alpha$ for all $\alpha \in G_k$.

Similarly, the following two security requirements are met, for all probabilistic distinguisher $\mathcal{A}$ and all $k \in \mathbb{N}$, there exists a negligible function ϵ such that: $\Pr[\text{INV}_{\mathcal{A}}(k, (h, h^{-1})) = 1] \leq 1/2 + \epsilon(k)$ and $\Pr[\text{IND}_{\mathcal{A}\{G\}}(k, h) = 1] \leq 1/2 + \epsilon(k)$, where the experiments are defined in Fig. 9.

$\text{INV}_{\mathcal{A}}(k, (h, h^{-1}))$
1: $b \leftarrow \{0, 1\}$
2: $b' \leftarrow \mathcal{A}^{\mathcal{O}_{h, h^{-1}, b}}(1^k)$, where oracle $\mathcal{O}_{h, h^{-1}, b}$ responds to a query by
sampling $r \in \{0, 1\}^{l(k)}$, and returning r if $b = 0$ or $h^{-1}(h(r))$ if $b = 1$
3: Output 1 iff $b = b'$

$\text{IND}_{\mathcal{A}\{G\}}(k, h)$
1: $b \in \{0, 1\}$
2: $b' \leftarrow \mathcal{A}^{\mathcal{O}_{b, h}}(1^k)$, where $\mathcal{O}_{b, h}$ responds to a query by
sampling $r \in \{0, 1\}^{l(k)}$ and sampling $h \in G_k$ and returning $h(r)$ if $b = 0$ or h if $b = 1$
3: Output 1 iff $b = b'$

Fig. 9: THE EXPERIMENTS $\text{IND}_{\mathcal{A}}$ AND $\text{INV}_{\mathcal{A}\{G\}}$ USED TO DEFINE SECURITY FOR WEAKLY SIMULATABLE SECURITY OF A GROUP FAMILY $\{G\}$.

Dent showed that groups in which the DDH assumptions are believed to hold are simulatable.

Lemma 4. [8] For an infinite sequence of pairs of primes q and p , where $p = 2q + 1$, let $G_{(p,q)}$ be the subgroup of $\mathbb{Z}_p^*$ of order q , then $\{G_{(p,q)}\}$ is simulatable group family.

Theorem 3. The DEG encryption scheme is weakly simulatable if it is instantiated with a simulatable group family $\{G_k\}$ on which the DDH problem is hard.

Proof. Let (h, h^{-1}) be the efficiently computable functions that exist by the fact that the $\{G_k\}$ is a simulatable group family. For ease of notation in this proof, we assume all functions get the required public parameters (e.g. the public-key) as part of their input. We need to give the two functions (f, f^{-1}) for DEG required by the definition of weakly simulatable. Define $f(x = (x_1, x_2, x_3)) = (h(x_1), h(x_2), h(x_3))$, and $f^{-1}(c = (c_1, c_2, c_3)) =$

$(h^{-1}(c_1), h^{-1}(c_2), h^{-1}(c_3))$. From the properties in the definition of a simulatable group (family), (f, f^{-1}) satisfies the corresponding requirements given in the definition of a weakly simulatable encryption scheme.

We now need to argue the final property of a weakly simulatable encryption scheme: no ppt adversary can distinguish between a valid ciphertext and one sampled via f (which may not be a valid ciphertext). Namely, we need to show $\Pr[\text{DIST}_{\text{DEG}}(k, (f, f^{-1}), \mathcal{A}) = 1] \approx 1/2$

To meet this goal, we design a series of games (or experiments) to show the advantage of an adversary being able to distinguish between legitimate ciphertexts, and sampled outputs of f is negligible. Let $W_{i,k}$ be the random variable output of the security experiment in Game i with security parameter k .

Let **Game 1** be exactly the experiment $\text{DIST}_{\text{DEG}}(k, (f, f^{-1}), \mathcal{A})$.

Let **Game 2** be a modification of Game 1, in which the ciphertext c produced on Line 3 of DIST returns an encryption of 1, independent of the value of m .

Claim 5. $\{W_{1,k}\}_k \approx_c \{W_{2,k}\}_k$.

Proof. Follows from CPA security of the DEG encryption scheme. $\square$

Let **Game 3** be a modification of **Game 2** in which again in Line 3 of DIST, the value W computed in Line 2 of the DEG encryption algorithm is computed as follows: $W \leftarrow g^{r'}$, where $r' \in \mathbb{Z}_q$ (instead of $W \leftarrow X_1^y$).

Claim 6. $\{W_{2,k}\} \approx_c \{W_{3,k}\}$

Proof. (sketch) If there is any distinguisher D that can distinguish $\{W_{2,k}\}$ from $\{W_{3,k}\}$ with reasonable probability, then one can use D be used to build a DDH distinguisher D' . In particular, D' when given either a tuple $(g, g^{x_1}, g^y, g^{x_1 y})$ or $(g, g^{x_1}, g^y, g^{r'})$, can choose a random x_2 , simulate a pk for the DEG scheme, and use x_2 and the provided information to compute an appropriate encryption for a perfect simulation of the either **Game 2** or **Game 3**. We can then simulate D , and use the result to break DDH. $\square$

Let **Game 4** be a modification of **Game 3** where the value U in Line 2 of the DEG encryption algorithm is computed as $U = g^{r''}$ for randomly selected $r'' \in \mathbb{Z}_q$.

Claim 7. $\{W_{3,k}\} \approx_c \{W_{4,k}\}$

Proof. (sketch) The proof of this claim parallels that of the previous. If there is a distinguisher D of $\{W_{2,k}\}$ and $\{W_{3,k}\}$ then it can be used to build a DDH distinguisher D' . In particular, D' given a tuple $(g, g^{x_2}, g^y, g^{x_2y})$ or $(g, g^{x_2}, g^y, g^{r'})$, it chooses a random x_1 , simulates a pk for the DEG scheme, and uses x_1 and the provided information to provide an appropriate encryption, for a perfect simulation of the either **Game 3** or **Game 4**. $\square$

In **Game 4** each of the components of the ciphertext in Line 3 of DIST is now a random group element. In **Game 5** we replace these random group elements with random output from the group element sampling algorithm h , which due to the simulatable group properties are indistinguishable from random group elements. Specifically, in **Game 5** in Line 5 of DIST we return the “ciphertext” $(Y = h(r_1), W = h(r_2), U = h(r_3))$, for randomly chosen $r_1, r_2, r_3 \in \{0, 1\}^{l(k)}$.

Claim 8. $\Pr\{W_{4,k}\} \approx_c \{W_{5,k}\}$

Proof. Follows immediately from simulatable group properties of G and h . $\square$

We note that by the definition of f , the output of the encryption algorithm in **Game 5** Line 5 of DIST is an identically, but independently distributed random variable as the one output on Line 4 of DIST (i.e., $f(r_1, r_2, r_3)$ for randomly chosen r_1, r_2, r_3). It is clear that $\Pr[W_{k,5} = 1] = 1/2$.

Therefore, since we can combine all the claims to show that $\Pr[W_{k,1} = 1] \approx_c \Pr[W_{k,5} = 1] = 1/2$ we conclude that DEG is weakly simulatable. $\square$

Theorem 4. *The Cramer-Shoup lite encryption scheme is weakly simulatable if it is instantiated on a simulatable group G on which the DDH problem is hard.*

Proof. Similar to the proof of Theorem 3. $\square$

B Instantiating SPA secure schemes

Definition 6 (DHK_ℓ Assumption (modification of [1])). *Let G be a prime-order-group generator. Let Crt_G be an algorithm that has access to an oracle, takes an ℓ sequence of two primes and two group elements, and returns nothing. Let Ext_G be an algorithm that takes a pair of group elements and some state information, and returns an exponent and a new state. We call Crt_G a DHK_ℓ -adversary and Ext_G a DHK_ℓ -extractor.*


```

 $\text{DHK}_\ell(G, \text{Crt}_G, \text{Ext}_G, k)$ 
1:  $(p_i, q_i, g_i \leftarrow G(1^k); a_i \leftarrow \mathbb{Z}_{q_i}; A_i \leftarrow g_i^{a_i} \bmod p_i)_{i \in \ell(k)}$ 
2: Let  $R[\text{Crt}_G]$  and  $R[\text{Ext}_G]$  be randomly selected strings for  $\text{Crt}_G$  and  $\text{Ext}_G$ 
3:  $st \leftarrow ((p_i, q_i, g_i, A_i)_{i \in [\ell(k)]}, R[\text{Crt}_G])$ 
4: while Simulate  $\text{Crt}_G((p_i, q_i, g_i, A_i)_{i \in [\ell(k)]}; R[\text{Crt}_G])$  do
5:   if  $\text{Crt}_G$  queries  $(i, B, W)$  then
6:      $(b, st) \leftarrow \text{Ext}_G((i, B, W), st; R[\text{Ext}_G])$ 
7:     if  $W \equiv B^{a_i} \bmod p_i$  and  $B \not\equiv g_i^b \bmod p_i$  then Return 1
8:     else Return  $b$  to  $\text{Crt}_G$ 
9: Return 0

```

Fig. 10: DHK_ℓ : AN EXTENSION TO THE DHK DEFINITION

We say that G satisfies the DHK_ℓ assumption if for every polynomial-time Crt_G there exists a polynomial-time Ext_G and negligible function μ , s.t. for all $k \in \mathbb{N}$: $\Pr[\text{DHK}_\ell(G, \text{Crt}_G, \text{Ext}_G, k) = 1] \leq \mu(k)$.

Our modification to DHK_ℓ versus the definition in [1] requires that the ciphertext creator be able to generate ciphertexts relative to a polynomial number of randomly chosen public-keys. It seems reasonable to conjecture that any extractor that could extract exponents with respect to single value $A = g^a$, could do so efficiently for many A_i . We now argue that DEG is sPA1_ℓ secure under the DHK_ℓ assumption.

Theorem 5. *For any polynomial ℓ , if the DHK_ℓ assumption holds and the DEG scheme is instantiated with a simulatable group family $\{G_k\}$, then the DEG scheme is sPA1_ℓ secure.*

Proof. We need to show that for any adversary Crt there exists an extractor Ext that can decrypt its queries flawlessly. Ext receives $(pk_i = \langle p_i, q_i, g_i, A_i, \hat{A}_i \rangle)_{i \in \ell(k)}$ and $R[\text{Crt}]$ as state information. Then Ext builds the DHK_ℓ adversary Crt_G that runs the sPA1_ℓ adversary Crt internally and simulates the sPA1_ℓ experiment for it. Crt_G receives $(p_i, q_i, g_i, A_i)_{i \in \ell(k)}$ and its random coins from Ext and parses its random coins as $(f_G^{-1}(\hat{A}_i))_{i \in [\ell(k)]} \parallel R[\text{Crt}]$ (prepared by Ext where $\hat{A}_i$ is a random group element in G). Notice that since G , the group from which $\hat{A}_i$ is sampled from, is simulatable, it follows that $f_G^{-1}(\hat{A}_i)$ is indistinguishable from random bits and should have indistinguishable effect on the output of the extraction. Because Crt_G is a DHK_ℓ adversary, therefore there exists an extractor for it Ext_G . For

each $i \in [\ell(k)]$, Crt_G sets $pk_i \leftarrow (p_i, q_i, g_i, A_i, \hat{A}_i)$. Crt_G then runs Crt on $(pk_i)_{i \in [\ell(k)]}$ and the random coins $R[\text{Crt}]$ until Crt halts, answering Crt 's queries as follows: upon receiving the query $C = (i, Y, W, U)$ from Crt , Crt_G submits (i, Y, W) to the DHK_ℓ extractor Ext_G . The DHK_ℓ extractor Ext_G returns the value b . If $Y \not\equiv g_i^b \pmod{p_i}$ or $W \not\equiv A_i^b \pmod{p_i}$ then Crt_G returns $\perp$ as the answer to this query, otherwise Crt_G computes $M \leftarrow U \cdot (\hat{A}_i^b)^{-1} \pmod{p_i}$ and return the result to Crt . Notice that since Crt_G is a DHK_ℓ adversary, the extractor Ext_G should return the right answer to the queries Crt_G submits. Since Ext makes a mistake in answering Crt 's queries only when there is a mistake in Ext_G 's answers to Crt_G 's queries, we conclude that Ext also returns the right decryption to the queries submitted by Crt and is an extractor for it. $\square$

Theorem 6. *For any polynomial ℓ , The CS-Lite scheme is sPA1_ℓ secure if the followings hold: i) the DHK_ℓ assumption, and ii) CS-Lite is instantiated with a simulatable group family $\{G_k\}$.*

Proof. Similar to the proof of Theorem 5. $\square$

Constant-Round Concurrent Zero Knowledge From P-Certificates

Kai-Min Chung*

Huijia Lin[†]

Rafael Pass[‡]

Abstract

We present a constant-round concurrent zero-knowledge protocol for **NP**. Our protocol relies on the existence of families of collision-resistant hash functions, and a new, but in our eyes, natural complexity-theoretic assumption: the existence of **P**-certificates—that is, “succinct” non-interactive proofs/arguments for **P**. As far as we know, our results yield the first constant-round concurrent zero-knowledge protocol for **NP** with an explicit zero-knowledge simulator based on *any* assumption.

*Cornell University, Email: chung@cs.cornell.edu

[†]MIT and Boston University, Email: huijia@csail.mit.edu.

[‡]Cornell University, Email: rafael@cs.cornell.edu. Pass is supported in part by a Alfred P. Sloan Fellowship, Microsoft New Faculty Fellowship, NSF CAREER Award CCF-0746990, AFOSR YIP Award FA9550-10-1-0093, and DARPA and AFRL under contract FA8750-11-2-0211. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the Defense Advanced Research Projects Agency or the US government.

1 Introduction

Zero-knowledge ($\mathcal{ZK}$) interactive proofs [GMR89] are paradoxical constructs that allow one player (called the Prover) to convince another player (called the Verifier) of the validity of a mathematical statement $x \in L$, while providing *zero additional knowledge* to the Verifier. Beyond being fascinating in their own right, $\mathcal{ZK}$ proofs have numerous cryptographic applications and are one of the most fundamental cryptographic building blocks.

The notion of concurrent zero knowledge, first introduced and achieved in the paper by Dwork, Naor and Sahai [DNS04], considers the execution of zero-knowledge proofs in an asynchronous and concurrent setting. More precisely, we consider a single adversary mounting a coordinated attack by acting as a verifier in many concurrent executions (called sessions). Concurrent $\mathcal{ZK}$ proofs are significantly harder to construct and analyze. Since the original protocol by Dwork, Naor and Sahai (which relied on so called “timing assumptions”), various other concurrent $\mathcal{ZK}$ protocols have been obtained based on different set-up assumptions (e.g., [DS98, Dam00, CGGM00, Gol02, PTV12, GJO⁺12]), or in alternative models (e.g., super-polynomial-time simulation [Pas03b, PV10]).

In the standard model, without set-up assumptions (the focus of our work,) Canetti, Kilian, Petrank and Rosen [CKPR01] (building on earlier works by [KPR98, Ros00]) show that concurrent $\mathcal{ZK}$ proofs for non-trivial languages, with “black-box” simulators, require at least $\tilde{\Omega}(\log n)$ number of communication rounds. Richardson and Kilian [RK99] constructed the first concurrent $\mathcal{ZK}$ argument in the standard model without any extra set-up assumptions. Their protocol, which uses a black-box simulator, requires $O(n^\epsilon)$ number of rounds. The round-complexity was later improved in the work of Kilian and Petrank (KP) [KP01] to $\tilde{O}(\log^2 n)$ round. Somewhat surprisingly, the simulator strategy of KP is “oblivious”—the “rewinding schedule” of the simulator ignores how the malicious verifier schedules its messages. The key insight behind this oblivious simulation technique is that a single “rewinding” may be helpful for simulating multiple sessions; in essence, KP performs an amortized analysis, which improves the round-complexity. (As we shall see shortly, such an amortized analysis will play an important role also in this work.) More recent work by Prabhakaran, Rosen and Sahai [PRS02] improves the analysis of the KP simulator, achieving an essentially optimal, w.r.t. black-box simulation, round-complexity of $\tilde{O}(\log n)$; see also [PTV12] for an (arguably) simplified and generalized analysis.

The central open problem in the area is whether a *constant-round* concurrent $\mathcal{ZK}$ protocol (for a non-trivial language) can be obtained. A major breakthrough towards resolving this question came with the work of Barak [Bar01], demonstrating a new non-black-box simulation technique that seemed amenable for constructing constant-round protocols that are resilient to concurrent attacks. Indeed, Barak demonstrated a constant-round *bounded-concurrent* argument for **NP** based on the existence of collision-resistant hash-functions; bounded-concurrency here means that for every *a-priori* polynomial bound m on the number of concurrent executions, there exists a protocol (which depends on m) that remains zero-knowledge as long as the number of concurrent execution does not exceed m . (In particular, in the protocol of Barak, the message length of the protocol grows linearly with the *a-priori* bound m on the number of concurrent executions.)

But a decade later, the question of whether “full” (i.e., unbounded) concurrent zero-knowledge is achievable in a constant number of rounds is still wide open. Note that it could very well be the case that all “classic” zero-knowledge protocols already are concurrent zero-knowledge; thus, simply assuming that those protocols are concurrent zero-knowledge yields an assumption under which constant-round concurrent zero-knowledge (trivially) exists—in essence, we are assuming that for every attacker a simulator exists. Furthermore, as we discuss in Section 1.4, if we make strong “concurrent extractability” assumptions of the knowledge-of-exponent type [Dam91, HT98, BP04b],

concurrent zero-knowledge is easy to construct.¹ But such extractability assumptions also simply assume that for every attacker, a simulator (“the extractor”) exists. In essence, rather than basing constant-round concurrent zero-knowledge on a hardness assumption, it is based on a “knowledge” assumption; that is, an assumption that is very similar in flavour to simply assuming that a protocol is zero-knowledge. The central question that we address in this paper is thus the following:

*Can constant-round concurrent zero-knowledge be based on any (reasonable) complexity-theoretic **hardness assumption**?*

As an additional point, even under the above-mentioned strong “knowledge” assumptions, an explicit construction of the concurrent zero-knowledge simulator is not known—it is simply assumed that one exists. For some applications of zero-knowledge such as *deniability* (see e.g., [DNS04, Pas03b]), having an explicit simulator is crucial. As far as we know, there are currently no assumptions (no matter how crazy) under which constant-round concurrent zero-knowledge with an explicit simulator is known.

In fact, even in the common reference string (CRS) model, there are no known constructions of constant-round concurrent zero-knowledge where the simulator does not “program” the CRS; such zero-knowledge protocols were referred to as *deniable zero-knowledge* in the CRS model in [Pas03b].² Indeed, as shown in [Pas03b], the black-box lower-bounds for concurrent zero-knowledge in the plain model extend also to such a “non-programmable” CRS model.

1.1 Our Results

In this work, we present new complexity-theoretic assumptions, which in our eyes are both natural and reasonable (and can be efficiently falsified), under which constant-round concurrent zero-knowledge is achievable. Furthermore, we provide an explicit zero-knowledge simulator.

P-certificates We consider an analogue of Micali’s non-interactive CS-proofs [Mic00] for languages in $\mathbf{P}$. Roughly speaking, we say that (P, V) is a **P-certificate system** if (P, V) is a non-interactive proof system (i.e., the prover send a single message to the verifier, who either accepts or rejects) allowing an efficient prover to convince the verifier of the validity of any *deterministic polynomial-time computation* $M(x) = y$ using a “certificate” of some fixed polynomial length (independent of the size and the running-time of M) whose validity the verifier can check in some fixed polynomial time (independent of the running-time of M). That is, a **P-certificate** allows every deterministic polynomial-time computation to be “succinctly” certified using a “short” certificate (of *a-priori* bounded polynomial length) that can be “quickly” verified (in *a-priori* bounded polynomial-time).

We may consider the existence of **P-certificates** either in the “plain” model (without any set-up), or with some set-up, such as the CRS model. We may also consider various different notions of soundness: *uniform computational soundness*—which states that no *uniform* polynomial-time algorithm can output an accepting certificate for any false statement, *non-uniform computational soundness*—where the same condition holds also w.r.t. non-uniform polynomial-time attackers, and *statistical soundness*—where soundness condition holds also with respect to unbounded attackers restricted to selecting statements of polynomial length.

¹Furthermore, as shown in the recent independent work of [GS12], even a “non-concurrent” (but quite strong in a different way) extractability-type assumption can be used.

²Again, if the simulator gets to program the CRS, such a simulator cannot be used to get deniability.

Note that in the plain model, non-uniform soundness and statistical soundness are equivalent, since if an accepting proof of a false statement exists, a non-uniform efficient attacker can simply get it as non-uniform advice. In the CRS model, however, the notions are (seemingly) distinct.

For our application we will require a slightly stronger soundness condition: soundness needs to hold even against $T(\cdot)$ -time attackers attempting to prove the validity also of $T(\cdot)$ -time computations, where $T(\cdot)$ is some “nice” (slightly) super-polynomial function (e.g., $T(n) = n^{\log \log \log n}$). We refer to such proof systems as *strong $\mathbf{P}$ -certificates*.

On the Existence of $\mathbf{P}$ -certificates In the plain model, a candidate construction of uniformly computationally-sound $\mathbf{P}$ -certificate systems come from Micali’s CS-proofs [Mic00]. These constructions provide short certificates even for all of $\mathbf{NEXP}$. However, since we here restrict to certificates only for $\mathbf{P}$, the assumption that these constructions are sound (resp. strongly sound) $\mathbf{P}$ -certificates is *falsifiable* [Pop63, Nao03]: Roughly speaking, we can efficiently test if an attacker outputs a valid proof of an incorrect statement, since whether a statement is correct or not can be checked in deterministic polynomial time.³

In our eyes, on a qualitatively level, the assumption that Micali’s CS-proofs yield strong $\mathbf{P}$ -certificates is not very different from the assumption that e.g., the Full Domain Hash [BR93] or Schnorr [Sch91] signature schemes are existentially unforgeable: 1) whether an attacker succeeds can be efficiently checked, 2) no attacks are currently known, and 3) the “design-principles” underlying the construction rely on similar intuitions.

As a final point, recall that Micali’s CS-proofs rely on the Fiat-Shamir heuristic, which in general may result in insecure schemes [Bar01, GK03]; however, note that whereas Micali’s construction is *unconditionally* secure in the random oracle model, the counterexamples of [Bar01, GK03] extensively rely on the underlying protocol only being computationally secure; as such, at this time, we have no reason to believe that the Fiat-Shamir heuristic does not work for Micali’s protocol (or any other protocol that is unconditionally secure in the random oracle model).

In the CRS model, we may additionally assume that Micali’s CS-proofs satisfy *non-uniform* computational soundness. Additionally, several recent works provide constructions of “SNARGs” (succinct non-interactive arguments) for $\mathbf{NP}$ in the CRS model [Gro10, Lip12, BCCT13, GGPR13]; such constructions are trivially $\mathbf{P}$ -certificates with non-uniform computational soundness in the CRS model. However, since we restrict to languages in $\mathbf{P}$, checking whether soundness of any of these constructions is broken now becomes efficiently checkable (and thus assuming that they are secure becomes falsifiable).

Finally, let us remark that even statistically-sound $\mathbf{P}$ -certificates may exist: Note that the existence of statistically-sound strong $\mathbf{P}$ -certificates is implied by the assumption that 1) $\mathbf{DTIME}(n^{\omega(1)}) \subseteq \mathbf{NP}$ and 2) $\mathbf{NP}$ proofs for statements in $\mathbf{DTIME}(t)$ can be found in time polynomial in t [BLV06]. In essence, these assumptions says that non-determinism can slightly speed-up computation, and that the non-deterministic choices can be found efficiently, where efficiency is measured in terms of the original deterministic computation. Although we have no real intuition for whether this assumption is true or false,⁴ it seems beyond current techniques to contradict it. (As far as we know, at this point, there is no substantial evidence that even $\mathbf{SUBEXP} \not\subseteq \mathbf{NP}$.)

From $\mathbf{P}$ -certificates to $\mathbf{O}(1)$ -round Concurrent $\mathcal{ZK}$ Our main theorem is the following.

³In contrast, as shown by Gentry and Wichs [GW11], (under reasonable complexity theoretic assumptions) non-interactive CS-proofs for $\mathbf{NP}$ cannot be based on any falsifiable assumption using a black-box proof of security.

⁴As far as we know, the only evidence against it is that it contradicts very strong forms of derandomization assumptions [BLV06, BOV07].

Theorem. *Assume the existence of families of collision-resistant hash-functions secure against polynomial-size circuits, and the existence of a strong $\mathbf{P}$ -certificate system with uniform (resp. non-uniform) soundness. Then there exists a constant-round concurrent zero-knowledge argument for $\mathbf{NP}$ with uniform (resp. non-uniform) soundness. Furthermore, the protocol is public-coin and its communication complexity depends only on the security parameter (but not on the length of the statement proved).*

Let us briefly remark that from a theoretical point of view, we find the notion of uniform soundness of interactive arguments as well-motivated as the one of non-uniform soundness; see e.g., [Gol93] for further discussion. From a practical point of view (and following Rogaway [Rog06])⁵, an asymptotic treatment of soundness is not needed for our results, even in the uniform setting: our soundness proof is a constructive black-box reduction that (assuming the existence of families of collision-resistant hash-functions), transforms any attacker that breaks soundness of our concurrent $\mathcal{ZK}$ protocol on a *single* security parameter 1^n into an attacker that breaks the soundness of the $\mathbf{P}$ -certificate systems with comparable probability on the *same* security parameter 1^n , with only a “small” polynomial overhead. In particular, if some attacker manages to break the soundness of a particular instantiation of our protocol using e.g., Micali’s CS-proof for languages in $\mathbf{P}$ implemented using some specific hash function (e.g., SHA-256), then this attacker can be used to break this *particular* implementation of CS-proofs.

Furthermore, by the above argument, we may also instantiate our protocol with $\mathbf{P}$ -certificates in the CRS model, leading to a constant-round concurrent zero-knowledge protocol (with non-uniform soundness) in the non-programmable CRS model.

Beyond Concurrent $\mathcal{ZK}$ Since the work of Barak [Bar01], non-black-box simulation techniques have been used in several other contexts (e.g., [BGGL01, DGS09, BP12, Lin03, PR03a, Pas04a, BS05, GJ10]). We believe that our techniques will be applicable also in those scenarios. In particular, in Section 1.3, we show that our protocols directly yield a constant-round simultaneously-resettable $\mathcal{ZK}$ [BGGL01, DGS09] for $\mathbf{NP}$, and discuss applications to concurrent secure computation.

1.2 Outline of Our Techniques

We here provide a detailed outline of our techniques. The starting point of our construction is Barak’s [Bar01] non-black-box zero-knowledge argument for $\mathbf{NP}$. Let us start by very briefly recalling the ideas behind his protocol (following a slight variant of this protocol due to [PR03b]). Roughly speaking, on common input 1^n and $x \in \{0, 1\}^{\text{poly}(n)}$, the Prover P and Verifier V , proceed in two stages. In Stage 1, P starts by sending a computationally-binding commitment $c \in \{0, 1\}^n$ to 0^n ; V next sends a “challenge” $r \in \{0, 1\}^{2n}$. In Stage 2, P shows (using a witness indistinguishable argument of knowledge) that either x is true, or there exists a “short” string $\sigma \in \{0, 1\}^n$ such that c is a commitment to a program M such that $M(\sigma) = r$.⁶

Soundness follows from the fact that even if a malicious prover P^* tries to commit to some program M (instead of committing to 0^n), with high probability, the string r sent by V will be different from $M(\sigma)$ for every string $\sigma \in \{0, 1\}^n$. To prove $\mathcal{ZK}$, consider the non-black-box

⁵Rogaway used this argument to formalize what it means for a concrete hash function (as opposed to a family of hash functions) to be collision resistant.

⁶We require that C is a commitment scheme allowing the committer to commit to an arbitrarily long string $m \in \{0, 1\}^*$. Any commitment scheme for fixed-length messages can easily be modified to handle arbitrarily long messages by asking the committer to first hash down m using a collision-resistant hash function h chosen by the receiver, and next commit to $h(m)$.

simulator S that commits to the code of the malicious verifier V^* ; note that by definition it thus holds that $M(c) = r$, and the simulator can use $\sigma = c$ as a “fake” witness in the final proof. To formalize this approach, the witness indistinguishable argument in Stage 2 must actually be a witness indistinguishable *universal argument* (WIUA) [Mic00, BG08] since the statement that c is a commitment to a program M of *arbitrary* polynomial-size, and that $M(c) = r$ within some *arbitrary* polynomial time, is not in NP.

Now, let us consider concurrent composition. That is, we need to simulate the view of a verifier that starts $m = \text{poly}(n)$ concurrent executions of the protocol. The above simulator no longer works in this setting: the problem is that the verifier’s code is now a function of *all* the prover messages sent in different executions. (Note that if we increase the length of r we can handle a bounded number of concurrent executions, by simply letting σ include all these messages).

So, if the simulator could commit not only to the code of V^* , but also to a program M that generates all other prover messages, then we would seemingly be done. And at first sight, this doesn’t seem impossible: since the simulator S is actually the one generating all the prover messages, why don’t we just let M be an appropriate combination of S and V^* ? This idea can indeed be implemented [PR03b, PRT11], but there is a serious issue: if the verifier “nests” its concurrent executions, the running-time of the simulation quickly blows up exponentially—for instance, if we have three nested sessions, to simulate session 3 the simulator needs to generate a WIUA regarding the computation needed to generate a WIUA for session 2 which in turn is regarding the generation of the WIUA of session 1 (so even if there is just a constant overhead in generating a WIUA, we can handle at most $\log n$ nested sessions).

P-certificates to The Rescue Our principal idea is to use **P**-certificates to overcome the above-mentioned blow-up in the running time. On a very high-level, the idea is that once the simulator S has generated a **P**-certificate π to certify some partial computation performed by S in a particular session i , then the same certificate may be reused (without any additional “cost”) to certify the same computation also in other sessions $i' \neq i$. In essence, by reusing the same **P**-certificates, we can amortize the cost of generating them and may then generate WIUA’s about WIUA’s etc., without blowing-up the running time of the simulator. Let us briefly mention how the two salient features of **P**-certificates, namely “non-interactivity” and “succinctness”, are used: Without non-interactivity, the same certificate cannot be reused in multiple sessions, and without succinctness, we do not gain anything by reusing a proof, since just reading the proof may be more expensive than verifying the statement from “scratch”.

Implementing the above high-level idea, however, is quite non-trivial. Below, we outline our actual implementation. We proceed in three steps:

1. We first present a protocol that only achieves bounded-concurrent $\mathcal{ZK}$, using **P**-certificates,
2. We next show how this bounded-concurrent protocol can be slightly modified to become a (fully) concurrent $\mathcal{ZK}$ protocol assuming the existence of so-called *unique P-certificates*—**P**-certificates having the property that for every true statement, there exists a *single* accepting certificate.
3. In the final step, we show how to eliminate the need for uniqueness, by generating **P**-certificates about the generation of **P**-certificates etc., in a tree-like fashion.

Step 1: Bounded Concurrency Using P-certificates In this first step, we present a (somewhat convoluted) protocol using strong **P**-certificates that achieves $m(\cdot)$ -bounded concurrency (using an even more convoluted simulation). As mentioned, Barak’s original protocol could already

be modified to handle bounded concurrency, without the use of **P**-certificates; but as we shall see shortly, our protocol can later be modified to handle full concurrency.

The protocol proceeds just as Barak's protocol in Stage 1 except that the verifier now sends a string $r \in \{0, 1\}^{2m(n)n^2}$ (instead of length $2n$). Stage 2 is modified as follows: instead of having P prove (using a WIUA) that either x is true, or there exists a "short" string $\sigma \in \{0, 1\}^{m(n)n^2}$ such that c is a commitment to a program M such that $M(\sigma) = r$, we now ask P to use a WIUA to prove that either x is true, or

- **commitment consistency:** c is a commitment to a program M_1 , and
- **input certification:** there exists a "short" string $\sigma \in \{0, 1\}^{m(n)n}$, and
- **prediction correctness:** there exists a **P**-certificate π of length n demonstrating that $M_1(\sigma) = r$.

(Note that the only reason we still need to use a *universal* argument is that there is no *a-priori* upper-bound on the length of the program M_1 ; the use of the **P**-certificate takes care of the fact that there is no *a-priori* upper-bound on the running-time of M_1 , though.) Soundness follows using essentially the same approach as above, except that we now also rely on the strong soundness of the **P**-certificate; since there is no *a-priori* upper-bound on neither the length nor the running-time of M_1 , we need to put a cap on both using a (slightly) super-polynomial function, and thus to guarantee soundness of the concurrent zero-knowledge protocol, we need the **P**-certificate to satisfy *strong* soundness.

Let us turn to (bounded-concurrent) zero-knowledge. Roughly speaking, our simulator will attempt to commit to its own code in a way that prevents a blow-up in the running-time. Recall that the main reason that we had a blow-up in the running-time of the simulator was that the generation of the WIUA is expensive. Observe that in the new protocol, the only expensive part of the generation of the WIUA is the generation of the **P**-certificates π ; the rest of the computation has *a-priori* bounded complexity (depending only on the size and running-time of V^*). To take advantage of this observation, we thus have the simulator only commit to a program that generates prover messages (in identically the same way as the actual simulator), but getting certificates $\tilde{\pi}$ as input.

In more detail, to describe the actual simulator S , let us first describe two "helper" simulators S_1, S_2 . S_1 is an interactive machine that simulates prover messages in a "right" interaction with V^* . Additionally, S_1 is expecting some "external" messages on the "left"—looking forward, these "left" messages will later be certificates provided by S_2 . See Figure 1 for an illustration of the communication patterns between S_1, S_2 and V^* .

S_1 proceeds as follows in the right interaction. In Stage 1 of every session i , S_1 first commits to a machine $\tilde{S}_1(j', \tau)$ that emulates an interaction between S_1 and V^* , feeding S_1 input τ as messages on the left, and finally $\tilde{S}_1$ outputs the verifier message in the j' 'th communication round in the right interaction with V^* . (Formalizing what it means for S_1 to commit to $\tilde{S}_1$ is not entirely trivial since the definition of $\tilde{S}_1$ depends on S_1 ; we refer the reader to the formal proof for a description of how this circularity is broken.⁷ S_1 next simulates Stage 2 by checking if it has received a message (j, π_j) in the left interaction, where j is the communication round (in the right interaction with V^*) where the verifier sends its random challenge and expects to receive the first message of Stage 2; if so, it uses $M_1 = \tilde{S}_1$ (and the randomness it used to commit to it), j and σ being the list of messages received by S_1 in the left interaction, as a "fake" witness to complete Stage 2.

⁷Roughly speaking, we let S_1 take the description of a machine M as input, and we then run S_1 on input $M = S_1$.

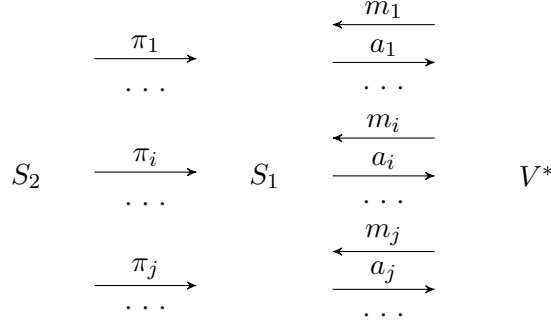


Figure 1: Simulation using **P**-certificates.

The job of S_2 is to provide **P**-certificates π_j for S_1 allowing S_1 to complete its simulation. S_2 emulates the interaction between S_1 and V^* , and additionally, at each communication round j , S_2 feeds S_1 a message (j, π_j) where π_j is a **P**-certificate showing that $\tilde{S}_1(j, \sigma_{<j}) = r_j$, where $\sigma_{<j}$ is the list of message already generated by S_2 , and r_j is the verifier message in the j 'th communication round. Finally, S_2 outputs its view of the full interaction.

The actual simulator S just runs S_2 and recovers from the view of S_2 the view of V^* and outputs it. Note that since S_1 has polynomial running-time, generating each certificate about $\tilde{S}_1$ (which is just about an interaction between S_1 and V^*) also takes polynomial time. As such S_2 can also be implemented in polynomial time and thus also S . Additionally, note that if there are $m(n)$ sessions, the length of σ is at most $O(m(n)n) \ll m(n)n^2$ —for each of the $m(n)$ sessions, and for each round of the constant number of rounds in each session, we need to store a pair (j, π) where π is of length n ; therefore, the simulation always succeeds without getting “stuck”.

Finally, indistinguishability of this simulation, roughly speaking, should follow from the hiding property of the commitment in Stage 1, and the WI property of the WIUA in Stage 2. Or does it? Note that since S_1 is committing to its own code (including its randomness), it is committing to a message that depends on the randomness used for the commitment. (In the language of [BCPT12], this constitutes a randomness-dependent message (RDM) attack on the commitment scheme.) This circularity can be easily overcome (as in [PRT11]) by simply not committing to the randomness of $\tilde{S}_1$, and instead providing it as an additional input to $\tilde{S}_1$ that may be incorporated in σ ; without loss of generality, we may assume that the randomness is “short” since S_1 can always use a PRG to expand it. But the same circularity arises also in the WIUA, and here σ , which contains the seed used to generate the randomness of S_1 , needs to be an input. To overcome it, we here require S_1 to use a *forward-secure* PRG [BY03] to expand its randomness; roughly speaking, a forward-secure PRG ensures that “earlier” chunks of the output of the PRG are indistinguishable from random, even if a seed generating the “later” ones is revealed. We next have S_1 use a new chunk of the output of the PRG to generate each new message in the interaction, but uses these chunk in *reverse order* (i.e., in step 1, the last chunk of the output of the PRG is used, etc.); this means that we can give proofs about “earlier” computations of S_1 (which requires knowing a seeds expanding the randomness used in the computation) while still guaranteeing indistinguishability of “later” messages.⁸

⁸Although the language of forward-security was not used, it was noticed in [PR03b] that GGM’s pseudo-random function [GGM86] could be used to remove circularity in situations as above. A related trick is used in the contemporary work of [CLP12].

Step 2: Full Concurrency using Unique P-certificates The reason that the above approach only yields a bounded concurrent zero-knowledge protocol is that for each new session i , we require S_2 to provide S_1 with new certificates, which thus grows the length of σ . If we could somehow guarantee that these certificates are *determined* by the statement proved in the WIUA, then soundness would hold even if σ is long. Let's first sketch how to do this when assuming the existence of *unique* strong **P**-certificates—that is, **P**-certificates having the property that for each true statement x , there exists a single proof π that is accepted. (We are not aware of any candidates for unique **P**-certificates, but using them serves as a simpler warm-up for our actual protocol.) We simply modify the input certification and prediction correction conditions in the WIUA to be the following:

- **input certification:** there exists a vector $\lambda = ((1, \pi_1), (2, \pi_2), \dots)$ and a vector of messages $\vec{m}$ such that π_i certifies that $M_1(\lambda_{<j})$ output m_j in its j 'th communication round, where $\lambda_{<j} = ((1, \pi_1), \dots, (j-1, \pi_{j-1}))$, and
- **prediction correctness:** there exists a **P**-certificate π of length n demonstrating that $M_1(\lambda) = r$.

Soundness of the modified protocol, roughly speaking, follows since by the unique certificate property, for every program M_1 it inductively follows that for every j , m_j is uniquely defined, and thus also the unique (accepting) certificate π_j certifying $M_1(\lambda_{<j}) = m_j$; it follows that M_1 determines a unique vector λ that passes the input certification conditions, and thus there exists a single r that make M_1 also pass the prediction correctness conditions. Zero-knowledge, on the other hand, can be shown in exactly the same way as above (using S_1, S_2), but we can now handle also unbounded concurrency (since there is no longer a restriction on the length of the input λ).

Step 3: Full Concurrency Using (Plain) P-certificates Let us finally see how to implement the above idea while using “plain” (i.e., non-unique) **P**-certificates. The above protocol is no longer sound since we cannot guarantee that the proofs π_j are unique, and thus the messages m_j may not be unique either, which may make it possible for an attacker to pass the “prediction correctness” condition (without knowing the code of V^*) and thus break soundness. A natural idea would thus be to ask the prover to commit to a machine M_2 (which in the simulation will be a variant of S_2) that produces the certificates π_j , and then require the prover to provide a “second-level” certificate that the “first-level” certificates were generated (deterministically) by running M_2 . But have we really gained anything? Now, to perform the simulation, we need to provide the second-level certificates as input to both M_1 and M_2 ; however, for these second-level certificates, we have no guarantees that they were deterministically generated and again there is no *a-prior* upper bound on the number of such certificates, so it seems we haven't really gained anything.

Our main observation is that a *single* “second-level” certificate can be used to certify the (deterministic generation) of n “first-level” certificates. And a sequence of n “second-level” certificates can be certified by a single “third-level” certificate, etc. At each level, there will be less than n certificates that are *not* certified by a higher-level certificate; we refer to these as “dangling” certificates. See Figure 2 for an illustration of the tree structure, and certified and dangling certificates.

Note that since the number of messages in the interaction with V^* is polynomially bounded, we only have a polynomial-number of level-1 certificates, and thus, the above higher-level certification process does not go beyond a constant number of levels (at each level we need a factor of n less certificates). Finally, note that the total number of “dangling” (uncertified) certificates is bounded by the number of levels times n (and is thus bounded by, say, n^2 .) This means that all the dangling

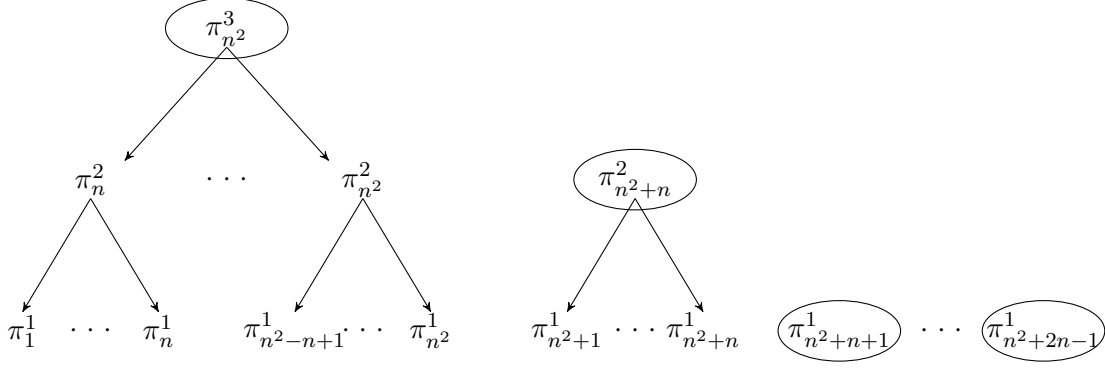


Figure 2: An illustration of the tree structure for generating **P**-certificates. Nodes that are not circled are “certified” certificates; nodes that are circled are “dangling” certificates.

certificates may be provided as a “short” input σ to the committed program, and all the certified certificates can be provided as a “long” (but certified deterministically generated) input λ .

Let us explain this idea more closely using only second-level certificates; this still only gives us bounded-concurrency, but we may now handle $O(m(n)n)$ sessions (instead of just $m(n)$). (More generally, if we use k -levels of certification, we can handle $m(n)n^k$ sessions.) We now change Stage 2 of the protocol to require P to use a WIUA to prove that either x is true, or

- **commitment consistency:** c is a commitment to programs M_1, M_2 , and
- **input certification:** there exists
 - a vector of “certified level-1 certificates” $\lambda^1 = ((1, \pi_1), (2, \pi_2), \dots, (an, \pi_{an}))$,
 - a “small” number of “dangling level-1 certificates” $\sigma^1 = (\sigma_1^1, \sigma_2^1, \dots, \sigma_{j'}^1)$, where $j' < n$ and for each $j \leq j'$, $\sigma_j^1 \in \{0, 1\}^n$,
 - $a \leq m(n)$ level-2 certificates $\sigma^2 = (\sigma_n^2, \sigma_{2n}^2, \dots, \sigma_{an}^2)$ where for each $j \leq a$, $\sigma_{jn}^2 \in \{0, 1\}^n$,

such that,

- σ_{an}^2 certifies that $M_2(\sigma_{<an}^2)$ generates the certificates λ^1 ,

and

- **prediction correctness:** there exists a **P**-certificate π of length n demonstrating that $M_1(\lambda^1, \sigma^1, \sigma^2) = r$.

Soundness of this protocol follows since the total length of “arbitrary” (not deterministic) input is bounded by $(m(n)+n)n \ll m(n)n^2$. $m(n)n$ -bounded concurrent zero-knowledge on the other hand, roughly speaking, follows by letting M_1 be as above (i.e., $\tilde{S}_1$) and M_2 be a variant of the simulator S_2 that outputs all the certificates generated by S_2 . We then define a simulator S_3 responsible for generating second-level certificates for S_2 , and finally outputs its full view of the interaction. The final simulator S runs S_3 and outputs the view of V^* in the interaction. See Figure 3 for an illustration of the communication patterns of S_1, S_2, S_3 and V^* .

Note that as long as there are less than $m(n)n$ message in the interaction with V^* , the number of first-level certificates is bounded by $m(n)n$, and thus we have enough “spots” for second-level certificates (in σ^2) to perform the simulation.

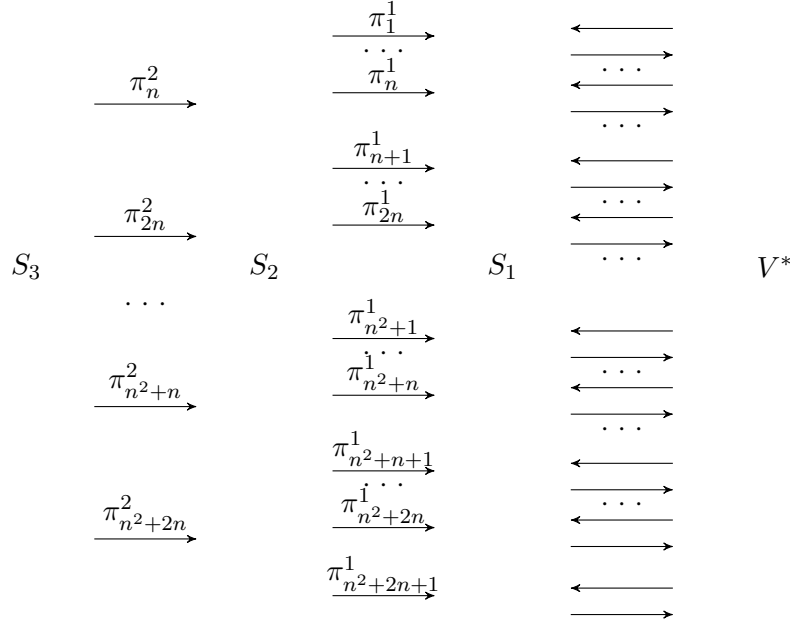


Figure 3: Simulation using second-level $\mathbf{P}$ -certificates.

In the final protocol, we instead have the simulator commit to a sequence $M_1, M_2, \dots$ of machine; roughly speaking, M_1 will be as above, M_2 is responsible for generating first-level certificates (while receiving level $k > 1$ certificates externally), M_3 will be responsible for generating second-level certificates (while receiving level $k > 2$ certificates externally), etc. Note that although there is a (potentially) exponential blow-up in the time needed to generate higher-level certificates, since we only have a constant-number of levels, simulation can be performed in polynomial-time.

1.3 Applications

Our techniques are useful also beyond concurrent $\mathcal{ZK}$. For instance, by applying the transformation of [BGGL01] to our protocol, we directly get a constant-round resettably-sound concurrent $\mathcal{ZK}$. By additionally applying the transformation of [DGS09] to the resulting resettably-sound concurrent $\mathcal{ZK}$ protocol, we get a constant-round *simultaneously-resettably* $\mathcal{ZK}$ protocol.

For another application, a recent result by Goyal on concurrent secure computation [Goy12], demonstrates classes of two-party functionalities that can be securely computed in a concurrent “single-input” setting—that is, we consider a “client-server” setting, where the (honest) server is using the same input in all concurrent sessions. By using our simulation techniques, we can get a crisp condition on the class of functions that can be securely computed in this setting (that significantly expands beyond the class considered in [Goy12]) : any function f for which there exists an efficient procedure M that on input an arbitrary polynomial sequence $(x_1, f(x_1, y)), (x_2, f(x_2, y)), \dots (x_m, f(x_m, y))$ can output a circuit of a-priori bounded (independent of m) size C and an input y' of a-priori bounded length such that for every $i \in [m]$, $f(x_i, y) = C(x_i, y')$. Note that if there exists an efficient procedure for finding the input y (i.e., inverting the function $f'(x_1, x_2, \dots x_m, y) = f(x_1, y) \dots f(x_m, y)$), then this condition is trivially satisfied by simply setting $C = f, y = y'$. (We remark that this condition is very related to the “bounded-entropy” conjecture of [Goy12].) This result is obtained by simply plugging-in our concurrent $\mathcal{ZK}$ protocol into the bounded-concurrent secure two-party computation protocol of

[Pas04b] and noticing that once “straight-line” concurrent $\mathcal{ZK}$ simulation is achieved (as it is in our protocol), the only obstacle for fully concurrent simulation is the need to “compress” outputs from trusted party computing f ; the above condition stipulates that such a compression is always possible. (We expand on these result in the final version of the paper.)

1.4 Related Work

We provide a detailed discussion of some other related works:

- As mentioned in the introduction, constant-round concurrent zero-knowledge protocols with *super-polynomial-time* simulators have been constructed in the plain model [Pas03a, PV08]. For the protocol of [Pas03a], the only super-polynomial-time “advantages” needed by the simulator is to find a pre-image $x' = f^{-1}(y)$ to any point y output by the malicious verifier V^* , as long as y actually is in the range of some one-way function f . If we *assume* that the only way for V^* to output some y in the range of f is by applying f to an input x *that it explicitly knows*, then the protocol of [Pas03a] is concurrent zero-knowledge. A problem with formalizing this is that V^* may already get some string $y = f(x)$ as its auxiliary input and thus may not know x . As in the literature on “knowledge-of-exponent”-type *extractability assumptions* (see e.g., [Dam91, HT98, BP04b, CD09, BCCT12, DFH12, GLR11]), this issue can be resolved by having the prover select the one-way function f from a family $\mathcal{F}$ of one-way functions. Now the extractability assumption we need is that for every polynomial-time oracle machine M , there exists some polynomial-time machine M' such that given any $z \in \{0, 1\}^*$, and uniformly selected functions $\vec{f} = f_1, \dots, f_{\text{poly}(n)} \in \mathcal{F}$, $M^{O(\vec{f})}(1^n, z, \vec{f})$ and $M'(1^n, z, \vec{f})$ generate the same output, where $O(\vec{f})$ is an oracle that inverts the functions in $\vec{f}$. In other words, we are assuming that in the simulation, the simulator together with the verifier can—in polynomial-time—emulate the one-way function inverter used in [Pas03a]. Note that the above extractability assumption is stronger than the typical “knowledge-of-exponent”-type extractability assumptions since we require simultaneous “concurrent” extractability of many images y that are chosen adaptively by the adversary.⁹ However, as shown in [Pas03b], any sufficiently length-expanding random oracle satisfies exactly such an concurrent extractability assumption—this was used in [Pas03a] to construct a concurrent $\mathcal{ZK}$ protocol in the “non-programmable” random oracle model.

A very recent work by Gupta and Sahai [GS12] independent of the current work present an alternative “non-concurrent” extractability assumption under which constant-round concurrent zero-knowledge can be achieved.

One important difference between the above approaches and our work is that we here provide an *explicit* concurrent $\mathcal{ZK}$ simulator. The above-mentioned approaches simply *assume* that such a simulator exists; and, even if the assumption is true, it is not clear, how to find it. In particular, for the purpose of *deniability* (see e.g., [DNS04, Pas03b]) it is not clear whether the approach based on “extractability” assumptions provides sufficient guarantees (unless an explicit simulator strategy is found).

- Barak, Lindell and Vadhan [BLV06] show that under the assumptions that 1) $\mathbf{DTIME}(n^{\omega(1)}) \subseteq \mathbf{NP}$ and 2) $\mathbf{NP}$ proofs for statements in $\mathbf{DTIME}(t)$ can be found in time polynomial in t , 2-round proof exists that are zero-knowledge for uniform verifiers that do not receive any

⁹On the other hand, it is weaker that most other usages of extractability in it requires less structure from the function (i.e., only one-wayness).

auxiliary input. Their zero-knowledge simulator is non-black-box. As mentioned in the introduction, the above-mentioned assumptions imply the existence of statistical strong **P**-certificates. We note that the protocol of [BLV06] is not known to be concurrent (or even sequential) zero-knowledge, even with respect to uniform malicious verifiers.

- Contemporary work by Canetti, Lin and Paneth [CLP12] constructs a *public-coin* concurrent zero-knowledge protocol using non-black-box simulation techniques¹⁰. As shown by Pass, Tseng and Wikstrom [PTW11], public-coin concurrent (and in fact even parallel) zero-knowledge protocols require non-black-box simulation, no matter what the round-complexity is. The protocol of [CLP12] is in the “non-programmable” CRS model of [Pas03a] but as showed in [Pas03a] black-box separation of the Goldreich-Krawczyk [GK96] type (and, in particular, the [PTW11] one, falls into this category) extend also to zero-knowledge in the non-programmable CRS model; thus non-black-box simulation is necessary also for their result. In contrast to our protocol, theirs, however, requires $O(\log^{1+\varepsilon} n)$ number of rounds for arbitrarily small constant ε , but instead only relies on the existence of families of collision-resistant hash functions. (Additionally, [CLP12] note that if assuming the existence of a single hash function that is collision-resistant against uniform adversaries, their protocol can be instantiated also in the plain model with uniform soundness.)

On a technical level, both our work and theirs provide methods for overcoming the exponential blow-up in the simulation time when dealing with non-black-box simulations, but the actual details of the methods are very different: [CLP12] increases the round-complexity to tackle this blow-up, and relies on ideas from the literature on concurrent zero-knowledge with *black-box simulation* [RK99, KP01, PRS02]; as a result, their techniques only apply in the context of super-logarithmic round protocols. In contrast, we rely on **P**-certificates to overcome the blow-up and obtain a constant-round protocol.

- A recent work by Bitansky, Canetti, Chiesa, Tromer [BCCT13] present techniques for composing SNARKs (succinct non-interactive arguments of knowledge) for **NP**; roughly speaking, [BCCT13] shows that if for *some* sufficiently large c , any *non-deterministic* n^c computation can be proved using an “argument of knowledge” of length n that can be verified in time n^2 , then for *any* d , every non-deterministic n^d -time computation can be also be proved (using a SNARK of length n that can be verified in time n^2). This is achieved by having the prover first generate a SNARK for each subcomputation of n^c steps, and then for each “chunk” of n SNARKs, having the prover prove that it *knows* SNARKs that are accepted for all these subcomputations, and so on in a tree-like fashion. Finally, the prover only provides the verifier with a “top-level” SNARK that it knows lower-level SNARKs that proves that it knows even lower-level SNARKs etc. This type of proof composition was previously also used by Valiant [Val08]. To prove that this type of composition works it is crucial to work with languages in **NP** (since we are proving statements about the *existence* of some SNARKs); additionally, it is crucial that we are dealing with arguments *of knowledge*—SNARKs of false statements may exists, so to guarantee soundness, the prover needs to show that not only appropriate SNARKs exists, but also that it “knows” them.

At a superficial level, our simulator strategy also uses a tree of “proofs”. However, rather than proving knowledge of lower-level “proofs” etc, in our approach, higher-level **P**-certificates are only used to demonstrate that lower-level **P**-certificates have been deterministically generated. As a consequence, we do not need to certify non-deterministic computations; additionally, we

¹⁰Our results and theirs were developed in parallel

do not need the certificates to satisfy an argument of knowledge property. Indeed, this is what allows us to base **P**-certificates on a falsifiable assumption.

1.5 Acknowledgements

We are very grateful to Ran Canetti, Johan Håstad Omer Paneth, and Alon Rosen for many discussions about concurrent zero-knowledge and non-black-box simulation. We are especially grateful to both Alon Rosen and Omer Paneth for very insightful discussions about how to formalize non-black-box simulations that “commit to their own code”; additionally, as we mention in the paper, several obstacles to using non-black-box simulation to obtain constant-round concurrent zero-knowledge were noticed in an unpublished manuscript with Alon dating back to 2003 [PR03a]. Thanks a lot!

2 Preliminaries

Let $\mathcal{N}$ denote the set of positive integers, and $[n]$ denote the set $\{1, 2, \dots, n\}$. We denote by PPT probabilistic polynomial time Turing machines. We assume familiarity with interactive Turing machines, denoted ITM, interactive protocols. Given a pair of ITMs, A and B , we denote by $(A(x), B(y))(z)$ the random variable representing the (local) output of B , on common input z and private input y , when interacting with A with private input x , when the random tape of each machine is uniformly and independently chosen, and $\text{View}_B \langle A(x), B(y) \rangle (z)$ the random variable representing B 's view in such an interaction. The term **negligible** is used for denoting functions that are (asymptotically) smaller than one over any polynomial. More precisely, a function $\nu(\cdot)$ from non-negative integers to reals is called **negligible** if for every constant $c > 0$ and all sufficiently large n , it holds that $\nu(n) < n^{-c}$.

2.1 Witness Relations

We recall the definition of a witness relation for a **NP** language [Gol01].

Definition 1 (Witness relation). A *witness relation* for a language $L \in \mathbf{NP}$ is a binary relation R_L that is polynomially bounded, polynomial time recognizable and characterizes L by $L = \{x : \exists y \text{ s.t. } (x, y) \in R_L\}$

We say that y is a witness for the membership $x \in L$ if $(x, y) \in R_L$. We will also let $R_L(x)$ denote the set of witnesses for the membership $x \in L$, i.e., $R_L(x) = \{y : (x, y) \in R_L\}$. In the following, we assume a fixed witness relation R_L for each language $L \in \mathbf{NP}$.

2.2 Computational Indistinguishability

The following definition of computational indistinguishability originates in the seminal paper of Goldwasser and Micali [GM84]. Let X be a countable set of strings. A *probability ensemble indexed by X* is a sequence of random variables indexed by X . Namely, any element of $A = \{A_x\}_{x \in X}$ is a random variable indexed by X .

Definition 2 (Indistinguishability). Let X be a countable set. Two ensembles $\{A_{n,x}\}_{n \in \mathcal{N}, x \in X}$ and $\{B_{n,x}\}_{n \in \mathcal{N}, x \in X}$ are said to be *computationally indistinguishable over N* if for every probabilistic

machine D (the distinguisher) whose running time is polynomial in its first input, there exists a negligible function $\nu(\cdot)$ so that for every $n \in N$ and $x \in X$:

$$|\Pr[a \leftarrow A_{n,x} : D(1^n, x, a) = 1] - \Pr[b \leftarrow B_{n,x} : D(1^n, x, b) = 1]| < \nu(n)$$

2.3 Interactive Proofs and Arguments

We recall the standard definitions of interactive proofs [GMR89] and arguments (a.k.a computationally sound proofs) [BCC88]. In our definition of arguments, we distinguish between uniform soundness, where soundness only needs to hold against a uniform probabilistic polynomial-time algorithms, and non-uniform soundness, where it holds against non-uniform polynomial-time algorithms. Typically, in the literature on zero-knowledge argument, non-uniform soundness is more commonly used (but there are exceptions, see e.g., [BP04a]). We find the uniform model of computation as well-motivated as the non-uniform one; see e.g., [Gol93].

Definition 3 (Interactive Proof System). A pair of interactive machines (P, V) is called an *interactive proof system* for a language L if there is a negligible function $\nu(\cdot)$ such that the following two conditions hold:

- *Completeness*: For every $n \in N$, $x \in L$, and every $w \in R_L(x)$,

$$\Pr[(P(w), V)(1^n, x) = 1] = 1$$

- *Soundness*: For every pair of machines B_1, B_2 and every $n \in N$,

$$\Pr[(x, z) \leftarrow B_1(1^n) : x \notin L \wedge (B_2(z), V)(1^n, x) = 1] \leq \nu(n)$$

If the soundness condition only holds against all polynomial-time (resp. non-uniform polynomial-time) machines B_1, B_2 , the pair (P, V) is called a *uniformly-sound* (resp. *non-uniformly sound*) *interactive argument system*.

2.4 Witness Indistinguishability

An interactive protocol is *witness indistinguishable* (WI) [FS90] if the verifier's view is “independent” of the witness used by the prover for proving the statement.

Definition 4 (Witness-indistinguishability). An interactive protocol (P, V) for $L \in \mathbf{NP}$ is *witness indistinguishable* for R_L if for every PPT adversarial verifier V^* , and for every two sequences $\{w_{n,x}^1\}_{n \in N, x \in L \cap \{0,1\}^{\text{poly}(n)}}$ and $\{w_{n,x}^2\}_{n \in N, x \in L \cap \{0,1\}^{\text{poly}(n)}}$, such that $w_{n,x}^1, w_{n,x}^2 \in R_L(x)$ for every $n \in N$ and $x \in L \cap \{0,1\}^{\text{poly}(n)}$, the following ensembles are computationally indistinguishable over N :

- $\{\text{View}_{V^*} \langle P(w_{n,x}^1), V^*(z) \rangle (1^n, x) \}_{n \in N, x \in L \cap \{0,1\}^{\text{poly}(n)}, z \in \{0,1\}^*}$
- $\{\text{View}_{V^*} \langle P(w_{n,x}^2), V^*(z) \rangle (1^n, x) \}_{n \in N, x \in L \cap \{0,1\}^{\text{poly}(n)}, z \in \{0,1\}^*}$

2.5 Commitment Schemes

Commitment protocols allow a *sender* to commit itself to a value while keeping it secret from the *receiver*; this property is called *hiding*. At a later time, the commitment can only be opened to a single value as determined during the commitment protocol; this property is called *binding*. Commitment schemes come in two different flavors, statistically binding and statistically hiding; we only make use of statistically binding commitments in this paper. Below we sketch the properties of a statistically binding commitment; full definitions can be found in [Gol01].

In statistically binding commitments, the binding property holds against unbounded adversaries, while the hiding property only holds against computationally bounded (non-uniform) adversaries. The statistical-binding property asserts that, with overwhelming probability over the randomness of the receiver, the transcript of the interaction fully determines the value committed to by the sender. The computational-hiding property guarantees that the commitments to any two different values are computationally indistinguishable.

Non-interactive statistically-binding commitment schemes can be constructed using any one-to-one one-way function (see Section 4.4.1 of [Gol01]). Allowing some minimal interaction (in which the receiver first sends a single random initialization message), statistically-binding commitment schemes can be obtained from any one-way function [Nao91, HILL99].

2.6 Universal Arguments

Universal arguments (introduced in [BG08] and closely related to the notion of CS-proofs [Mic00]) are used in order to provide “efficient” proofs to statements of the universal language $L_{\mathcal{U}}$ with witness relation $\mathbf{R}_{\mathcal{U}}$ defined in [BG08, Mic00]. A triplet $y = (M, x, t) \in L_{\mathcal{U}}$ if the non-deterministic machine M accepts input X within $t < T(|x|)$ steps, for a slightly super-polynomial function $T(n) = n^{\log \log n}$. We denote by $T_M(x, w)$ the running time of M on input x using the witness w . Notice that every language in $\mathbf{NP}$ is linear time reducible to $L_{\mathcal{U}}$. Thus, a proof system for $L_{\mathcal{U}}$ allows us to handle all $\mathbf{NP}$ -statements. Below we recall the definition in [BG08].

Definition 5 (Universal argument). A pair of interactive Turing machines (P, V) is called a *universal argument* system if it satisfies the following properties:

- *Efficient verification*: There exists a polynomial p such that for any $y = (M, x, t)$, the total time spent by the (probabilistic) verifier strategy V , on common input 1^n , y , is at most $p(n + |y|)$. In particular, all messages exchanged in the protocol have length smaller than $p(n + |y|)$.
- *Completeness by a relatively efficient prover*: For every $n \in N$, $y = (M, x, t) \in L_{\mathcal{U}}$ and w in $\mathbf{R}_{\mathcal{U}}(y)$,

$$\Pr[(P(w), V)(1^n, (M, x, t)) = 1] = 1$$

Furthermore, there exists a polynomial q such that the total time spent by $P(w)$, on common inputs 1^n and (M, x, t) , is at most $q(n + |y| + T_M(x, w)) \leq q(n + |y| + t)$.

- *Computational Soundness*: For every polynomial size circuit family $\{P_n^*\}_{n \in N}$, there is a negligible function ν , such that, for every $n \in N$ and every triplet $(M, x, t) \in \{0, 1\}^{\text{poly}(n)} \setminus L_{\mathcal{U}}$,

$$\Pr[(P_n^*, V)(1^n, (M, x, t)) = 1] < \nu(n)$$

- *Weak proof of knowledge:* For every positive polynomial p there exists a positive polynomial p' and a probabilistic polynomial-time oracle machine E such that the following holds: for every polynomial-size circuit family $\{P_n^*\}_{n \in N}$, every sufficiently large $n \in N$ and every $y = (M, x, t) \in \{0, 1\}^{\text{poly}(n)}$ if $\Pr[(P_n^*, V)(1^n, y) = 1] > 1/p(n)$ then

$$\Pr_r[\exists w = w_1, \dots, w_t \in \mathbf{R}_{\mathcal{U}}(y) \text{ s.t. } \forall i \in [t], E_r^{P_n^*}(1^n, y, i) = w_i] > \frac{1}{p'(n)}$$

where $\mathbf{R}_{\mathcal{U}}(y) \stackrel{\text{def}}{=} \{w : (y, w) \in \mathbf{R}_{\mathcal{U}}\}$ and $E_r^{P_n^*}(\cdot, \cdot, \cdot)$ denotes the function defined by fixing the random-tape of E to equal r , and providing the resulting E_r with oracle access to P_n^* .

The weak proof-of-knowledge property of universal arguments only guarantees that each individual bit w_i of some witness w can be extracted in probabilistic polynomial time. Given an input 1^n and $y = (M, x, t)$ in $L_{\mathcal{U}} \cap \{0, 1\}^{\text{poly}(n)}$, since the witness $w \in \mathbf{R}_{\mathcal{U}}(y)$ is of length at most t , it follows that there exists an extractor running in time polynomial in $\text{poly}(n) \cdot t$ that extracts the whole witness; we refer to this as the *global proof-of-knowledge property* of a universal argument.

The notion of witness indistinguishability of universal argument for $\mathbf{R}_{\mathcal{U}}$ is defined similarly as that for interactive proofs/arguments for $\mathbf{NP}$ relations; we refer the reader to [BG08] for a formal definition. [BG08] (based on [Mic00, Kil95]) presents a witness indistinguishable universal argument based on the existence of families of collision-resistant hash functions.

2.7 Concurrent Zero-Knowledge

An interactive proof is said to be *zero-knowledge* if it yields nothing beyond the validity of the statement being proved [GMR89].

Definition 6 (Zero-knowledge). An interactive protocol (P, V) for language L is *zero-knowledge* if for every PPT adversarial verifier V^* , there exists a PPT simulator S such that the following ensembles are computationally indistinguishable over $n \in N$:

- $\{\text{View}_{V^*} \langle P(w), V^*(z) \rangle (1^n, x)\}_{n \in N, x \in L \cap \{0, 1\}^{\text{poly}(n)}, w \in R_L(x), z \in \{0, 1\}^{\text{poly}(n)}}$
- $\{S(1^n, x, z)\}_{n \in N, x \in L \cap \{0, 1\}^{\text{poly}(n)}, w \in R_L(x), z \in \{0, 1\}^{\text{poly}(n)}}$

In this work we consider the setting of concurrent composition. Given an interactive protocol (P, V) and a polynomial m , an m -session *concurrent adversarial verifier* V^* is a PPT machine that, on common input x and auxiliary input z , interacts with up to $m(|x|)$ independent copies of P concurrently. The different interactions are called *sessions*. There are no restrictions on how V^* schedules the messages among the different sessions, and V^* may choose to abort some sessions but not others. For convenience of notation, we overload the notation $\text{View}_{V^*} \langle P, V^*(z) \rangle (1^n, x)$ to represent the view of the cheating verifier V^* in the above mentioned concurrent execution, where V^* 's auxiliary input is z , both parties are given common input 1^n , $x \in L$, and the honest prover has a valid w witness of x .

Definition 7 (Concurrent Zero-Knowledge [DNS04]). An interactive protocol (P, V) for language L is *concurrent zero-knowledge* if for every concurrent adversarial verifier V^* (i.e., any m -session concurrent adversarial verifier for any polynomial m), there exists a PPT simulator S such that following two ensembles are computationally indistinguishable over $n \in N$.

- $\{\text{View}_{V^*} \langle P(w), V^*(z) \rangle (1^n, x)\}_{n \in N, x \in L \cap \{0, 1\}^{\text{poly}(n)}, w \in R_L(x), z \in \{0, 1\}^{\text{poly}(n)}}$
- $\{S(1^n, x, z)\}_{n \in N, x \in L \cap \{0, 1\}^{\text{poly}(n)}, w \in R_L(x), z \in \{0, 1\}^{\text{poly}(n)}}$

2.8 Forward Secure PRG

Roughly speaking, a *forward-secure pseudorandom generator (PRG)* (first formalized by [BY03], but early usages go back to [BH92]) is a pseudorandom generator where the seed is periodically updated—thus we have a sequence of seeds $s_1, s_2, \dots$ generating a pseudorandom sequence $q_1, q_2, \dots$ —such that if the seed s_t is exposed (and thus the “later” sequence $q_{t+1}, q_{t+2}, \dots$ is also exposed), the “earlier” sequence $q_1, \dots, q_t$ still remains pseudorandom.

We provide a simple definition of a forward secure pseudorandom generator, where the “exposure” time t is statically selected.¹¹

Definition 8 (Forward-secure Pseudorandom Generator). We say that a polynomial-time computable function G is a *forward secure Pseudo-Random Generator (fsPRG)* if on input a string s , and $\ell \in N$, it outputs two sequences $(s_1, s_2, \dots, s_\ell)$ and $(q_1, q_2, \dots, q_\ell)$ such that the following properties hold:

- *Consistency*: For every $n, \ell \in N$, $s \in \{0, 1\}^n$, the following holds
 - if $G(s, \ell) = ((s_1, \vec{s}), (q_1, \vec{q}))$, then $G(s_1, \ell - 1) = (\vec{s}, \vec{q})$.
- *Forward Security*: For every polynomial p , the following ensembles are computationally indistinguishable
 - $\{s \leftarrow U_n, (\vec{s}, \vec{q}) \leftarrow G(s, \ell) : s_t, \vec{q}_{\leq t}\}_{n \in N, \ell \in [p(n)], t \in [\ell]}$
 - $\{s_t \leftarrow U_n, \vec{q} \leftarrow (U_n)^\ell : s_t, \vec{q}_{\leq t}\}_{n \in N, \ell \in [p(n)], t \in [\ell]}$

where U_n is the uniform distribution over $\{0, 1\}^n$ and $\vec{q}_{\leq t} = (q_1, \dots, q_t)$.

Any (traditional) PRG implies the existence of a forward secure PRG; thus by the result of [HILL99] the existence of forward secure PRGs are implied by the existence of one-way functions.

In our application of forward secure PRGs, we will use the outputs of the PRG in *reverse order*, and thus write $G(s, \ell) = (s_\ell, s_{\ell-1}, \dots, s_1), (q_\ell, q_{\ell-1}, \dots, q_1)$. As a consequence, we may reveal a seed s_t “explaining” the “earlier” sequence $((s_{t-1}, \dots, s_1), (q_{t-1}, \dots, q_1))$ while guaranteeing that the “later” sequence $(q_\ell, \dots, q_t)$ still is indistinguishable from random.

3 P-certificates

In this section we define the notion of **P**-certificates. On a high-level, **P**-certificates can be viewed as an analogue of Micali’s CS-proofs [Mic00], but where we restrict to languages in **P**. As we shall see, by restricting to languages in **P**, we can make the soundness condition of (a restricted class of) **P**-certificates falsifiable.

Roughly speaking, we say that (P, V) is a **P**-certificate system if (P, V) is a non-interactive proof system (i.e., the prover send a single message to the verifier, who either accepts or rejects) allowing an efficient prover to convince the verifier of the validity of any *deterministic polynomial-time computation* $M(x) = y$ using a “certificate” of some fixed polynomial length (independent of the size and the running-time of M) whose validity the verifier can check in some fixed polynomial time (independent of the running-time of M); that is, any deterministic polynomial-time computation can be certified using a “short” certificate that can be “quickly” verified.

¹¹The definition of [BY03] allows an attacker to adaptively select the exposure time t . For our purposes the simpler non-adaptive notion suffices.

To formalize this, we consider the following canonical languages for $\mathbf{P}$: for every constant $c \in N$, let $L_c = \{(M, x, y) : M(x) = y \text{ within } |x|^c \text{ steps}\}$. Let $T_M(x)$ denotes the running time of M on input x .

Definition 9. A pair of probabilistic interactive Turing machines, $(P_{\text{cert}}, V_{\text{cert}})$, is a $\mathbf{P}$ -certificate system if there exist polynomials g_P, g_V, ℓ such that the following holds:

- *Efficient Verification:* On input $c \geq 1, 1^k$ and a statement $q = (M, x, y) \in L_c$, and $\pi \in \{0, 1\}^*$, V_{cert} runs in time at most $g_V(k + |q|)$;
- *Completeness by a Relatively Efficient Prover:* For every $c, d \in N$, there exists a negligible function μ such that for every $k \in N$ and every $q = (M, x, y) \in L_c$ such that $|q| \leq k^d$,

$$\Pr[\pi \leftarrow P_{\text{cert}}(c, 1^k, q) : V_{\text{cert}}(c, 1^k, q, \pi) = 1] \geq 1 - \mu(k)$$

Furthermore, P_{cert} on input $(c, 1^k, q)$ outputs a certificate of length $\ell(k)$ in time bounded by $g_P(k + |M| + T_M(x))$.

- *Soundness:* For every $c \in N$, and every PPT P^* , there exists a negligible function μ such that for every $k \in N$,

$$\Pr[(q, \pi) \leftarrow P^*(c, 1^k) : V_{\text{cert}}(c, 1^k, q, \pi) = 1 \wedge q \notin L_c] \leq \mu(k)$$

We also consider a stronger soundness condition stipulating that soundness holds even if the attacker selects a slightly super-polynomial-size statement and specifies some slightly super-polynomial runtime.

- *Strong Soundness:* There exists some “nice” super-polynomial function¹² $T(k) \in k^{\omega(1)}$ and some “nice” super-constant function¹³ $C(\cdot) \in \omega(1)$ such that for every probabilistic algorithm P^* with running-time bounded by $T(\cdot)$, there exists a negligible function μ , such that, for every $k \in N$, $c \leq C(k)$

$$\Pr[(c, q, \pi) \leftarrow P^*(1^k) : V_{\text{cert}}(c, 1^k, q, \pi) = 1 \wedge q \notin L_c] \leq \mu(k)$$

We say that $(P_{\text{cert}}, V_{\text{cert}})$ is a *statistically-sound $\mathbf{P}$ -certificate system* (resp. *statistically sound strong $\mathbf{P}$ -certificate system*) if the soundness condition holds also against (unbounded) P^* with polynomially-bounded (resp. $T(\cdot)$ -bounded) output.

Remark 1. *The reason that we do not consider a notion of computational soundness with respect to non-uniform polynomial-time attackers is that such a notion is equivalent to statistical soundness: if an accepting proof of a false statement exists, a non-uniform efficient attacker can simply get it as non-uniform advice. Nevertheless, it still makes sense to consider a notion of $a(\cdot)$ -bounded-non-uniform soundness, where soundness holds for attacker that on input the security parameter 1^k additionally receive $a(k)$ bits of non-uniform advice. Our results regarding uniform soundness directly extend also to the regime of bounded non-uniform soundness.*

As we shall see shortly, a candidate construction of a (computationally-sound) $\mathbf{P}$ -certificate systems comes from Micali’s CS-proofs [Mic00]. We also note that the assumption that statistically-sound strong $\mathbf{P}$ -certificates exists is implied by the assumption that 1) $DTIME(n^{\omega(1)}) \subseteq NP$ and 2) $\mathbf{NP}$ proofs for statements in $DTIME(t)$ can be found in time polynomial in t [BLV06]. (In essence, the assumption says that non-determinism can slightly speed-up computation, and that the non-deterministic choices can be found efficiently, where efficiency is measured in terms of the original deterministic computation.)

¹²For instance, $T(n) = n^{\log \log \log n}$.

¹³For instance, $C(k) = \log \log \log n$.

3.1 Time-Representation Invariant $\mathbf{P}$ -certificates

At first sight it may seem that since we consider only languages in $\mathbf{P}$, the sound (resp., strongly soundness) condition of $\mathbf{P}$ -certificates is *falsifiable* [Pop63, Nao03]: we should be able to efficiently test if an attacker outputs a valid proof of an incorrect statement, since whether a statement is correct or not can be checked in deterministic polynomial time.

This intuition is somewhat misleading: recall that soundness needs to hold for *all* polynomial-time computations, where the time-bound n^c may be selected by the attacker trying to break soundness. Since there is no *a-priori* constant bound on c , the attacker may make the test (checking whether soundness was broken) run in super-polynomial-time (by selecting a large c .) The situation is even worse for the case of strongly sound $\mathbf{P}$ -certificates.

At first one may think that this issue can be easily resolved by restricting to certificate systems where the prover is asked to provide an upper-bound on the running-time of M in unary; this certainly makes the soundness condition falsifiable, but such certificates are no longer “short”. We overcome this issue by allowing for a more flexible representation of (upper-bound on) the running-time of M , and restrict to *time-representation invariant $\mathbf{P}$ -certificates*—namely proof systems where whether the verifier accepts a proof of a statement x does not depend on how the time-bound is represented. For a time-invariant $\mathbf{P}$ -certificate, it suffices to define soundness in the case that the attacker specifies the running-time bound in unary; by the time-representation invariance condition, this implies soundness also for other (more efficient) representations.

Towards this, we consider an alternative variant of canonical languages in $\mathbf{P}$: for every constant $c \in N$, let $L'_c = \{(M, x, y, 1^n) : M(x) = y \text{ within } n^c \text{ steps}\}$.

Definition 10. A pair of probabilistic interactive Turing machines, $(P_{\text{cert}}, V_{\text{cert}})$, is a *time-representation invariant $\mathbf{P}$ -certificate system* if there exist polynomials g_P, g_V, ℓ such that the following holds:

- *Efficient Verification:* On input $c \geq 1, 1^k$ and a statement $q = (M, x, y, 1^n) \in L'_c$, and $\pi \in \{0, 1\}^*$, V_{cert} runs in at most $g_V(k + |q|)$ time.
- *Time-Representation Invariant Verification:* There exists a negligible function μ such that every $c, \tilde{c}, n, \tilde{n}$, such that $n^c = \tilde{n}^{\tilde{c}}$, every $k \in N$ and every $(M, x, y) \in \{0, 1\}^*$ and every certificate $\pi \in \{0, 1\}^*$,

$$|\Pr[V_{\text{cert}}(c, 1^k, (M, x, y, 1^n), \pi) = 1] - \Pr[V_{\text{cert}}(\tilde{c}, 1^k, (M, x, y, 1^{\tilde{n}}), \pi) = 1]| \leq \mu(k)$$

- *Completeness by a Relatively Efficient Prover:* For every $c, d \in N$, there exists a negligible function μ such that for every $k \in N$ and every $q = (M, x, y, 1^n) \in L'_c$ such that $|q| \leq k^d$,

$$\Pr[\pi \leftarrow P_{\text{cert}}(c, 1^k, q) : V_{\text{cert}}(c, 1^k, q, \pi) = 1] \geq 1 - \mu(k)$$

Furthermore, P_{cert} on input $(c, 1^k, q)$ outputs a certificate of length at most $\ell(k)$ in time bounded by $g_P(k + |M| + n^c)$.

- *Soundness for L'_1 :* For every PPT P^* , there exists a negligible function μ such that for every $k \in N$,

$$\Pr[(q, \pi) \leftarrow P^*(1^k) : V_{\text{cert}}(1, 1^k, q, \pi) = 1 \wedge q \notin L'_1] \leq \mu(k)$$

We say that $(P_{\text{cert}}, V_{\text{cert}})$ is a *strong time-representation invariant $\mathbf{P}$ -certificate system* if there exists some “nice” $T(k) \in k^{\omega(1)}$ such that the soundness for L'_1 condition holds with respect to all probabilistic algorithms with running-time bounded by $T(\cdot)$. We say that $(P_{\text{cert}}, V_{\text{cert}})$ is

a *statistically-sound time-representation invariant $\mathbf{P}$ -certificate system* (resp. *statistically sound strong time-representation invariant $\mathbf{P}$ -certificate system*) if the soundness for L'_1 condition holds also against (unbounded) P^* with polynomially-bounded (resp. $T(\cdot)$ -bounded) output.

Note that the soundness condition of time-representation invariant $\mathbf{P}$ -certificates is clearly falsifiable since checking whether the attacker actually outputs a statement $q \notin L'_1$ can be done in linear-time in the length of the statement, and verification of a certificate π for a statement q can be done in polynomial-time by definition.

Let us briefly outline a candidate construction of time-representation invariant $\mathbf{P}$ -certificates (where both P_{cert} and V_{cert} are deterministic).

A Candidate Construction Based on CS-proofs. Micali's CS proofs [Mic00] are obtained by first constructing a public-coin 4-round interactive argument for **NEXP** (similar to the “succinct” 4-round interactive argument for NP of [Kil95]) and then eliminating interaction through the Fiat-Shamir paradigm [FS90]: that is, the verifier's random message are generated by applying a random oracle to the prover's messages, and next the random oracle may be instantiated with a concrete family of hash function $\{h_k\}_{k \in N}$. More precisely, CS proofs are used to prove membership of the CS language L_{CS} with witness relation R_{CS} as defined in [Mic00]. A quadruple $(M, x, y, t) \in L_{\text{CS}}$ iff the lengths of x and y are smaller than t and $M(x) = y$ in t steps. Roughly speaking, to prove a statement $q = (M, x, y, t)$, the prover, on input a security parameter 1^k , proceeds in two steps. In the first step, it constructs a PCP (Probabilistically Checkable Proof) [BFLS91, FGL⁺91] proof π' for q and computes a Merkle's hash tree [Mer89] with π' as the leaves using a hash function h_k , producing a root r . Then, in the second step, it computes a polylogarithmic number l of PCP queries, determined by the hash value $h_k(r)$; for each PCP query i , it finds the authentication path a_i that reveals the corresponding PCP answer b_i . Finally, the prover sends a CS proof $\pi = t \| r \| b_i \| a_i \| \dots \| b_l \| a_l$. The verifier, on input a statement x and such a proof π , checks whether all the authentication paths are accepting w.r.t. r , recomputes the PCP queries using $h_k(r)$ and checks whether all the PCP answers are accepting.

In our language L'_c , recall that a statement is of form $q = (M, x, y, 1^n)$. The prover and the verifier on input c , 1^k and q can thus recover a time bound t by computing n^c and then recover the corresponding CS language instance (M, x, y, t) , and next simply runs the prover and verifier algorithms of CS-proofs. By construction it follows that the above construction satisfies prover's relative efficiency and completeness. Additionally, since the verification procedure only depends on the time bound $t = n^c$, and not on the values of n and c , the verification procedure also has the time-representation invariance property.

Finally, in our eyes, assuming that the above construction satisfies the soundness condition of time-representation invariant $\mathbf{P}$ -certificates is a reasonable and “well-behaved” complexity theoretic assumption: on a qualitatively level, the assumption is not very different from the assumption that e.g., the Full Domain Hash [BR93] or Schnorr [Sch91] signature schemes are existentially unforgeable: 1) whether an attacker succeeds can be efficiently checked, 2) no attacks are currently known, and 3) the “design-principles” underlying the constructions rely on similar intuitions (e.g., that instantiating random-oracles with hash functions in “natural” schemes lead to secure protocol).

Finally, recall that Micali's CS-proofs rely on the Fiat-Shamir heuristic, which in general may result in insecure schemes [Bar01, GK03]; however, note that whereas Micali's construction is *unconditionally* secure in the random oracle model, the counterexamples of [Bar01, GK03] extensively rely on the underlying protocol only being computationally secure; as such, at this time, we have no reason to believe that the Fiat-Shamir heuristic does not work for Micali's protocol (or any other protocol that is unconditionally secure in the random oracle model).

Time Representation Invariant P-certificates in the CRS model In the CRS model, we may additionally assume that Micali’s CS-proofs satisfy *non-uniform* computational soundness. Additionally, several recent works provide constructions of “SNARGs” (succinct non-interactive arguments) for **NP** in the CRS model [Gro10, Lip12, BCCT13, GGPR13]; such constructions are trivially **P**-certificates with non-uniform computational soundness in the CRS model. It follows using exactly the same argument as above that these new constructions also are time-representation invariant.

From Time-Representation Invariant P-certificates to P-certificates As we now show, time-representation invariant **P**-certificates imply **P**-certificates.

Theorem 1. *Assume the existence of a time-representation invariant **P**-certificate system (resp. a strong time-representation invariant **P**-certificate system) $(P'_{\text{cert}}, V'_{\text{cert}})$. Then, there exists a **P**-certificate system (resp. a strong **P**-certificate system) $(P_{\text{cert}}, V_{\text{cert}})$. Furthermore if $(P'_{\text{cert}}, V'_{\text{cert}})$ is statistically sound (resp. statistically strong sound), then $(P_{\text{cert}}, V_{\text{cert}})$ is so as well.*

Proof. Let $(P'_{\text{cert}}, V'_{\text{cert}})$ be a time-representation invariant **P**-certificate system. Consider a **P**-certificate system $(P_{\text{cert}}, V_{\text{cert}})$ where P_{cert} and V_{cert} simply runs P'_{cert} and V'_{cert} respectively with n fixed to the length of the input x . More precisely, P_{cert} on input $c, 1^k$ and a statement $q = (M, x, y) \in L_c$, lets $q' = (M, x, y, 1^{|x|}) \in L'_c$, runs $P'_{\text{cert}}(c, 1^k, q')$ and outputs whatever P'_{cert} outputs.; V_{cert} on input $(c, 1^k, q, \pi)$ computes q' in exactly the same way, runs $V'_{\text{cert}}(c, 1^k, q', \pi)$ and outputs the verdict of V'_{cert} . It follows from the relative prover efficiency and completeness properties of $(P'_{\text{cert}}, V'_{\text{cert}})$ that $(P_{\text{cert}}, V_{\text{cert}})$ also satisfies relative prover efficiency and completeness. Let us turn to soundness. We only prove the case of strong soundness (assuming that $(P_{\text{cert}}, V_{\text{cert}})$ is strongly sound), all the other cases follow analogously.

Assume for contradiction that for every $T(k) \in k^{\omega(1)}$ and $C(k) \in \omega(1)$, there exists a $T(k)$ -time cheating prover A , and a polynomial p such that for infinitely many $k \in N$ and $c_k \leq C(k)$, it holds that the probability that $A(1^k)$ outputs c_k , a false statement $q = (M, x, y) \notin L_{c_k}$ and a certificate π for $q \in L_{c_k}$ that is accepted by V_{cert} (that is, $V_{\text{cert}}(c_k, 1^k, q, \pi) = 1$) is at least $1/p(k)$. Fix some arbitrary function $T'(k) \in k^{\omega(1)}$. Let $T(k) \in k^{\omega(1)}$ and $C(k) \in \omega(1)$ be two functions such that $T(k)^{C(k)} \leq T'(k)$. By our assumption, there exists a cheating prover A that violates the strong soundness property of $(P_{\text{cert}}, V_{\text{cert}})$ w.r.t. the functions $T(k)$ and $C(k)$ with some polynomial probability $1/p(k)$. Using A , we construct another cheating prover A' that violates the strong soundness for L'_1 of $(P'_{\text{cert}}, V'_{\text{cert}})$ w.r.t. function $T'(k)$ with the same probability $1/p(k)$. Machine A' on input 1^k simply runs $A(1^k)$ to obtain $c_k, q = (M, x, y)$ and π ; it then sets $n = |x|^{c_k}$ and outputs $q' = (M, x, y, 1^n)$ and π . Clearly, A' runs in time $T(k)^{C(k)} \leq T'(k)$. By construction of V_{cert} , the probability that $V_{\text{cert}}(c_k, 1^k, q, \pi) = 1$ is the same as the probability that $V'_{\text{cert}}(c_k, 1^k, \tilde{q}, \pi) = 1$, where $\tilde{q} = (M, x, y, 1^{|x|})$. Furthermore, by the time-representation-invariance of V'_{cert} , the probability that $V'_{\text{cert}}(c_k, 1^k, \tilde{q}, \pi) = 1$ is negligibly close to the probability that $V'_{\text{cert}}(1, 1^k, q', \pi) = 1$. It follows that A' (whose running-time is bounded by $T'(k)$) outputs accepting proofs of false statements with probability negligibly close to $\frac{1}{p(k)}$ for infinitely many $k \in N$. Since the above holds for any function $T'(k)$, we have that $(P'_{\text{cert}}, V'_{\text{cert}})$ is not strongly sound for L'_1 , which is a contradiction. $\square$

4 Constant-round Concurrent $\mathcal{ZK}$

In this section, we present our construction of a constant-round concurrent $\mathcal{ZK}$ protocol. To simplify the exposition (and following the description in the introduction), as a warm-up, we first

present a protocol that only uses one level of **P**-certificates and thus only handles a bounded number, $O(m(n))$, of concurrent executions; we refer to this protocol as “Protocol 1”. We then generalize Protocol 1 and describe a protocol that uses k levels of certificates and can handle $O(n^k)$ concurrent executions; we refer to this protocol as “Protocol k ”. By setting k to be super-constant, say, $k = \log n$, we obtain a (fully) concurrent $\mathcal{ZK}$ protocol.

4.1 Protocol 1

We proceed to describe Protocol 1, (P_1, V_1) , which we prove is m -bounded concurrent zero-knowledge. The protocol relies on the following primitives:

- A commitment scheme **com**: for simplicity of presentation, we assume that **com** is a non-interactive commitment scheme, but the protocol can be modified in a straight-forward way to work for any two-message commitment scheme (as in [Nao91]).
- A strong **P**-certificate system $(P_{\text{cert}}, V_{\text{cert}})$ with parameter $T(\cdot)$ and $C(\cdot)$, where $T(\cdot)$ is a “nice” super-polynomial function and $C(\cdot)$ is a “nice” super-constant function: for, simplicity of exposition, we assume that both P_{cert} and V_{cert} are *deterministic*. We discuss in Section 4.3 how to modify the protocol to also handle randomized **P**-certificate systems.
- A family of hash functions $\{\mathcal{H}_n\}_n$: to simplify the exposition, we here assume that both **com** and $\{\mathcal{H}_n\}_n$ are collision resistant against circuits of size $T'(\cdot)$, where $T'(\cdot)$ is “nice” super-polynomial function. As in [BG02], this assumption can be weakened to just collision resistance against polynomial-size circuits by modifying the protocol to use a “good” error-correcting code ECC (i.e., with constant distance and with polynomial-time encoding and decoding), and replace commitments $\text{com}(h(\cdot))$ with $\text{com}(h(\text{ECC}(\cdot)))$.

Let us now turn to specifying the protocol (P_1, V_1) . The protocol makes use of three parameters: $m(\cdot)$ is a polynomial that upper bounds the number of concurrent sessions; $\Gamma(\cdot)$ is a “nice” super-polynomial function such that $T(n), T'(n) \in \Gamma(n)^{\omega(1)}$, and $D(\cdot)$ is a “nice” super-constant function such that $D(n) \leq C(n)$. Let $m = m(n)$, $\Gamma = \Gamma(n)$ and $D = D(n)$. In the description below, when discussing **P**-certificates, we always consider the language L_D .

The prover P_1 and the verifier V_1 , on common input 1^n and x and private input a witness w to P_1 , proceed as follow:

Phase 1: P_1 and V_1 exchanges the following three messages.

1. V_1 chooses a randomly sampled hash function $h \leftarrow \mathcal{H}_n$.
2. P_1 sends a commitment to 0^n using **com**.
3. V_1 replies with a random “challenge” r of length $3mn$.

Phase 2: P_1 gives a WIUA argument of the statement that either $x \in L$ OR there exists $\tilde{S}_1 \in \{0, 1\}^{\Gamma(n)}$, $j \in [m]$, $s \in \{0, 1\}^n$, $\pi \in \{0, 1\}^n$, $\sigma \in \{0, 1\}^{\Gamma(n)}$, ρ , such that

1. **Commitment Consistency:** $c = \text{com}(h(\tilde{S}_1); \rho)$,
2. **Input Certification:** $|\sigma| \leq 2mn$,
3. **Prediction Correctness:** π certifies that $\tilde{S}_1(1^n, j, s, \sigma) = r$.

A formal description of the protocol can be found in Figure 4 and 5.

PROTOCOL 1

Common Input: A security parameter 1^n in unary and an instance x of a language $L \in \mathbf{NP}$ with witness relation $\mathbf{R}_L$.

Parameters: $m = m(n)$ is an upper bound on the number of concurrent sessions. $\Gamma = \Gamma(n)$ and $D = D(n)$ are respectively upper bounds on the size of the committed program and the time bound.

Phase 1:

$V_1 \rightarrow P_1$: Send $h \leftarrow \mathcal{H}_n$.

$P_1 \rightarrow V_1$: Send $c = \text{com}(0^n; \rho)$.

$V_1 \rightarrow P_1$: Send $r \leftarrow \{0, 1\}^{3mn}$.

Phase 2:

$P_1 \Leftrightarrow V_1$: A WIUA $\langle P_{\text{UA}}, V_{\text{UA}} \rangle$ proving the OR of the following statements:

1. $\exists w \in \{0, 1\}^{\text{poly}(|x|)} \text{ s.t. } \mathbf{R}_L(x, w) = 1.$
2. $\exists \langle \tilde{S}_1, j, s, \pi, \sigma, \rho \rangle \text{ s.t. } \mathbf{R}_S(\langle h, c, r \rangle, \langle \tilde{S}_1, j, s, \pi, \sigma, \rho \rangle) = 1.$

Figure 4: A public-coin non-black-box bounded concurrent zero-knowledge protocol.

Our Simulator. As explained in the introduction, the goal of our simulator is to try to “commit to its own code” in a way that prevents a blow-up in the running-time. Note that in our protocol, the only expensive part of the generation of the WIUA is the generation of the $\mathbf{P}$ -certificates π ; the rest of the computation has *a-priori* bounded complexity (depending only on the size and running-time of V^*). To take advantage of this observation, we thus have the simulator only commit to a program that generates prover messages (in identically the same way as the actual simulator), but getting certificates $\tilde{\pi}$ as input.

In more detail, to describe the actual simulator S , let us first describe two “helper” simulators S_1, S_2 . Roughly speaking, S_1 is an interactive machine that simulates prover messages in a “right” interaction with V^* . Additionally, S_1 is expecting some “external” messages on the “left”; these “left” messages will be certificates provided by S_2 . See Figure 1 in the introduction for an illustration of the communication patterns between S_1, S_2 and V^* .

Let us turn to a formal description of the S_1 and S_2 . To simplify the exposition, we assume w.l.o.g that V^* has its non-uniform advice z hard-coded, and is deterministic (as it can always get its random tape as non-uniform advice).

On a high-level, $S_1(1^n, x, M, s, \ell)$ acts as a prover in a “right” interaction, communicating with a concurrent verifier V^* , while receiving some additional “external” messages on the “left”. (The input x is the statement to be proved, the input M will later be instantiated with the code of S_1 , and the input (s, ℓ) is used to generate the randomness for S_1 ; s is the seed for the forward secure pseudorandom generator g , and ℓ is the number of n -bit long blocks to be generated using g .) A communication round in the “right” interaction with V^* refers to a verifier message (sent by V^*) followed by a prover message (sent by S_1).

Let us now specify how S_1 generates prover messages in its “right” interaction with V^* . $S_1(1^n, x, M, s, \ell)$ acts as follows:

- Upon invocation, S_1 generates its “random-tape” by expanding the seed s ; more specifically,

Instance: A triplet $\langle h, c, r \rangle \in \mathcal{H}_n \times \{0, 1\}^{\text{poly}(n)} \times \{0, 1\}^{3mn}$.
Witness: $\langle \tilde{S}_1, j, s, \pi, \sigma, \rho \rangle$: A program $\tilde{S} \in \{0, 1\}^\Gamma$, an integer $j \in [m]$, a seed $s \in \{0, 1\}^n$, a P -certificate $\pi \in \{0, 1\}^n$, a string $\sigma \in \{0, 1\}^\Gamma$, a randomness $\rho \in \{0, 1\}^n$.
Relation: $\mathbf{R}_S(\langle h, c, r \rangle, \langle \tilde{S}_1, j, s, \pi, \sigma, \rho \rangle) = 1$ if and only if: 1. Commitment Consistency: $c = \text{com}(h(\tilde{S}_1); \rho)$, 2. Input Certification: $\sigma \leq 2mn$, 3. Prediction Correctness: $V_{\text{cert}}(D, 1^n, (\tilde{S}_1, (1^n, j, s, \sigma), r), \pi) = 1$ (i.e., π certifies that $\tilde{S}_1(1^n, j, s, \sigma) = r$).

Figure 5: $\mathbf{R}_S$, a relation that Protocol 1 uses in WIUA of Phase 2.

let $(s_\ell, s_{\ell-1}, \dots, s_1)$, $(q_\ell, q_{\ell-1}, \dots, q_1)$ be the output of $g(s, \ell)$. We assume without loss of generality that S_1 only needs n bits of randomness to generate any prover message (it can always expand these n bits into a longer sequence using a PRG); in order to generate its j 'th prover message, it uses q_j as randomness.

- Upon receiving a hash function h_i in session i during the j -th communication round, S_1 provides a commitment c_i to (the hash of) the program $\tilde{S}_1(1^n, j, s', \tau) = \text{wrap}(M(1^n, x, M, s', j), V^*, \tau, j)$, where $\text{wrap}(A, B, \tau, j)$ is the program that lets A communicate with B for j rounds, while allowing A to receive τ as external messages, and finally outputting B 's message in the j 'th communication round. (That is, $\tilde{S}_1(1^n, j, s', \tau)$ emulates j rounds of an execution between S_1 and V^* where S_1 expands out the seed s' into j blocks of randomness and additionally receives τ as external messages.)
- Upon receiving a challenge r_i in session i during the j 'th communication round, S_1 needs to provide a WIUA. To do so, it checks whether it was received an external message (j, π_j) , and if so, it uses the certificate π_j to complete the WIUA (and otherwise halts). More precisely, S_1 provides an honest WIUA that c_i is a commitment to $\tilde{S}_1$ and that π_j certifies that $\tilde{S}_1(1^n, j, s_j, \tau) = r_i$ where τ is list of external messages received by S_1 so far. (Note that since we only require $\tilde{S}_1$ to generate the j 'th verifier message, giving him the seed (s_j, j) as input suffices to generate all prover messages in rounds $j' < j$. It follows from the consistency requirement of the forward secure PRG that $\tilde{S}_1$ using (s_j, j) as seed will generate the exact same random sequence for the $j - 1$ first blocks as if running $\tilde{S}_1$ using (s, ℓ) as seed.)

$S_2(1^n, x, M, s, \ell)$ internally emulates ℓ messages of an execution between $S_1(1^n, x, M, s, \ell)$ and V^* . In *each* communication round j , after V^* generates a verifier message m_j , S_2 generates a certificate π_j (using P_{cert}) that $\tilde{S}_1(1^n, j, s_j, \sigma) = m_j$, where σ is the list of external messages received by S_1 so far, and feeds (j, π_j) to S_1 . Finally, S_2 outputs its view (which in particular, contains the view of V^*) at the end of the execution.

The final simulator $S(1^n, x)$ simply runs $S_2(1^n, x, S_1, s, T(n + |x|))$, where s is a uniformly random string of length n and $T(n + |x|)$ is a polynomial upper-bound on the number of messages sent by V^* given the common input $1^n, x$, and extracts out, and outputs, the view of V^* from the output of S_2 .

Running-time of S . Let us first argue that S_1 runs in polynomial time. Clearly it only takes S_1 polynomial-time to generate the commitments in Phase 1 (since V^* has a polynomial-length description, and thus also the code of S_1). During the WIUA in Phase 2, the length of the witness used by the simulator is polynomial in length of the description of $\tilde{S}_1$ and the length of the certificate π used by S_1 ; both are of polynomial length. Since the **P**-certificates verification time is polynomial in the length of the statement proved, it follows that the relation being proved in the WIUA has a time complexity that is upper bounded by a fixed polynomial in the length of V^* . By the relative prover efficiency condition of the WIUA, each such proof only requires some fixed polynomial-time, and thus the whole execution of S_1 takes some fixed polynomial time (in the length of V^* and thus also in the length of x .) It directly follows that also $\tilde{S}_1$'s running-time is polynomially bounded.

Finally, since S_2 is simply providing certificates about the execution of $\tilde{S}_1$, it follows by the relative prover efficiency condition of **P**-certificates, that S_2 runs in polynomial time, and thus also S .

Indistinguishability of the simulation Assume that there exists a cheating verifier V^* , a distinguisher D and a polynomial p such that the real view and the simulated view of V^* can be distinguished by D with probability $\frac{1}{p(n)}$ for infinitely many n . More formally, for infinitely many $n \in N$, $x \in L \cap \{0, 1\}^{\text{poly}(n)}$, $w \in \mathbf{R}_L(x)$ and $z \in \{0, 1\}^{\text{poly}(n)}$, it holds that

$$|\Pr[D(\text{View}_{V^*} \langle P(w), V^*(z) \rangle (1^n, x)) = 1] - \Pr[D(S(1^n, x, z)) = 1]| \geq \frac{1}{p(n)}$$

Consider a hybrid experiment $\text{Real}'_{V^*}(n, x, z)$ that proceeds just as the real experiment except that all phase 1 commitments are generated by committing to the code of $\tilde{S}_1$ (as done by S). We also denote by $\text{Real}'_{V^*}(n, x, z)$ the view of the verifier V^* in the hybrid. It follows by a simple hybrid argument that there exists a polynomial p' such that the view of V^* in the hybrid Real' and in simulation by S can be distinguished by D with probability $\frac{1}{p'(n)}$ for infinitely many n . That is, for infinitely many $n \in N$, $x \in L \cap \{0, 1\}^{\text{poly}(n)}$, $w \in \mathbf{R}_L(x)$ and $z \in \{0, 1\}^{\text{poly}(n)}$, it holds that

$$|\Pr[D(\text{Real}'_{V^*}(n, x, w, z)) = 1] - \Pr[D(S(1^n, x, z)) = 1]| \geq \frac{1}{p'(n)} \quad (1)$$

Consider such n, x, z (and assume that z is hard-coded into the description of V^*), and consider $T = T(n + |x|)$ hybrid experiments (recall that $T(n + |x|)$ is the maximum number of communication rounds given common input $1^n, x$). In hybrid H_j , the first j communication rounds are simulated exactly as by S (using pseudo-randomness), but all later communication round $j' > j$ are simulated by S (and more specifically by S_1) using *true* randomness q'_j being uniformly distributed in $\{0, 1\}^n$; additionally, to complete all WIUA that *begin at or after* communication round j , S_1 uses the true witness w instead of the “fake” witness used by S_1 . (Note that once we start using real randomness in some session i , it is no longer clear whether simulation of “later” sessions can be completed. To deal with this issue, we thus also switch all WIUA that begin at or after round j to use a real witness; if some WIUA already began at some communication round $j' < j$, then the simulation of this WIUA can still be completed.)

It follows by Equation 1 and a hybrid argument that there exist some j and a polynomial p'' such that D distinguishes H_j and H_{j+1} with probability $\frac{1}{p''(n)}$. Now, consider another hybrid experiment $\tilde{H}_j$ that proceeds just at H_j , but where true randomness is used in communication round $j + 1$ (but still using the fake witness). It follows by the forward security of the PRG g that the outputs of H_{j+1} and $\tilde{H}_j$ are indistinguishable—the reason we need *forward security* is

that to emulate communication rounds $j' \leq j$, the seeds $s_{j'}$ may need to be known (as they are used by S_1 to provide WIUA's). Indistinguishability of $\tilde{H}_j$ and H_j follows directly by the witness indistinguishability property of the WIUA. It thus leads to a contradiction and completes the proof of the indistinguishability of the simulation.

4.2 Protocol k

We move on to describe our actual concurrent $\mathcal{ZK}$ protocol: Protocol k , (P_k, V_k) . We refer the reader to the introduction for the ideas underlying this protocol.

As with Protocol 1, Protocol k proceeds in two phases. In Phase 1, the prover P_k and the verifier V_k proceeds exactly as in Protocol 1 but the length of the “challenge” r is modified to be $3kn^2$. Next, Phase 2 is modified as follows:

Phase 2: P_k gives a WIUA argument of the statement that either $x \in L$ OR there exists $\tilde{S}_1, \dots, \tilde{S}_k \in \{0, 1\}^{\Gamma(n)}$, $0 < j < n^k$, $s^1, \dots, s^k \in \{0, 1\}^n$, $\pi^1, \dots, \pi^k \in \{0, 1\}^n$, $\sigma^1, \dots, \sigma^k \in \{0, 1\}^{\Gamma(n)}$, $\lambda^1, \dots, \lambda^k \in \{0, 1\}^{\Gamma(n)}$, ρ , such that

1. **Commitment Consistency:** $c = \text{com}(h(\tilde{S}_1, \dots, \tilde{S}_k); \rho)$,
2. **Input Certification:**
 - (a) $|\vec{\sigma}| \leq 2kn^2$; and
 - (b) Let l^* be the largest l such that $j \geq n^{l-1}$. Then $\lambda^{\geq l^*} = \text{null}$ and for $2 \leq l \leq l^*$, π^l certifies that $\tilde{S}_l(1^n, [j]_{n^{l-1}}, s^l, ([\lambda^{\geq l}]_{[j]_{n^{l-1}}}, \sigma^{\geq l})) = \lambda^{l-1}$.
3. **Prediction Correctness:** π^1 certifies that $\tilde{S}_1(1^n, j, s^1, ([\lambda^{\geq 1}]_j, \sigma^{\geq 1})) = r$

where $[j]_x \triangleq j - (j \bmod x)$, and the bracket operator $[\cdot]_j$ is defined as follows: The input is expected to be a set of triples of the form $(j', l', \pi_{j'}^{l'})$, and the output is a subset of these obtained by removing elements with $j' \geq j$; that is, we are “filtering out” all messages that were generated in communication round j or later. Roughly speaking, the bracket operator is used to eliminate “unnecessary” inputs to the program. We require this to be able to reuse **P**-certificates; we provide a more detailed explanation of why this is needed in Remark 2, after having formalized the simulator.

Using the notation from the introduction, the messages $\vec{\lambda}$ are “certified” certificates (each component of $\vec{\lambda}$ may of an unbounded polynomial length), and the messages $\vec{\sigma}$ are “dangling” certificates (each component of $\vec{\sigma}$, however, is “short” by the input certification condition).

A formal description of Protocol k can be found in Figure 6 and 7.

We will be analyzing (P_k, V_k) when $k = \log n$ (but the analysis works as long as k is a “nice” super-constant, but polynomially-bounded, function). It is easy to check that the protocol is complete. Furthermore, since the honest prover P_k , on private input a valid witness w of the statement x , always succeeds in the Phase 2 by proving that $x \in L$, by the prover and verifier efficiency conditions of WIUA, both the honest prover P_k and verifier V_k run in some fixed polynomial time. Furthermore note that the communication complexity of the protocol depends only on the security parameter 1^n but not the length of the statement x ; thus the protocol is “succinct”.

We turn to showing that (P_k, V_k) is sound and concurrent $\mathcal{ZK}$ when $k = \log n$.

PROTOCOL k Common Input: A security parameter 1^n and an instance x of a language $L \in \mathbf{NP}$ with witness relation $\mathbf{R}_L$. Parameters: $m = m(n)$ is an upper bound on the number of concurrent sessions. $\Gamma = \Gamma(n)$ and $D = D(n)$ are respectively upper bounds on the size of the committed program and the time bound. Phase 1: $V_k \rightarrow P_k$: Send $h \leftarrow \mathcal{H}_n$. $P_k \rightarrow V_k$: Send $c = \text{com}(0^n; \rho)$. $V_k \rightarrow P_k$: Send $r \leftarrow \{0, 1\}^{3kn^2}$. Phase 2: $P_k \Leftrightarrow V_k$: A WIUA $\langle P_{\text{UA}}, V_{\text{UA}} \rangle$ proving the OR of the following statements: 1. $\exists w \in \{0, 1\}^{\text{poly}(x)}$ s.t. $\mathbf{R}_L(x, w) = 1$. 2. $\exists \langle \vec{S}, j, \vec{s}, \vec{\pi}, \vec{\sigma}, \vec{\lambda}, \rho \rangle$ s.t. $\mathbf{R}_S(\langle h, c, r \rangle, \langle \vec{S}, j, \vec{s}, \vec{\pi}, \vec{\sigma}, \vec{\lambda}, \rho \rangle) = 1$.

Figure 6: A public-coin non-black-box concurrent zero-knowledge protocol.

4.2.1 Soundness of Protocol k

Lemma 1. *Under the above-mentioned cryptographic assumptions, (P_k, V_k) is uniformly sound. Additionally, if $(P_{\text{cert}}, V_{\text{cert}})$ is a statistically strong $\mathbf{P}$ -certificate system, then (P_k, V_k) is non-uniformly sound.*

Proof. We prove this lemma w.r.t. uniform soundness assuming $(P_{\text{cert}}, V_{\text{cert}})$ is a strong $\mathbf{P}$ -certificate; the non-uniform part of the lemma follows in identically the same way.

Assume for contradiction that there is a probabilistic polynomial time cheating prover P^* and a polynomial p , such that for infinitely many $n \in \mathbb{N}$, with probability $1/p(n)$, P^* selects a false statement $x \in \{0, 1\}^{\text{poly}(n)} \setminus L$ and convinces V_k of the membership of x in L .

Fix one such n . Let $P_{u,h,r}^*$ be the “residual” deterministic WIUA prover resulting from fixing P^* ’s randomness to u and feeding it the messages h and r . Let E be the “global” proof-of-knowledge extractor of the WIUA. Note that E runs in time $\text{poly}(\Gamma(n))$. Let E_s denote E with randomness fixed to s . Now, consider the following experiment Exp:

- Sample a tuple (u, h, r, s) uniformly at random.
- Let $(x, c) \leftarrow P_{u,h,r}^*$ and $w' \leftarrow E_s^{P_{u,h,r}^*}$, where x is the statement selected by $P_{u,h,r}^*$, c is the commitment generated by $P_{u,h,r}^*$, and w' is the witness extracted by $E_s^{P_{u,h,r}^*}$.

Let BAD denote the event that $E_s^{P_{u,h,r}^*}$ extracts a valid “fake” witness $w' = (\vec{S}, j', \vec{s}', \vec{\pi}', \vec{\sigma}', \vec{\lambda}', \rho') \in \mathbf{R}_S(h, c, r)$ in the above experiment.

Let us first argue that by our assumption (that P^* breaks soundness), BAD happens with non-negligible probability: By an averaging argument, with probability at least $1/2p(n)$ over (u, h, r) , the statement x selected by $P_{u,h,r}^*$ is not a member of L and yet $P_{u,h,r}^*$ convinces the WIUA verifier with

Instance: A triplet $\langle h, c, r \rangle \in \mathcal{H}_n \times \{0, 1\}^{\text{poly}(n)} \times \{0, 1\}^{3kn^2}$.

Witness: $\langle \vec{S}, j, \vec{s}, \vec{\pi}, \vec{\sigma}, \vec{\lambda}, \rho \rangle$: A sequence of programs $\vec{S} = (\tilde{S}_1, \dots, \tilde{S}_k) \in \{0, 1\}^{k \cdot \Gamma}$, an integer $j \in [n^k]$, a sequence of seeds $\vec{s} = (s^1, \dots, s^k) \in \{0, 1\}^{k \cdot n}$, a sequence of $\mathbf{P}$ -certificates $\vec{\pi} = (\pi^1, \dots, \pi^k) \in \{0, 1\}^{k \cdot n}$, a sequence $\vec{\sigma} = (\sigma^1, \dots, \sigma^k) \in \{0, 1\}^{k \cdot \Gamma}$, a sequence $\vec{\lambda} = (\lambda_1, \dots, \lambda_k) \in \{0, 1\}^{k \cdot \Gamma}$, a randomness $\rho \in \{0, 1\}^n$.

Relation: $\mathbf{R}_S(\langle h, c, r \rangle, \langle \vec{S}, j, \vec{s}, \vec{\pi}, \vec{\sigma}, \vec{\lambda}, \rho \rangle) = 1$ if and only if:

1. **Commitment Consistency:** $c = \text{com}(h(\vec{S}); \rho)$,
2. **Input Certification:**
 - (a) $|\vec{\sigma}| \leq 2kn^2$, and
 - (b) Let l^* be the largest l such that $j \geq n^{l-1}$. $\lambda^{\geq l^*} = \text{null}$ and for $2 \leq l \leq l^*$, $V_{\text{cert}}(D, 1^n, (\tilde{S}_l, (1^n, [j]_{n^{l-1}}, s^l, ([\lambda^{\geq l}]_{[j]_{n^{l-1}}}, \sigma^{\geq l})), \lambda^{l-1}), \pi^l) = 1$ (i.e., π^l certifies that $\tilde{S}_l(1^n, [j]_{n^{l-1}}, s^l, ([\lambda^{\geq l}]_{[j]_{n^{l-1}}}, \sigma^{\geq l})) = \lambda^{l-1}$).
3. **Prediction Correctness:** $V_{\text{cert}}(D, 1^n, (\tilde{S}_1, (1^n, j, s^1, ([\lambda^{\geq 1}]_j, \sigma^{\geq 1})), r), \pi^1) = 1$ (i.e., π certifies that $\tilde{S}_1(1^n, j, s^1, ([\lambda^{\geq 1}]_j, \sigma^{\geq 1})) = r$).

where $[j]_x \triangleq j - (j \bmod x)$, and the operator $[\cdot]_j$ is defined as follows: The input is expected to be a set of triples of the form $(j', l', \pi'_{j'})$, and the output is a subset of these obtained by removing elements with $j' \geq j$.

Figure 7: $\mathbf{R}_S$, a relation that Protocol k uses in WIUA of Phase 2.

probability $1/2p(n)$. For each such a tuple (u, h, r) , by the “global” proof-of-knowledge property of WIUA, $E_s^{P^*_{u,h,r}}$ extracts a valid “fake” witness $w' \in \mathbf{R}_S(h, c, r)$ with some non-negligible probability $1/q(n)$ (over the randomness s). It follows that BAD happens with non-negligible probability.

We now show that under our cryptographic assumptions, BAD can only happen with negligible probability, which is a contradiction.

First, note that by the soundness of $(P_{\text{cert}}, V_{\text{cert}})$ with parameters $T(\cdot)$ and $C(\cdot)$, and the fact that $T(n) = \Gamma(n)^{\omega(1)}$ and $D(n) \leq C(n)$, we have that except with negligible probability over the choice of (u, h, r, s) , whenever the $\mathbf{P}$ -certificates $\vec{\pi}'$ that $E_s^{P^*_{u,h,r}}$ extracts out are convincing, their corresponding statements are true; otherwise, we can construct a uniform $\text{poly}(\Gamma(n))$ -time adversary that samples u, h, r, s uniformly at random, runs $E_s^{P^*_{u,h,r}}$, and outputs a random certificate from w' . Additionally, by the binding property of com and the collision-resistant property of $\mathcal{H}_n$ it follows that with overwhelming probability over (u, h) , there exists a vector of machines $\vec{S}^*$ such that except with negligible probability over the choice of r, s , it holds that if $E_s^{P^*_{u,h,r}}$ outputs a valid $w' \in \mathbf{R}_S(h, c, r)$, then the machines $\vec{S}$ in w' equals $\vec{S}^*$.¹⁴ By a union bound it follows that with overwhelming probability over (u, h) , there exists a vector of machines $\vec{S}^*$ such that except with negligible probability over the choice of r, s , the following holds: a) if $E_s^{P^*_{u,h,r}}$ outputs a valid $w' \in \mathbf{R}_S(h, c, r)$, then the machines $\vec{S}$ in w' equals $\vec{S}^*$, and b) all accepting certificates $\vec{\pi}'$ prove

¹⁴Note that for this to hold, we here rely on the fact that binding of com and collision-resistance of $\mathcal{H}_n$ hold also for circuits of size $\text{poly}(\Gamma(n))$; however, as mentioned, by slightly modifying the protocol as in [BG02], this assumption can be weakened to just collision resistance against polynomial-size circuits.

true statements. Let us refer to such pairs (u, h) as **good**.

For any valid “fake” witness $w' = (\vec{S}, j', \vec{s}', \vec{\pi}', \vec{\sigma}', \vec{\lambda}', \rho') \in \mathbf{R}_S(h, c, r)$ define a machine $M_{w'}$ (using $\vec{S}$ in w') that given the input $(j', \vec{s}', \vec{\sigma}')$ of length smaller than $2kn^2$, outputs r :

Machine $M_{w'}$: $M_{w'}(1^n, j, \vec{s}, \vec{\sigma})$ lets l^* be the largest l such that $j > n^{l-1}$. $M_{w'}$ next runs the machines $\tilde{S}_{l^*}, \tilde{S}_{l^*-1}, \dots, S_1$ in sequence as follows: $\tilde{S}_{l^*}$ is run on input $1^n, j^{l^*}, s^{l^*}$ and σ^{l^*} ; let λ^{l^*-1} denote its output. Next for each $l \leq l^*$, $\tilde{S}_l$ is given $1^n, j^l, s^l, \sigma^{\geq l}$ and $[\lambda^{\geq l}]_{j,l}$ where $\lambda^{\geq l}$ are the outputs of the executions of $\tilde{S}_{l+1}, \dots, \tilde{S}_{l^*}$. Finally, M outputs the string r returned by $\tilde{S}_1$.

Note that by definition, if all the **P**-certificates in w' prove true statements, then $M_{w'}$ given the input $(j', \vec{s}', \vec{\sigma}')$ indeed outputs r . However, for any machine M , since the input to the machine M is of length $2kn^2$, it follows by a counting argument that only for a negligible fraction of length $3kn^2$ strings r , there exists some input that makes M output r . Thus, whenever (u, h) is **good** (which happens with overwhelming probability), except with negligible probability (over the choice of r, s) **BAD** cannot happen; it follows that **BAD** can only happen with negligible probability, which is a contradiction. $\square$

4.2.2 Concurrent $\mathcal{ZK}$ of Protocol k

The simulator S for Protocol k will define $k+1$ “helper” simulators $S_1, \dots, S_{k+1}$. Before providing the formal definition of $S_1, \dots, S_{k+1}$, let us first describe the interaction among them.

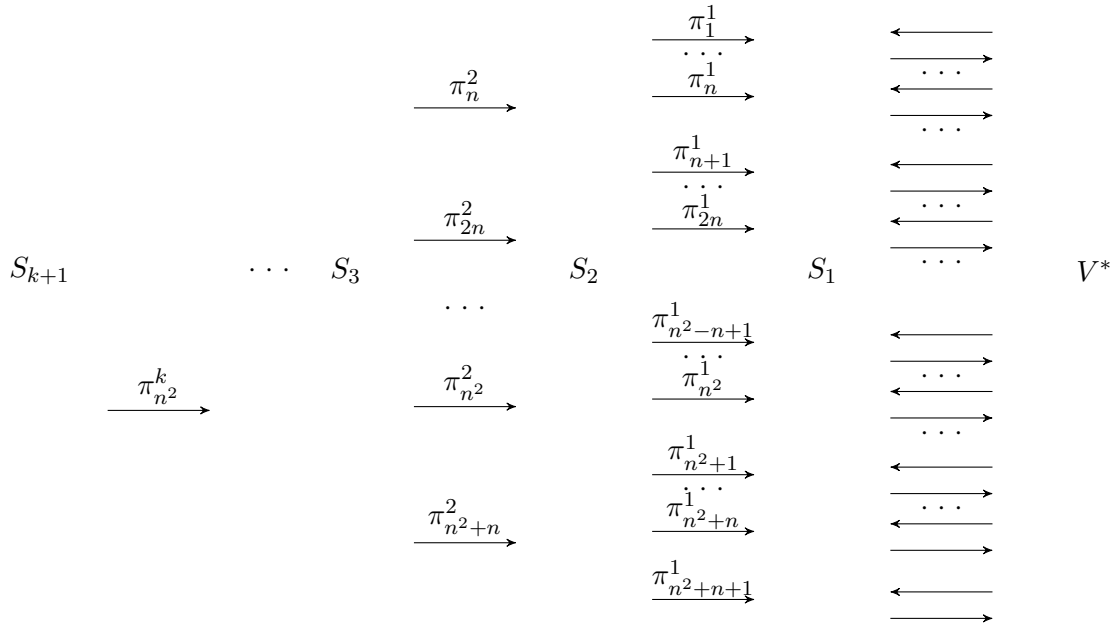


Figure 8: Simulation of protocol (P_k, V_k) for $k = 3$.

Recall that in the simulation of Protocol 1, S_1 is an interactive machine that communicates with a concurrent verifier V^* , on the “right”, while expecting to receive a **P**-certificates (j, π_j) from S_2 , on the “left”, for *every* communication round j in the right interaction with V^* ; S_1 then makes use of these certificates to complete the right interaction with V^* (and more specifically,

to complete the WIUAs it is supposed to provide V^*). In the simulation of Protocol k , S_1 still communicates with V^* on the “right”, but now additionally expects to receive **P**-certificates from all of $S_2, \dots, S_{k+1}$ on the “left”. In more detail, recall that a communication round in the “right” interaction refers to a verifier message (sent by V^*) followed by a prover message (sent by S_1). Now, in each communication round j in the right interaction, upon receiving a message from the verifier V^* , S_1 also expects to receive $(j, 1, \pi_j^1)$ from S_2 , and furthermore, for every $2 \leq l \leq k$, if $j \bmod n^{l-1} = 0$, then S_1 additionally expects to receive (j, l, π_j^l) from S_{l+1} . In other words, S_1 expects to receive a “level- l ” certificate (of the form $(j = a \cdot n^{l-1}, l, \pi_j^l)$ for some a) from S_{l+1} every n^{l-1} communication rounds. Roughly speaking, each such “level- l ” certificate, certifies that all “level- $(l-1)$ ” certificates up to round j were actually generated by S_l ; and those “level- $(l-1)$ ” certificates certify that S_{l-1} actually generated the “level- $(l-2)$ ” certificates up until round j , etc. See Figure 8 for an illustration of the communication pattern between $V^*, S_1, \dots, S_{k+1}$.

For every $2 \leq l \leq k$, for S_l to be able to generate its level $(l-1)$ -certificates, S_l internally emulates the interaction among $S_{l-1}, \dots, S_1, V^*$, but additionally needs to receive all level- l' certificates, where $l' \geq l$; thus each machine S_l produces level- $l-1$ certificates on the “right”, while receiving level- l , level- $(l+1)$, ... level- k certificates from respectively $S_{l+1}, S_{l+2}, \dots, S_{k+1}$, on the “left”. See Figure 9 for an illustration of S_l .

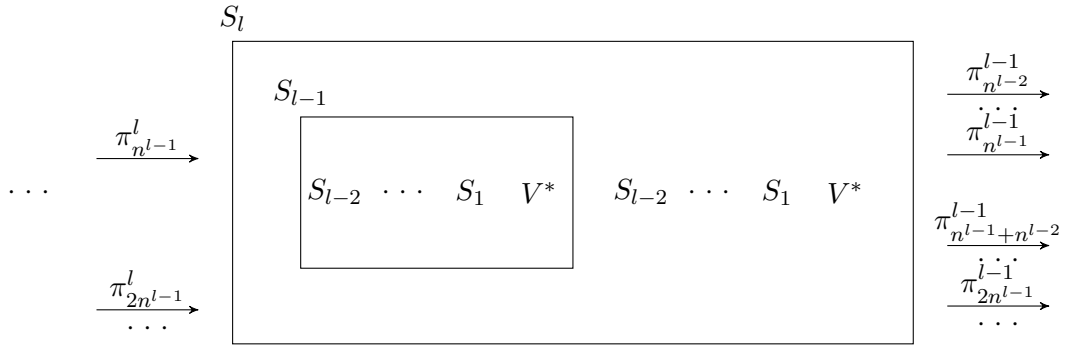


Figure 9: Simulator S_l .

We now define S_1 . As before, on a high-level, $S_1(1^n, x, \vec{M}, s, \ell)$, acts as a prover in a “right” interaction, communicating with a concurrent verifier V^* , while receiving some additional “external” messages on the “left”. (The input x is the statement to be proved, the input $\vec{M}$ will later be instantiated with the codes of $S_1, \dots, S_k$, and the input (s, ℓ) is used to generate the randomness for S_1 ; s is the seed for the forward secure pseudorandom generator g , and ℓ is the number of n -bit long blocks to be generated using g .)

Let us now specify how S_1 generates prover messages in its “right” interaction with V^* . $S_1(1^n, x, \vec{M}, s, \ell)$ acts as follows:

- Upon invocation, S_1 generates its “random-tape” by expanding the seed s ; more specifically, let $(s_\ell, s_{\ell-1}, \dots, s_1), (q_\ell, q_{\ell-1}, \dots, q_1)$ be the output of $g(s, \ell)$. Again, we assume without loss of generality that S_1 only needs n bits of randomness to generate any prover message; in order to generate its j ’th prover message, it uses q_j as randomness.
- Upon receiving a hash function h_i for session i in communication round j , S_1 provides a commitment c_i to the hash of the programs $\tilde{S}_1, \dots, \tilde{S}_k$ defined as follows.

$$- \tilde{S}_1(1^n, j, s', \sigma) = \text{wrap}(M_1(1^n, x, \vec{M}, s', j), V^*, \sigma, j).$$

- For $2 \leq l \leq k$, $\tilde{S}_l(1^n, j, s', \sigma) = \text{wrap}'(M_l(1^n, x, \vec{M}, s', j), \sigma, j)$ where $\text{wrap}'(A, \sigma, j)$ is the program that executes A for j “communication rounds,” while allowing A to receive σ as external messages “on the left”, and finally outputs the set of messages generated by A “on the right”—recall that M_l will be instantiated by S_l , who emulates the interaction among $S_{l-1}, \dots, S_1, V^*$, receives level- l' certificates for $l' \geq l$ externally “on the left”, and generates level- $(l-1)$ certificates on the “right”; “communication rounds” here still refer to the communication rounds of S_1 and V^* . (wrap' simply returns $\perp$ whenever A does not have the specified structure.)
- Upon receiving a challenge r_i in session i during the j th communication round, S_1 needs to provide a WIUA. To do so, S_1 collects the witness as follows.
 - Let l^* be the largest l such that $j \geq n^{l-1}$.
 - For $1 \leq l \leq l^*$, set $s^l = s_{\lfloor j \rfloor_{n^{l-1}}}$ (i.e., the seed corresponding to communication rounds $\lfloor j \rfloor_{n^{l-1}}$; recall that $\lfloor j \rfloor_x \triangleq j - (j \bmod x)$).
 - For $1 \leq l \leq l^*$, recall that S_1 expects to have received $a_l = \lfloor j \rfloor_{n^{l-1}}/n^{l-1}$ messages from S_{l+1} of the form $(a \cdot n^{l-1}, l, \pi_{a \cdot n^{l-1}}^l)$ for $a \in [a_l]$.
 - * Let π^l be the **P**-certificate in the last message received from S_{l+1} ; by construction, this message was received in round $\lfloor j \rfloor_{n^{l-1}}$ and thus we have $\pi^l = \pi_{\lfloor j \rfloor_{n^{l-1}}}^l$.
 - * Let λ^l be the messages received from S_{l+1} up until and including round $\lfloor j \rfloor_{n^l}$; by construction, since S_{l+1} generates a message every n^{l-1} communication rounds, λ^l contains a total of $\lfloor j \rfloor_{n^l}/n^{l-1}$ messages.
 - * Let σ^l be the messages generated by S_{l+1} after round $\lfloor j \rfloor_{n^l}$ but before round $\lfloor j \rfloor_{n^{l-1}}$ (thus, we exclude the last message π^l and the messages included in λ^l); since there are at most n^l communication rounds after round $\lfloor j \rfloor_{n^l}$ and before round $\lfloor j \rfloor_{n^{l-1}}$, and (again) S_{l+1} generates a message every n^{l-1} rounds, σ^l contains at most n messages; each such message is of length $n + O(\log n) < 2n$.
 - For $l^* < l \leq k$, let $\lambda_l = \text{null}$. (Note that also $\lambda_{l^*} = \text{null}$ since $\lfloor j \rfloor_{n^{l^*}} = 0$.)
 - Finally, let ρ and $\vec{S}$ be the randomness and machines, respectively, used to generate the commitment c_i in the i^{th} session.

If S_1 fails to find a valid witness, S_1 simply halts. Otherwise, S_1 uses the above witness to provide an honest WIUA to V^* that

1. (Commitment consistency:) $c_i = \text{com}(h_i(\vec{S}_1, \dots, \vec{S}_k); \rho)$,
2. (Input certification:) $|\vec{\sigma}| \leq 2kn^2$, $\lambda^{\geq l^*} = \text{null}$ and for $2 \leq l \leq l^*$, π^l certifies that $\tilde{S}_l(1^n, \lfloor j \rfloor_{n^{l-1}}, s^l, ([\lambda^{\geq l}]_{\lfloor j \rfloor_{n^{l-1}}}, \sigma^{\geq l})) = \lambda^{l-1}$,
3. (Prediction correctness:) π^1 certifies that $\tilde{S}_1(1^n, j, s^1, ([\lambda^{\geq 1}]_j, \sigma^{\geq 1})) = r_i$

Remark 2. Above, for every $1 \leq l \leq l^*$, S_1 uses the **P**-certificates π^l to certify that the execution of $\tilde{S}_l$ up until communication round $\lfloor j \rfloor_{n^{l-1}}$ when providing $\tilde{S}_l$ with the “certified” inputs $[\lambda^{\geq l}]_{\lfloor j \rfloor_{n^{l-1}}}$ and “dangling” inputs $\sigma^{\geq l}$. The bracket operator is used to ensure that the inputs given to $\tilde{S}_l$ are identically the same as were given to it when generating the **P**-certificate π^l at round $\lfloor j \rfloor_{n^l}$ (or else the statement proved by π^l would be different from the one that S_1 needs to provide a certificate about). The bracket operator simply “filters” out all messages that are generated at or after communication round $\lfloor j \rfloor_{n^{l-1}}$.

As noted above, by construction, $\vec{\sigma}$ always satisfies the appropriate length restrictions. Thus, the only thing we need to ensure is that the certificates received by S_1 indeed prove the “right” statements for S_1 to be able to complete its WIUAs; we shall see why this is the case shortly.

We now turn to defining S_l for $2 \leq l \leq k+1$, inductively. Suppose $S_1, \dots, S_{l-1}$ are defined. $S_l(1^n, x, \vec{M}, s, \ell)$ emulates the interaction among $S_{l-1}(1^n, x, \vec{M}, s, \ell), \dots, S_1(1^n, x, \vec{M}, s, \ell), V^*$ for ℓ communication rounds, while expecting to receive external messages “on the left”.

- In each communication round j with $j \bmod n^{l-1} = 0$, after V^* sends a verifier message m_j , we distinguish two cases.
 - If $l = 2$, S_2 generates a certificate π_j^1 (using P_{cert}) that $\text{wrap}(S_1(1^n, x, \vec{M}, s_j, j), V^*, \tau, j) = m_j$, where τ is the set of messages S_1 has received so far, and outputs $(j, 1, \pi_j^1)$.
 - If $l > 2$, S_l continues to emulate the round to the point that (the internally emulated) S_{l-1} outputs its message $(j, l-2, \pi_j^{l-2})$, and then S_l generates a certificate π_j^{l-1} that $\text{wrap}'(S_{l-1}(1^n, x, \vec{M}, s_j, j), \tau, j) = \eta$, where τ is the set of messages that S_{l-1} has received so far and η is the set of messages S_{l-1} has generated so far (in the internal emulation). Then S_l outputs the message $(j, l-1, \pi_j^{l-1})$.
- In each communication round j s.t., $j \bmod n^l = 0$, after generating its message $(j, l-1, \pi_j^{l-1})$, S_l expects to receive external messages $(j, l'-1, \pi_j^{l'-1})$ “on the left” for every $l' > l$ such that $j \bmod n^{l'-1} = 0$. S_l simply relays these messages to its internally emulated $S_{l-1}, \dots, S_1$.

Finally, S_l outputs its own view at the end of the execution (which in particular, contains the view of V^* , and all the messages generated by S_l).

Note that the construction of $S_2, \dots, S_{k+1}$ ensures that S_1 will always have the appropriate certificates to complete every WIUA it reaches; as a consequence, S_1 never gets “stuck”.

Let $\vec{S} = (S_1, \dots, S_k)$. The final simulator $S(1^n, x)$ simply runs $S_{k'}(1^n, x, \vec{S}, s, T(n+|x|))$, where s is a uniformly random string of length n , $T(n+|x|)$ is a polynomial upper-bound on the number of messages sent by V^* on input 1^n and statement $x \in \{0, 1\}^{\text{poly}(n)}$, and $k' = \lceil \log_n T(n+|x|) \rceil + 1$, and then extracts and outputs the view of V^* from the output of $S_{k'}$. Note that since T is polynomial in n , k' is a constant.

Running-time of S We first note that essentially the same argument as for Protocol 1 shows that S_1 runs in polynomial time: It only takes S_1 polynomial-time to generate the commitments in Phase 1 (since V^* has a polynomial-length description, and the programs $\tilde{S}_l$ ’s have length polynomial in the size of V^*). During the WIUA in Phase 2, the length of the witness used by the simulator is polynomial in length of the programs $\tilde{S}_l$ ’s, and their inputs and outputs, all of which are polynomial in the circuit-size of V^* . Since the **P**-certificates verification time is polynomial in the length of the statement proved, it follows that the relation being proved in the WIUA has a time complexity that is upper bounded by a fixed polynomial in the length of V^* . By the relative prover efficiency condition of the WIUA, each such proof only requires some fixed polynomial-time, and thus the whole execution of S_1 takes some fixed polynomial time (in the size of V^* and thus also in the length of x .) It directly follows that also $\tilde{S}_1$ ’s running-time is polynomially bounded.

It now follows by an induction that S_l and thus $\tilde{S}_l$ run in polynomial time for every constant l . Suppose S_{l-1} and $\tilde{S}_{l-1}$ run in polynomial time. Since S_l is simply providing certificates about the execution of $\tilde{S}_{l-1}$, it follows by the relative prover-efficiency condition of **P**-certificates, that S_l runs in polynomial time, and thus also $\tilde{S}_l$. Finally, as S simply runs $S_{k'}$ with a constant k' , the running-time of S is polynomially bounded as well.

Indistinguishability of the simulation Note that by construction of S , it follows that the simulation never gets “stuck” in the sense that whenever V^* expects a WIUA in some session, S has an appropriate “fake” witness and can complete the WIUA using this “fake” witness. Indistinguishability of the simulation follows in identically the same way as for Protocol 1.

4.3 Dealing with Randomized $\mathbf{P}$ -certificates

As mentioned above, to simplify the exposition, our protocol uses strong $\mathbf{P}$ -certificate system $(P_{\text{cert}}, V_{\text{cert}})$ with *deterministic* prover and verifier strategies. We here sketch how to deal with the case when P_{cert} and V_{cert} are randomized.

- **Handling randomized V_{cert} .** If V_{cert} is randomized, we simply need to the verifier V generate the randomness for V_{cert} , but to guarantee soundness of the $\mathbf{P}$ -certificate, V needs to do so after the $\mathbf{P}$ -certificates are determined. We do this by adding a new communication round before Phase 2 where the prover first is asked to commit to the k $\mathbf{P}$ -certificates $\pi^1, \dots, \pi^k$ that it wants to use in Phase 2 (the honest prover should simply commit to $0^{k \cdot n}$) and next the verifier selects randomness $\rho^1, \dots, \rho^k$ for V_{cert} for each of these certificates. In Phase 2, the prover is then asked to demonstrate that for each certificate $l \in [k]$, V_{cert} using randomness ρ^l accepts π^l .
- **Handling randomized P_{cert} .** If P_{cert} is randomized, the helper simulators $S_2, \dots, S_{k+1}$ also become randomized. As with S_1 , there is now a potential “randomness-dependent” issue since the simulators generate certificates about their own behaviour in earlier communication rounds (in particular, S_1 needs to know the randomness of all “helper” simulators). We can break the circularity by using forward secure PRGs in exactly the same way as was done for S_1 ; each the simulator S_l use *independent* seeds $s^{(l)}$ for a forward secure PRG to expand the randomness for generating level- $(l - 1)$ certificates in each communication round, and then uses the seed $s_j^{(l)}$ as an input to $\tilde{S}_l$ ’s when generating certificates at communication round j .

4.4 A Note on Uniform Assumptions

We remark that even in the case of uniform soundness, our protocol currently relies on families of hash-functions collision-resistant also for non-uniform polynomial-time. Note, however, that for our soundness proof, it suffices to use commitment schemes that are binding for uniform polynomial-time algorithms and a WIUA where the proof of knowledge property is proven secure using a uniform security reduction. (We still need the hiding and the witness indistinguishability properties to hold for non-uniform polynomial-time to establish $\mathcal{ZK}$ with arbitrary auxiliary inputs). We see no obstacles in getting these properties by instantiating our protocol with statistically-hiding commitments and a “special-purpose” WIUA from [PR05], which also relies on statistically-hiding commitments, but we haven’t verified the details. In particular, if we only rely on statistically-hiding commitments where the (computational) binding hold against uniform polynomial-time algorithms, such commitment can be based on families of hash functions collision-resistant against uniform polynomial-time.

References

- [Bar01] Boaz Barak. How to go beyond the black-box simulation barrier. In *FOCS*, volume 0, pages 106–115, 2001.

- [BCC88] Gilles Brassard, David Chaum, and Claude Crépeau. Minimum disclosure proofs of knowledge. *J. Comput. Syst. Sci.*, 37(2):156–189, 1988.
- [BCCT12] Nir Bitansky, Ran Canetti, Alessandro Chiesa, and Eran Tromer. From extractable collision resistance to succinct non-interactive arguments of knowledge, and back again. In *ITCS*, pages 326–349, 2012.
- [BCCT13] Nir Bitansky, Ran Canetti, Alessandro Chiesa, and Eran Tromer. Recursive composition and bootstrapping for snarks and proof-carrying data. In *STOC*, 2013.
- [BCPT12] Eleanor Birrell, Kai-Min Chung, Rafael Pass, and Sidharth Telang. Randomness-dependent message security. Unpublished Manuscript, 2012.
- [BFLS91] László Babai, Lance Fortnow, Leonid A. Levin, and Mario Szegedy. Checking computations in polylogarithmic time. In *STOC*, pages 21–31, 1991.
- [BG02] Boaz Barak and Oded Goldreich. Universal arguments and their applications. In *Computational Complexity*, pages 162–171, 2002.
- [BG08] Boaz Barak and Oded Goldreich. Universal arguments and their applications. *SIAM J. Comput.*, 38(5):1661–1694, 2008.
- [BGGL01] Boaz Barak, Oded Goldreich, Shafi Goldwasser, and Yehuda Lindell. Resettable-sound zero-knowledge and its applications. In *FOCS*, pages 116–125, 2001.
- [BH92] Donald Beaver and Stuart Haber. Cryptographic protocols provably secure against dynamic adversaries. In Rainer A. Rueppel, editor, *EUROCRYPT*, volume 658 of *Lecture Notes in Computer Science*, pages 307–323. Springer, 1992.
- [BLV06] Boaz Barak, Yehuda Lindell, and Salil P. Vadhan. Lower bounds for non-black-box zero knowledge. *J. Comput. Syst. Sci.*, 72(2):321–391, 2006.
- [Bon03] Dan Boneh, editor. *Advances in Cryptology - CRYPTO 2003, 23rd Annual International Cryptology Conference, Santa Barbara, California, USA, August 17-21, 2003, Proceedings*, volume 2729 of *Lecture Notes in Computer Science*. Springer, 2003.
- [BOV07] Boaz Barak, Shien Jin Ong, and Salil P. Vadhan. Derandomization in cryptography. *SIAM J. Comput.*, 37(2):380–400, 2007.
- [BP04a] Boaz Barak and Rafael Pass. On the possibility of one-message weak zero-knowledge. In Moni Naor, editor, *TCC*, volume 2951 of *Lecture Notes in Computer Science*, pages 121–132. Springer, 2004.
- [BP04b] Mihir Bellare and Adriana Palacio. Towards plaintext-aware public-key encryption without random oracles. In *ASIACRYPT*, pages 48–62, 2004.
- [BP12] Nir Bitansky and Omer Paneth. From the impossibility of obfuscation to a new non-black-box simulation technique. In *FOCS*, 2012.
- [BR93] Mihir Bellare and Phillip Rogaway. Random oracles are practical: A paradigm for designing efficient protocols. In Dorothy E. Denning, Raymond Pyle, Ravi Ganesan, Ravi S. Sandhu, and Victoria Ashby, editors, *ACM Conference on Computer and Communications Security*, pages 62–73. ACM, 1993.

- [BS05] Boaz Barak and Amit Sahai. How to play almost any mental game over the net - concurrent composition via super-polynomial simulation. In *FOCS*, pages 543–552, 2005.
- [BY03] Mihir Bellare and Bennet S. Yee. Forward-security in private-key cryptography. In Marc Joye, editor, *CT-RSA*, volume 2612 of *Lecture Notes in Computer Science*, pages 1–18. Springer, 2003.
- [CD09] Ran Canetti and Ronny Ramzi Dakdouk. Towards a theory of extractable functions. In Omer Reingold, editor, *TCC*, volume 5444 of *Lecture Notes in Computer Science*, pages 595–613. Springer, 2009.
- [CGGM00] Ran Canetti, Oded Goldreich, Shafi Goldwasser, and Silvio Micali. Resettable zero-knowledge (extended abstract). In *STOC*, pages 235–244, 2000.
- [CKPR01] Ran Canetti, Joe Kilian, Erez Petrank, and Alon Rosen. Black-box concurrent zero-knowledge requires $\tilde{\omega}(\log n)$ rounds. In *STOC*, pages 570–579, 2001.
- [CLP12] Ran Canetti, Huijia Lin, and Omer Paneth. Public coin concurrent zero-knowledge in the global hash model. Manuscript, 2012.
- [Dam91] Ivan Damgård. Towards practical public key systems secure against chosen ciphertext attacks. In Joan Feigenbaum, editor, *CRYPTO*, volume 576 of *Lecture Notes in Computer Science*, pages 445–456. Springer, 1991.
- [Dam00] Ivan Damgård. Efficient concurrent zero-knowledge in the auxiliary string model. In *EUROCRYPT*, pages 418–430, 2000.
- [DFH12] Ivan Damgård, Sebastian Faust, and Carmit Hazay. Secure two-party computation with low communication. In *TCC*, pages 54–74, 2012.
- [DGS09] Yi Deng, Vipul Goyal, and Amit Sahai. Resolving the simultaneous resettability conjecture and a new non-black-box simulation strategy. In *FOCS*, pages 251–260, 2009.
- [DNS04] Cynthia Dwork, Moni Naor, and Amit Sahai. Concurrent zero-knowledge. *J. ACM*, 51(6):851–898, 2004.
- [DS98] Cynthia Dwork and Amit Sahai. Concurrent zero-knowledge: Reducing the need for timing constraints. In *CRYPTO*, pages 177–190, 1998.
- [FGL⁺91] Uriel Feige, Shafi Goldwasser, László Lovász, Shmuel Safra, and Mario Szegedy. Approximating clique is almost np-complete (preliminary version). In *FOCS*, pages 2–12. IEEE Computer Society, 1991.
- [FS90] Uriel Feige and Adi Shamir. Witness indistinguishable and witness hiding protocols. In *STOC*, pages 416–426, 1990.
- [GGM86] Oded Goldreich, Shafi Goldwasser, and Silvio Micali. How to construct random functions. *J. ACM*, 33(4):792–807, 1986.
- [GGPR13] Rosario Gennaro, Craig Gentry, Bryan Parno, and Mariana Raykova. Quadratic span programs and succinct nizks without pcps. In *EUROCRYPT*, 2013.

- [GJ10] Vipul Goyal and Abhishek Jain. On the round complexity of covert computation. In Schulman [Sch10], pages 191–200.
- [GJO⁺12] Vipul Goyal, Abhishek Jain, Rafail Ostrovsky, Silas Richelson, and Ivan Visconti. Concurrent zero knowledge in the bounded player model. Cryptology ePrint Archive, Report 2012/279, 2012. <http://eprint.iacr.org/>.
- [GK96] Oded Goldreich and Hugo Krawczyk. On the composition of zero-knowledge proof systems. *SIAM Journal on Computing*, 25(1):169–192, 1996.
- [GK03] Shafi Goldwasser and Yael Tauman Kalai. On the (in)security of the fiat-shamir paradigm. In *FOCS*, pages 102–, 2003.
- [GLR11] Shafi Goldwasser, Huijia Lin, and Aviad Rubinfeld. Delegation of computation without rejection problem from designated verifier cs-proofs. *IACR Cryptology ePrint Archive*, 2011:456, 2011.
- [GM84] Shafi Goldwasser and Silvio Micali. Probabilistic encryption. *J. Comput. Syst. Sci.*, 28(2):270–299, 1984.
- [GMR89] Shafi Goldwasser, Silvio Micali, and Charles Rackoff. The knowledge complexity of interactive proof systems. *SIAM Journal on Computing*, 18(1):186–208, 1989.
- [Gol93] Oded Goldreich. A uniform-complexity treatment of encryption and zero-knowledge. *J. Cryptology*, 6(1):21–53, 1993.
- [Gol01] Oded Goldreich. *Foundations of Cryptography — Basic Tools*. Cambridge University Press, 2001.
- [Gol02] Oded Goldreich. Concurrent zero-knowledge with timing, revisited. In *STOC*, pages 332–340, 2002.
- [Goy12] Vipul Goyal. Positive results for concurrently secure computation in the plain model. In *FOCS*, 2012.
- [Gro10] Jens Groth. Short pairing-based non-interactive zero-knowledge arguments. In *ASIACRYPT*, pages 321–340, 2010.
- [GS12] Divya Gupta and Amit Sahai. On constant-round concurrent zero-knowledge from a knowledge assumption. Cryptology ePrint Archive, Report 2012/572, 2012. <http://eprint.iacr.org/>.
- [GW11] Craig Gentry and Daniel Wichs. Separating succinct non-interactive arguments from all falsifiable assumptions. In *STOC*, pages 99–108, 2011.
- [HILL99] Johan Håstad, Russell Impagliazzo, Leonid Levin, and Michael Luby. A pseudorandom generator from any one-way function. *SIAM Journal on Computing*, 28:12–24, 1999.
- [HT98] Satoshi Hada and Toshiaki Tanaka. On the existence of 3-round zero-knowledge protocols. In Hugo Krawczyk, editor, *CRYPTO*, volume 1462 of *Lecture Notes in Computer Science*, pages 408–423. Springer, 1998.

- [Kil95] Joe Kilian. Improved efficient arguments (preliminary version). In *CRYPTO*, pages 311–324, 1995.
- [KP01] Joe Kilian and Erez Petrank. Concurrent and resettable zero-knowledge in polynomial algorithm rounds. In *STOC*, pages 560–569, 2001.
- [KPR98] Joe Kilian, Erez Petrank, and Charles Rackoff. Lower bounds for zero knowledge on the internet. In *FOCS*, pages 484–492, 1998.
- [Lin03] Yehuda Lindell. Bounded-concurrent secure two-party computation without setup assumptions. In *STOC*, pages 683–692, 2003.
- [Lip12] Helger Lipmaa. Progression-free sets and sublinear pairing-based non-interactive zero-knowledge arguments. In *TCC*, pages 169–189, 2012.
- [Mer89] Ralph C. Merkle. A certified digital signature. In *CRYPTO*, pages 218–238, 1989.
- [Mic00] Silvio Micali. Computationally sound proofs. *SIAM J. Comput.*, 30(4):1253–1298, 2000.
- [Nao91] Moni Naor. Bit commitment using pseudorandomness. *Journal of Cryptology*, 4:151–158, 1991.
- [Nao03] Moni Naor. On cryptographic assumptions and challenges. In Boneh [Bon03], pages 96–109.
- [Pas03a] Rafael Pass. On deniability in the common reference string and random oracle model. In Boneh [Bon03], pages 316–337.
- [Pas03b] Rafael Pass. Simulation in quasi-polynomial time, and its application to protocol composition. In *EUROCRYPT*, pages 160–176, 2003.
- [Pas04a] Rafael Pass. Bounded-concurrent secure multi-party computation with a dishonest majority. In *STOC*, pages 232–241, New York, NY, USA, 2004. ACM.
- [Pas04b] Rafael Pass. Bounded-concurrent secure multi-party computation with a dishonest majority. In *STOC*, pages 232–241, 2004.
- [Pop63] Karl Popper. *Conjectures and Refutations: The Growth of Scientific Knowledge*. Routledge, 1963.
- [PR03a] Rafael Pass and Alon Rosen. Bounded-concurrent secure two-party computation in a constant number of rounds. In *FOCS*, pages 404–, 2003.
- [PR03b] Rafael Pass and Alon Rosen. How to simulate using a computer virus. Unpublished manuscript, 2003.
- [PR05] Rafael Pass and Alon Rosen. Concurrent non-malleable commitments. In *FOCS*, pages 563–572, 2005.
- [PRS02] Manoj Prabhakaran, Alon Rosen, and Amit Sahai. Concurrent zero knowledge with logarithmic round-complexity. In *FOCS*, pages 366–375, 2002.

- [PRT11] Rafael Pass, Alon Rosen, and Wei-Lung Dustin Tseng. Public-coin parallel zero-knowledge for np. *J. Cryptology*, 2011.
- [PTV12] Rafael Pass, Wei-Lung Dustin Tseng, and Muthuramakrishnan Venkitasubramaniam. Concurrent zero-knowledge, revisited. Unpublished manuscript, 2012.
- [PTW11] Rafael Pass, Wei-Lung Dustin Tseng, and Douglas Wikström. On the composition of public-coin zero-knowledge protocols. *SIAM J. Comput.*, 40(6):1529–1553, 2011.
- [PV08] Rafael Pass and Muthuramakrishnan Venkitasubramaniam. On constant-round concurrent zero-knowledge. In *TCC*, pages 553–570, 2008.
- [PV10] Rafael Pass and Muthuramakrishnan Venkitasubramaniam. Private coins versus public coins in zero-knowledge proofs. To appear in *TCC 2010*, 2010.
- [RK99] Ransom Richardson and Joe Kilian. On the concurrent composition of zero-knowledge proofs. In *Eurocrypt*, pages 415–432, 1999.
- [Rog06] Phillip Rogaway. Formalizing human ignorance. In *VIETCRYPT*, pages 211–228, 2006.
- [Ros00] Alon Rosen. A note on the round-complexity of concurrent zero-knowledge. In *CRYPTO*, pages 451–468, 2000.
- [Sch91] Claus-Peter Schnorr. Efficient signature generation by smart cards. *J. Cryptology*, 4(3):161–174, 1991.
- [Sch10] Leonard J. Schulman, editor. *Proceedings of the 42nd ACM Symposium on Theory of Computing, STOC 2010, Cambridge, Massachusetts, USA, 5-8 June 2010*. ACM, 2010.
- [Val08] Paul Valiant. Incrementally verifiable computation or proofs of knowledge imply time/space efficiency. In Ran Canetti, editor, *TCC*, volume 4948 of *Lecture Notes in Computer Science*, pages 1–18. Springer, 2008.

Unprovable Security of Perfect NIZK and Non-interactive Non-malleable Commitments

Rafael Pass*
Cornell University
rafael@cs.cornell.edu

May 27, 2015

Abstract

We present barriers to provable security of two fundamental (and well-studied) cryptographic primitives *perfect non-interactive zero knowledge (NIZK)*, and *non-malleable commitments*:

- Black-box reductions cannot be used to demonstrate *adaptive* soundness (i.e., that soundness holds even if the statement to be proven is chosen as a function of the common reference string) of any statistical (and thus also perfect) NIZK for $\mathcal{NP}$ based on any “standard” intractability assumptions.
- Black-box reductions cannot be used to demonstrate non-malleability of non-interactive, or even 2-message, commitment schemes based on any “standard” intractability assumptions.

We emphasize that the above separations apply even if the construction of the considered primitives makes a *non-black-box* use of the underlying assumption.

As an independent contribution, we suggest a taxonomy of game-based intractability assumptions.

*A preliminary version of this paper appeared in TCC’13. Pass is supported in part by a Alfred P. Sloan Fellowship, Microsoft New Faculty Fellowship, NSF Award CNS-1217821, NSF CAREER Award CCF-0746990, NSF Award CCF-1214844, AFOSR YIP Award FA9550-10-1-0093, and DARPA and AFRL under contract FA8750-11-2-0211. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the Defense Advanced Research Projects Agency or the US Government.

1 Introduction

Modern Cryptography relies on the principle that cryptographic schemes are proven secure based on mathematically precise assumptions; these can be *general*—such as the existence of one-way functions—or *specific*—such as the hardness of factoring products of large primes. The security proof is a *reduction* that transforms any attacker A of the scheme into a machine that breaks the underlying assumption (e.g., inverts an alleged one-way function). This study has been extremely successful, and during the past three decades many cryptographic tasks have been put under rigorous treatment and numerous constructions realizing these tasks have been proposed under a number of well-studied complexity-theoretic hardness assumptions.

In this paper, we study two fundamental cryptographic primitives—*perfect non-interactive zero-knowledge with adaptive statements* and *non-interactive non-malleable commitments*—for which security proofs based on well-studied intractability assumptions have remained elusive.

Perfect NIZK with Adaptive Inputs A non-interactive zero-knowledge (NIZK) protocol [?] is protocol between two parties, a Prover, and a Verifier, through which the Prover can non-interactively (i.e., by sending a single message π) convince the Verifier of the validity of a statement x , only if x is true (this is called the *soundness* property), while at the same time revealing nothing beyond the fact that x is true (this is called the *zero-knowledge* property). To make such constructs possible both parties are additionally assumed to have access to a “Common Reference String” (CRS) that has been ideally sampled according to some distribution. The original definition of [?] only considered *non-adaptive* notions of soundness and zero-knowledge: Roughly speaking, the (non-adaptive) soundness condition requires that for every false statement $x \notin L$, with high probability over the choice of the CRS, any proof π output by a malicious prover will be rejected by the verifier. The (non-adaptive) zero-knowledge property, on the other hand, requires that for every true statement $x \in L$, the joint distribution consisting of the reference string, and an honestly generated proof, can be reconstructed by a simulator. In both of these properties, the statement x is required to be *fixed* before the reference string is known. Feige, Lapidot and Shamir [?] introduced stronger *adaptive* notions of both soundness and zero-knowledge; roughly speaking, here soundness and zero-knowledge should hold even if the statement x is adversarially chosen *as a function of* the reference string.

As with traditional zero-knowledge protocols, NIZKs come in several flavors: *computational NIZK*, *statistical NIZK*, and *perfect NIZK*. In the computational notion, the simulator’s output is only required to be computationally indistinguishable from an honestly generated view, whereas in the statistical (resp. perfect) variants, it is required to be statistically close (resp. identical) to an honestly generated view. Computational NIZK with adaptive zero-knowledge and soundness were constructed early on based on standard cryptographic intractability assumptions [?, ?], but constructions of statistical and perfect NIZK were elusive.

Only recently, a breakthrough result by Groth, Ostrovsky and Sahai (GOS) [?] provided a construction of a perfect NIZK for $\mathcal{NP}$ based on the hardness of a number theoretic assumption over bilinear groups. Their protocol satisfies the adaptive notion of zero-knowledge; however, it only satisfies the non-adaptive notion of soundness (that is, soundness is no longer guaranteed to hold if the attacker chooses a statement $x \notin L$ as a function of the common reference string). We focus on whether there exists a perfect NIZK for $\mathcal{NP}$ with both adaptive soundness and zero-knowledge.

A step towards answering this question appears in the work of Abe and Fehr [?], which presented a perfect NIZK for $\mathcal{NP}$ with both adaptive soundness and zero-knowledge, using an “knowledge-extraction” assumption (similar to the “knowledge-of-exponent” assumption of [?]), as opposed

to a computational-intractability assumption. Abe and Fehr also demonstrate that certain (arguably natural) types of proof techniques—which they refer to as “direct” black-box reductions—cannot be used to prove adaptive soundness of perfect NIZKs for $\mathcal{NP}$. Their notion of a “direct” proof, however, is quite restrictive (very roughly speaking, it requires the security reduction to “directly embed” some hard instance into the CRS in a “structure preserving way”).¹

Non-interactive Non-malleable Commitments Often described as the “digital” analogue of sealed envelopes, commitment schemes enable a *sender* to commit itself to a value while keeping it secret from the *receiver*. This property is called *hiding*. Furthermore, the commitment is *binding*, and thus in a later stage when the commitment is opened, it is guaranteed that the “opening” can yield only a single value determined in the committing stage. For many applications, however, the most basic security guarantees of commitments are not sufficient. For instance, the basic definition of commitments does not rule out an attack where an adversary, upon seeing a commitment to a specific value v , is able to commit to a related value (say, $v - 1$), even though it does not know the actual value of v . This kind of attack might have devastating consequences if the underlying application relies on the *independence* of committed values (e.g., consider a case in which the commitment scheme is used for securely implementing a contract bidding mechanism). In order to address the above concerns, Dolev, Dwork and Naor introduced the concept of *non-malleable commitments* [?]. Loosely speaking, a commitment scheme is said to be non-malleable if it is infeasible for an adversary to “maul” a commitment to a value v into a commitment to a related value $\tilde{v}$.

More precisely, we consider a *man-in-the-middle* (MIM) attacker that participates in two concurrent executions of a commitment scheme Π ; in the “left” execution it interacts with an honest committer; in the “right” execution it interacts with an honest receiver. Additionally, we assume that the players have n -bit identities (where n is polynomially related to the security parameter), and that the commitment protocol depends only on the identity of the committer; we sometimes refer to this as the identity of the interaction.² Intuitively, Π being non-malleable means that if the identity of the right interaction is different than the identity of the left interaction (i.e., A does not use the same identity as the left committer), the value A commits to on the right does not depend on the value it receives a commitment to on the left; this is formalized by requiring that for any two values v_1, v_2 , the value A commits to after receiving left commitments to v_1 or v_2 are indistinguishable.³

The first non-malleable commitment protocol was constructed by Dolev, Dwork and Naor [?] in 1991. The security of their protocol relies on the minimal assumption of one-way functions and requires $\Omega(\log k)$ rounds of interaction, where $k \in N$ is the length of party identities. The round-complexity of non-malleable commitments has since been extensively studied (see e.g., [?, ?, ?, ?, ?, ?]), leading up to *constant round* protocols based on one-way functions [?, ?].

¹Among other things, the structure preserving property requires that if the “hard instance” being directly embedded in the CRS is true, the CRS is valid, and if the hard instance is false, then the CRS is “invalid”. This property can never hold when considering NIZK in the Uniform Reference String model (as every CRS is valid), and as such their result holds vacuously when considering NIZK in the Uniform Reference String model.

²Non-malleable commitments are sometimes also considered in settings where players do not have identities. However, as shown in [?], any non-malleable commitment that handles sufficiently long identities can be turned into a non-malleable commitment without identities (and any non-malleable commitment without identities can trivially be turned into one with identities). Since our goal is to prove lower bounds, we focus on the more general (relaxed) notion of non-malleability with respect to identities.

³Note that the value A commits to is not efficiently computable from the transcript of the right interaction; nevertheless, if the commitment is statistically binding, the value is determined with overwhelming probability. Our focus here is on non-malleability for statistically-binding commitments (as is typically the case in the literature).

The question of whether *non-interactive*, or even 2-round, non-malleable commitments exist, however, is wide open. (We note that in the Common Reference String model, constructions of non-interactive non-malleable commitments are known [?]; we here focus on constructions in the plain model, without any set-up.) Some initial progress towards this question can be found in [?] where a construction of non-interactive non-malleable commitments based on a new hardness assumption is given. This assumption, however, has a strong non-malleability flavor; as such, it provides little insight into the question of whether non-malleability can be obtained from a “pure” hardness assumptions (such as e.g., the hardness of factoring).

1.1 Our results

The main result of this paper is showing that Turing (i.e., black-box) reductions cannot be used to base the security of the above-mentioned primitives, on a general class of intractability assumption.

More precisely, following Naor [?] (see also [?, ?, ?, ?, ?]), we model an *intractability assumption* as an arbitrary game between a (potentially) unbounded challenger C , and an attacker A . The attacker A is said to break the challenger C with respect to the threshold t if it can make C output 1 with probability non-negligibly higher than the threshold t . An intractability assumption is defined as a pair (C, t) where C is a challenger and t is a threshold. All traditional cryptographic hardness assumptions (e.g., the hardness of factoring, the hardness of the discrete logarithm problem, the decisional Diffie-Hellman problem etc.) can be modeled as 2-round challengers C with the threshold t being either 0 (in case of the factoring or discrete logarithm problems) or $1/2$ (in case of the decisional Diffie-Hellman problem).⁴ In all these examples, C is polynomial-time; Naor [?] and Gentry and Wichs [?] refer to such assumptions as “falsifiable”. For generality, we (following [?]) refer to these as “efficient-challenger” assumptions. More generally, we refer to an assumption where the challenger can be implemented in time (resp. circuit size) $T(\cdot)$ as a “ $T(\cdot)$ -time (resp. size) challenger assumption”. Note that some “esoteric” assumptions such as the “one-more discrete logarithm assumption” [?, ?], or “adaptive one-way functions” [?], are not efficient-challenger assumptions, but they are exponential-time challenger assumptions.

Our first result rules out basing statistical (and thus also perfect) NIZK with adaptive soundness on efficient-challenger (a.k.a falsifiable) assumptions.

Theorem 1 (Informally stated). *Assume the existence of (non-uniformly hard) one-way functions. Then there exists an $\mathcal{NP}$ -language L such that the following holds. Let Π be a statistical non-interactive adaptively zero-knowledge argument for L , and let (C, t) be an efficient challenger assumption. Assume there exists a polynomial-time (resp. polynomial-size) Turing reduction R such that R^A breaks the C w.r.t. the threshold t for every A that breaks adaptive soundness of Π . Then C can be broken in polynomial-time (resp. by a polynomial-size circuit) with respect to the threshold t .*

We also show that if we additionally assume the existence of sub-exponential one-way functions, and consider constructions of NIZK for proving *any* polynomial-length (in the security parameter) statement in $\mathcal{NP}$ based on a particular *exponential-time* challenger assumption (C, t) , then the assumption can already be broken in polynomial time.

Moving on to non-interactive non-malleable commitments, we show that if non-malleability of a non-interactive, or two message, commitment scheme Π can be based on a efficient-challenger (resp. $T(\cdot)$ -size) challenger assumption (C, t) using a polynomial-time (resp. $T(\cdot)$ -sized) security reduction, then C can be broken in polynomial-time (resp. by a $\text{poly}(T(\cdot))$ -sized circuit).

⁴For instance, for the case of factoring, the challenger C picks two random k -bit primes p, q , and outputs $N = pq$; the attacker A sends back a number p' and C finally outputs 1 iff $p' \in \{p, q\}$.

Theorem 2 (Informally stated). *Let Π be a two-message commitment scheme, and let (C, t) be an efficient-challenger (resp. $T(\cdot)$ -size) assumption. Assume there exists a polynomial-time (resp. $T(\cdot)$ -size) Turing reduction R such that R^A breaks C w.r.t. the threshold t for every A that breaks non-malleability of Π . Then C can be broken in polynomial-time (resp. by a $\text{poly}(T(\cdot))$ -sized circuit) with respect to the threshold t .*

We emphasize that for all the above-mentioned results, the *construction* of the protocols Π need not make use of the underlying assumption in a black-box way; the only restriction we impose is that the security *reduction* (establishing the security of Π) is a Turing (i.e., black-box) reduction.

Why these primitives? On a very high-level, non-interactive statistical NIZK and non-interactive non-malleable commitments share three properties that enable our unprovability results: 1) they are both “non-interactive” primitives, and 2) whether the primitives get broken cannot be verified efficiently, and 3) they both have a zero-knowledge flavor (explicitly in the case of NIZK, and implicitly for the case of commitments). These are exactly the properties that we need for our unprovability results⁵, and consequently both unprovability results have significant overlaps in terms of the techniques employed.

Dealing with “Slightly” Non-black-box Security Reductions In this work we focus on ruling out security reductions that only use the attacker in a black-box way. In particular, the security reduction, although it may be a non-uniform algorithm, may *not* get any non-uniform advice about the attacker (more precisely, if it is a non-uniform algorithm, the same non-uniform advice should work for every attacker). In contrast, some quite commonly used proof techniques in cryptography rely on the reduction receiving a non-uniform advice string that *depends* on the attacker—this may be viewed as a slightly non-black-box use of the attacker. We mention that a recent work by Chung, Lin, Mahmoody and Pass [?] provides techniques for extending certain types of separation results for the black-box setting to deal also with reductions receiving non-uniform advice about the attacker. These techniques readily apply to our results, which thus also extend to rule out such “slightly non-black-box” security reductions.

A Taxonomy of Intractability Assumption As an independent contribution, we slightly generalize the notion of an intractability assumption from [?] (see also [?, ?, ?, ?, ?]) and provide a natural taxonomy of intractability assumptions based on 1) the *security threshold*, 2) the number of *communication rounds* in the security game, 3) the *computational complexity* of the game challenger, 4) the *communication complexity* of the challenger, and 5) the *computational complexity of the security reduction*. Our results, combined with [?, ?], demonstrate several natural primitives that may be (trivially) based on an assumption of a certain type (e.g., the soundness condition of a perfect NIZK can trivially be viewed as a bounded-round assumption), but cannot be based on a different type of assumption (e.g., an assumption where the challenger is efficient). Our results focus on understanding limitations in terms of items 1, 2, 3 and 5; we leave open an exploration of item 4, i.e., the communication complexity of the challenger. More generally, we are optimistic that cryptographic tasks may be classified in this taxonomy, based on whether they can be achieved—even using a *non-black-box construction*—based on a class of assumptions in this taxonomy, but not on another.

⁵But our results do not apply to all primitives satisfying these properties; for instance, computational NIZK also satisfies them.

A Note on Random Oracles Let us point out that in the Random Oracle model [?], both of the above-mentioned primitives are easy to construct. Perfect NIZK were constructed in [?] (by relying on the “Fiat-Shamir heuristic” [?] which is sound in this model) and non-interactive non-malleable commitments in [?]. Indeed, many practical protocols rely on the assumption that a “good” hash function behaves like a non-interactive non-malleable commitment, and on non-interactive zero-knowledge arguments constructed by applying the “Fiat-Shamir heuristic” [?] to a three-message perfect zero-knowledge protocol. Our results show that such commonly used sub-protocols cannot be proven secure based on standard hardness assumptions. Note that these results are incomparable to those of e.g., [?, ?] on the “uninstantiability of random oracles”: the results of [?, ?] are stronger in the sense that any instantiation of their scheme with a concrete function can actually be *broken*, whereas we just show that the instantiated scheme cannot be *proven secure* using a Turing reduction based on standard assumptions. On the other hand, the separations of [?, ?] consider “artificial protocols”, whereas the protocols we consider are natural (and commonly used in practice).

1.2 Related Separation Results

There is a large literature on separation results between cryptographic primitives and/or assumptions. We distinguish between two types of results.

Separations for fully black-box constructions The seminal work of Impagliazzo and Rudich [?] provides a framework for proving black-box separations between cryptographic primitives. We stress that this framework considers so-called “fully-black-box constructions” (see [?] for a taxonomy of various black-box separations); that is, the framework considers both black-box *constructions* (i.e., the higher-level primitive only uses the underlying primitive as a black-box), and black-box *reductions*.

Separations for black-box reductions In recent years, new types of black-box separations have emerged. These types of separation apply even to non-black-box constructions, but still only rule out black-box proofs of security: Pass [?] and Pass, Tseng and Venkatasubramanian [?] (relying on the works of Brassard [?] and Akavia et al [?], demonstrating limitations of “NP-hard Cryptography”⁶) demonstrate that under certain (new) complexity theoretic assumptions, various cryptographic task cannot be based on *one-way functions* using a black-box security reduction, even if the protocol uses the one-way function in a non-black-box way. Very recently, two independent works demonstrate similar types of separation bounds, but this time ruling out security reductions to a *general* set of intractability assumptions: Pass [?] demonstrates impossibility of using black-box reductions to prove the security of several primitives (e.g., Schnorr’s identification scheme, commitment scheme secure under weak notions of selective opening, Chaum Blind signatures, etc) based on any “bounded-round” intractability assumption (where the challenger uses an a-priori bounded number of rounds, but is otherwise unbounded). Gentry and Wichs [?] demonstrate (assuming the existence of strong pseudorandom generators) impossibility of using black-box security reductions to prove soundness of “succinct non-interactive arguments” based on any “falsifiable” assumption (where the challenger is computationally bounded). Both of the above-mentioned work fall into the “meta-reduction” paradigm of Boneh and Venkatesan [?], which was earlier used to

⁶See also the results of Feigenbaum and Fortnow [?] and the result of Bogdanov and Trevisan [?] that demonstrate limitations of NP-hard cryptography for *restricted* types of reductions.

prove separations for *restricted* types of reductions (see e.g., [?, ?, ?, ?]).⁷ Our separation results are in the vein of these two works, and follows some of their techniques.

1.3 Proof Overview: Ruling out Perfect NIZK with Adaptive Inputs

Following in an overview of the proof of Theorem 1. Assume there exists a perfect NIZK (P, V) for a *hard-on-the average* language L ; for simplicity, let us further assume that there exists efficient algorithms Sam_L , $Sam_{\bar{L}}$ such that a) with overwhelming probability, Sam_L (resp $Sam_{\bar{L}}$) sample elements $x \in L \cap \{0, 1\}^n$ (resp $x \in L \cap \{0, 1\}^n$), and b) elements sampled by Sam_L and $Sam_{\bar{L}}$ are indistinguishable. Such an L exists based on the existence of one-way functions, which exists by hypothesis.

For simplicity, in this proof overview we focus on the case when the reference string is uniformly random (i.e., we consider only NIZK in the so-called Uniform Reference String (URS) Model). Assume, further, that there exists a Turing reduction R such that R^A breaks the assumption C (with respect to some thresholds t) whenever A breaks adaptive soundness of (P, V) . Following the “meta-reduction” paradigm by Boneh and Venkatesan [?] (which is used in both [?] and [?], and also [?]), we want to use R to directly break C .

More precisely (just as in [?, ?]) we exhibit a particular attacker A to the adaptive soundness of (P, V) and next show how to “emulate” this attacker for R without disturbing R ’s interaction with C . Whereas in [?] the emulation was statistically close (and thus the separation could be applied also to unbounded challengers), in [?] the emulation was only *computationally indistinguishable*, but this still suffices for convincing C as long as C is computationally efficient. We here follow the approach of [?].

Let us turn to describing our attacker A , and next explain how to emulate it. Given a CRS ρ , A first attempts to recover the random coins r used by the simulator S when outputting the CRS ρ ; since the simulation is perfect, such a string r exists (but finding r might require super-polynomial time and so A is not necessarily polynomial-time). (Recall that since we are dealing with adaptive zero-knowledge, the zero-knowledge simulator needs to output a reference string ρ before knowing what statement it needs to simulate a proof of.) Next, A samples a false instance $x \notin L$ that is indistinguishable from a true instance (by hypothesis, this can be done efficiently).⁸ Finally, it runs the simulator S on the random coins r to generate ρ , and next feeds it the instance x , and lets π denote the proof output by S (again this final step is efficient).

Let us argue that the proof π of x is accepted by $V(\rho)$. Towards this, consider a hybrid attacker A' that performs exactly the same steps as A , but instead samples a *true* instance $x \in L$. It follows from the ZK property (combined with the completeness property) that V accepts the proofs output by A' . Now, intuitively, it should follow from the hard-on-the-average property of L that V also accepts the proofs output by A . But there is a problem: recall that A is *not (necessarily) efficient*. However, since it is only the first step of A that is inefficient, we can fix the random string r non-uniformly and still use the remaining steps of A and the efficient verifier V to contradict the hard-on-average property of L , as long as we assume that L is hard-on-average for non-uniform polynomial-time. Note that we here rely on the fact that A is allowed to choose the statement x *after* having seen the reference string ρ (i.e., we rely on A breaking *adaptive* soundness)—this is what allows us to non-uniformly choose r as a function of ρ , *before* sampling $x \in L$.

⁷For instance, these separations results restrict to “algebraic” reductions, or reductions that run the attacker in a “straight-line” fashion.

⁸The careful reader may notice that A actually does not choose the statement x *adaptively*. The fact that the reduction needs to work for attackers A that *may* choose the statement adaptively, and as a consequence must output the reference string ρ before A gets to pick the statement, suffices for us.

Now given this breaker A , let us see an efficient attacker $\tilde{A}$ that is computationally indistinguishable from A . On inputs a URS ρ , $\tilde{A}(\rho)$ simply picks a random true statement x together with a witness w , and next runs the honest prover strategy $P(\rho, x, w)$ to produce a proof π (this strategy is similar to the one used in [?]). It follows by the ZK property that the output of C when communicating with $\tilde{A}$ and A' are indistinguishable, and we can then apply a similar argument as above (but more complicated) to argue that the output of C when communicating with A' and A are indistinguishable, and thus $R^{\tilde{A}}$ breaks C with roughly the same probability as R^A does.

Generalizing Theorem 1 to Exponential-time Challenger Assumptions In case the running-time of the challenger C is super-polynomial in the security parameter k , the above approach seemingly fails: the fact that $\tilde{A}$ generates computationally indistinguishable messages does not suffice to argue that C still accepts in the interaction with $R^{\tilde{A}}$. However, if we assume that the language L is hard-on-the-average for non-uniform subexponential time, then the above approach still works, as long as C is subexponential time; in fact, it rules out also subexponential-time reductions. To deal with also exponential-time challenger assumptions, we proceed as follows. If the *same* assumption C can be used to prove *any* statement in $\mathcal{NP}$ of length polynomial in the security parameter, then if the language L is hard-on-the-average for non-uniform sub-exponential time, it suffices to pick statements x that are sufficiently long (but still of polynomial length) to ensure that $\tilde{A}$ generates messages that are indistinguishable from those sent by $\tilde{A}$, even by C .

1.4 Proof Overview: Ruling out Non-interactive Non-malleable Commitments

Following is an overview of the proof of Theorem 2. Assume there exists a non-interactive commitment scheme Π ; for simplicity of exposition we here focus only on non-interactive, as opposed to two-message, commitments. Assume, further, that there exists a Turing reduction R such that R^A breaks the assumption C (with respect to some thresholds t) whenever A breaks non-malleability of Π . Recall that an attacker A that breaks non-malleability of Π participates in two interactions—one on the “left” acting as a receiver, and one on the “right” acting as a committer. To be successful A needs to choose a different identity for the left and right interactions, and must commit to a value $\tilde{v}$ which is related to the value v it receives a commitment to on the left. Consider a strong attacker A that chooses identity 0 on the left, and identity 1 on the right, and upon receiving a commitment c recovers (using brute force) the unique value v that c is a commitment to (if the value is not unique v is set to $\perp$), and next honestly commits to v on the right. Clearly A breaks non-malleability of Π , and thus R^A also breaks C w.r.t. t .

Let us now see how to efficiently “emulate” A . We simply consider a “trivial” adversary $\tilde{A}$ that picks identity 0 on the left and 1 on the right (just as A), but instead of trying to commit to v on the right, it simply commits to 0 on the right. Now, intuitively, if the reduction R and the challenger C are polynomial-time, then it should follow by the hiding property of Π that $R^{\tilde{A}}$ still breaks C (w.r.t. t). Note, however, that R may be asking its oracle to break non-malleability of *multiple* commitments (rather than a single one as we implicitly assumed above), and since A is not efficiently computable, we need to be a bit careful when doing the hybrid argument. Nevertheless, using a careful ordering of the hybrids, relying on the *non-uniform* security of Π (more precisely, that Π is hiding w.r.t. non-uniform PPT algorithms) we can show that $R^{\tilde{A}}$ still breaks C (w.r.t. t).

Note that the above proof idea applies to a very weak notion of “one-sided” non-malleability, where the attacker always uses identity 0 on the left and identity 1 on the right; Liskov et al [?] call commitments satisfying this weak notion of non-malleability, *mutually independent*. Interest-

ingly, [?] shows a construction of a mutually independent commitment based on the existence of subexponentially-hard one-way permutations. The idea (a.k.a. “complexity-leveraging” [?]) is simple: Let Com_0 be a commitment scheme that is hiding for subexponential time, and let Com_1 be a (polynomial-time) secure commitment scheme whose hiding property can be fully broken (i.e., the committed value can be recovered) in subexponential time. A committer with identity $b \in \{0, 1\}$ shall use Com_b . Now, if a MIM upon receiving a commitment of v using Com_0 is able to output a commitment to a related value $\tilde{v}$ using Com_1 , then we can violate the hiding of Com_0 by simply breaking Com_1 by brute-force. This security reduction, however, is super-polynomial (subexponential) time. A natural question is whether subexponential time/size reductions may be helpful for constructing “full-fledged” (as opposed to one-sided) non-interactive commitments.⁹ We proceed to rule out such reductions (or rather to show that if there exists such a reduction, then the reduction itself must already break the assumption).

Consider a $T(k)$ -sized reduction R , where $T(k)$ is super-polynomial, for basing non-malleability on an efficient challenger assumption C^{10} , and consider the algorithms A and $\tilde{A}$ described above. Note that if R has super-polynomial size, we have no guarantees that $R^{\tilde{A}}$ breaks C even when R^A does; indeed, since hiding of Π is only required to hold for polynomial-sized algorithms, $R^{\tilde{A}}$ ’s success probability may be very different from R^A ’s success probability. But, in this case, intuitively, R itself must be able to break the hiding of commitments using identity 1 (recall that A and $\tilde{A}$ use identity 1 on the right).

So, very roughly, if $R^{\tilde{A}}$ does not already convince C , we can use R (in conjunction with C) to obtain a circuit D that distinguishes, say, commitments to 0^k and 1^k using identity 1.¹¹ We may then use D to construct a man-in-the-middle attacker A' that chooses identity 1 on the *left* and 0 on the right (as opposed to 0 on the left and 1 on the right, as A and $\tilde{A}$ did) to break non-malleability of Π , and finally use R combined with A' to directly break C . So, summarizing, either $R^{\tilde{A}}$ works, or else, we use R in order to construct an MIM A' that breaks non-malleability, and then use $R^{A'}$ to convince C —in essence, we use R “on itself” to convince C .

1.5 Overview of the Paper

We provide some preliminaries and standard definitions in Section 2. We provide definitions of intractability assumptions and black-box reductions in Section 3; this section also contains our taxonomy of intractability assumptions. We formally state and prove our results about NIZK in Section 4. We formally state and prove our results about non-malleable commitments in Section 5.

2 Preliminaries

2.1 Notation

Integer, Strings and Vectors. We denote by $\mathbb{N}$ the set of natural numbers. Unless otherwise specified, when given as an input to an algorithm, a natural number is presented in its binary

⁹Indeed, [?] rely on intuitions similar to those from mutually independent commitments to construct a “full-fledged” non-malleable commitment, but this construction requires multiple communication rounds.

¹⁰The assumption that C is an efficient challenger is only made here to simplify exposition; our actual proof also works when C is $T(k)$ -sized.

¹¹As in the previous proof, to obtain a machine that breaks the hiding of the commitment, we need to rely a polynomial-length non-uniform advice to deal with the above-mentioned inefficiency issue in the hybrid argument; this is why we work with circuits here.

expansion (with no *leading* 0s). For $n \in \mathbb{N}$, we denote by 1^n the unary expansion of n (i.e., the concatenation of n 1's).

Probabilistic notation. We employ the following probabilistic notation from [?]. We focus on probability distributions $X : S \rightarrow R^+$ over finite sets S .

Probabilistic assignments. If D is a probability distribution then “ $x \leftarrow D$ ” denotes the elementary procedure consisting of choosing an element x at random according to D and returning x . If F is a finite set, then the notation “ $x \leftarrow F$ ” denotes the act of choosing x uniformly from F .

Probabilistic experiments. Let p be a predicate and $D_1, D_2, \dots$ probability distributions, then the notation $\Pr [x_1 \leftarrow D_1; x_2 \leftarrow D_2; \dots : p(x_1, x_2, \dots)]$ denotes the probability that $p(x_1, x_2, \dots)$ will be true after the ordered execution of the probabilistic assignments $x_1 \leftarrow D_1; x_2 \leftarrow D_2; \dots$.

New probability distributions. If $D_1, D_2, \dots$ are probability distributions, the notation $\{x \leftarrow D_1; y \leftarrow D_2; \dots : (x, y, \dots)\}$ denotes the new probability distribution over $\{(x, y, \dots)\}$ generated by the ordered execution of the probabilistic assignments $x \leftarrow D_1, y \leftarrow D_2, \dots$.

Probability ensembles. A *probability ensemble* is an infinite sequence of random variables $X = \{X_n\}_{n \in \mathbb{N}}$. We will consider ensembles of the form $X = \{X_n\}_{n \in \mathbb{N}}$ where X_n ranges over strings of length $p(k)$, for some fixed, positive polynomial p .

In order to simplify notation, we sometimes abuse of notation and employ the following “short-cut”: Given a probability distribution X , we let X denote the random variable obtained by selecting $x \leftarrow X$ and outputting x .

Algorithms. We employ the following notation for algorithms.

Probabilistic algorithms. By a probabilistic algorithm we mean a Turing machine that receives an auxiliary random tape as input. If M is a probabilistic algorithm, then for any input x , the notation “ $M_r(x)$ ” denotes the output of the M on input x when receiving r as random tape. We let the notation “ $M(x)$ ” denote the probability distribution over the outputs of M on input x where each bit of the random tape r is selected at random and independently, and then outputting $M_r(x)$.

Oracle algorithms. An oracle algorithm is a machine that gets oracle access to another machine. Given a probabilistic oracle algorithm M and a probabilistic algorithm A , we let $M^A(x)$ denote the probability distribution over the outputs of the oracle algorithm M on input x , when given oracle access to A . We emphasize that if the algorithm A is probabilistic, M does not get to control the randomness of A , and each time A is invoked fresh randomness is used by A . The fact that we do not allow black-box reductions to control the randomness of the oracle they communicate with may seem restrictive. As we shall see later on, however, our result apply even if we consider reductions that work only for *deterministic* attackers using a technique from Goldreich and Krawczyk [?]; see Remark 2 for more details.

Negligible functions. The term “negligible” is used for denoting functions that are asymptotically smaller than the inverse of any fixed polynomial. More precisely, a function $\nu(\cdot)$ from non-negative integers to reals is called *negligible* if for every constant $c > 0$ and all sufficiently large k , it holds that $\nu(k) < k^{-c}$.

Subexponential Function. We say that a function $f(\cdot)$ is *subexponential* if there exists some constant $c < 1$ such that for all sufficiently large k , it holds that $f(k) < 2^{k^c}$.

2.2 Indistinguishability

The following definition of (computational) indistinguishability originates in the seminal paper of Goldwasser and Micali [?].

Let X be a countable set of strings. A *probability ensemble indexed by X* is a sequence of random variables indexed by X . Namely, any element of $A = \{A_x\}_{x \in X}$ is a random variable indexed by X .

Definition 1 (Indistinguishability). *Let $X \subseteq \{0, 1\}^*$. Two ensembles $\{A_{n,x}\}_{n \in N, x \in X}$ and $\{B_{n,x}\}_{n \in N, x \in X}$ are said to be computationally indistinguishable, if for every probabilistic machine D (the “distinguisher”) whose running time is polynomial in its first input, there exists a negligible function $\nu(\cdot)$ so that for every $n \in N, x \in X$:*

$$|\Pr[D(n, x, A_{n,x}) = 1] - \Pr[D(n, x, B_{n,x}) = 1]| < \nu(k)$$

In the above expression, D is simply given a sample from $A_{x,y}$ and $B_{x,y}$, respectively. $\{A_{n,x}\}_{n \in N, x \in X}$ and $\{B_{n,x}\}_{n \in N, x \in X}$ are said to be statistically indistinguishable over X if the above condition holds for all (possibly computationally unbounded) machines D .

2.3 Witness Relations

We recall the standard definition of a witness relation for an $\mathcal{NP}$ language.

Definition 2 (Witness relation). *A witness relation for a language $L \in \mathcal{NP}$ is a binary relation R_L that is polynomially bounded, polynomial time recognizable and characterizes L by $L = \{x : \exists w \text{ s.t. } (x, w) \in R_L\}$.*

We say that w is a *witness* for the membership $x \in L$ if $(x, w) \in R_L$. We will also let $R_L(x)$ denote the set of witnesses for the membership $x \in L$, i.e., $R_L(x) = \{w : (x, w) \in L\}$.

3 Intractability Assumptions and Black-box Reductions

Our definition of an intractability assumption closely follows [?]. Following Naor [?] (see also [?, ?, ?]), we model an intractability assumption as an interaction (or game) between a probabilistic machine C —called the challenger—and an attacker A . Both parties get as input 1^k where k is the security parameter. Any such challenger C , together with a threshold function $t(\cdot)$ intuitively corresponds to the assumption:

For every polynomial-time adversary A , there exists a negligible function μ such that for all $k \in N$, the probability that C outputs 1 after interacting with A is bounded by $t(k) + \mu(k)$.

Hence, we say that A *breaks C w.r.t t with probability p on common input 1^k* if $\Pr[\langle A, C \rangle(1^k) = 1] \geq t(k) + p$.

If the challenger C is polynomial-time in the length of the messages it receives, we say that the assumption is *efficient challenger*; such assumptions are referred to as *falsifiable* assumptions by Naor [?] and Gentry and Wichs [?]. More generally, we refer to an assumption as having a

$T(\cdot, \cdot)$ -time (resp. size) challenger if C can be implemented in time (resp. size) $T(k, \ell)$ on input the security parameter 1^k , and when receiving messages of length ℓ . Hence, (C, t) is an efficient challenger assumption if C is a $T(\cdot, \cdot)$ -assumption where $T(k, \ell)$ is polynomial in both k and ℓ . For simplicity, we here consider either $\text{poly}(k, \ell)$ -time (or size) challengers, or $T(k, \ell) = T(k)$ -time (or size) challengers, where the running-time of the challenger is bounded only as a function of the security parameter.

A Taxonomy of Intractability Assumption We can easily model all “traditional” cryptographic assumptions as efficient challengers C and a threshold t . For instance, the assumption that a particular function f is (strongly) one-way corresponds to the threshold $t(k) = 0$ and the 2-round challenger C that on input 1^k pick a random input x of length k , sends $f(x)$ to the attacker, and finally outputs 1 iff the attacker returns an inverse to $f(x)$. Indistinguishability assumptions (such as, e.g., the decisional Diffie-Hellman problem, or the assumption that a particular function g is a pseudorandom generator) can also easily be modelled as 2-round challengers but now we have the threshold $t(k) = 1/2$. More esoteric assumptions such as the “one-more discrete logarithm assumption” [?, ?], or “adaptive one-way functions” [?], are not efficient-challenger assumptions: for these notions of one-way functions, the attacker gets (some restricted) access to an inversion oracle that cannot be implemented in polynomial-time (or else the function could not be one-way) ; however, these assumption can be modeled as *exponential-time* challenger assumptions (the challenger can now implement the oracle for the attacker).

We may also consider other restricted types of intractability assumptions. For instance, [?] considers challengers C that are computationally unbounded, but for which there exists a polynomial upper bound (in the terms of the security parameter k) on the number of communications rounds by C ; we refer to these assumptions as *bounded-round* intractability assumptions. Another interesting class of assumptions is obtained by further restricting the communication complexity of C ; for instance, we may require that there is a polynomial bound (again in terms of the security parameter k) on the communication complexity of C ; we refer to these assumptions as *bounded-communication intractability assumption*.

Finally, let us note that our notion of an assumption (C, t) does not talk about the complexity of the attacker A that attempts to break the assumption. Rather, we simply let *security reduction* used to based some primitive P on the hardness of (C, t) dictate computational complexity limitations on the attacker. For instance, if we have a polynomial-time (resp. polynomial-size) security reduction to an assumption (C, t) , the security of P is based on the hardness of breaking (C, t) w.r.t. polynomial-time (resp. polynomial-size) attackers A . Similarly, super-polynomial hardness of an assumption can be captured by allowing super-polynomial-time reductions to the assumption.

The above way of modeling assumptions, provides an, in our eyes, natural taxonomy of intractability assumptions based on 1) the *security threshold* t , 2) the number of *communication rounds* used by C , 3) the *computational complexity* of C , 4) the *communication complexity* of C , and 5) the *computational complexity of the security reduction* (specifying whether we consider e.g., polynomial-time hardness or subexponential-time hardness).

Let us end this section by noting that “knowledge-extraction” assumptions (similar to the “knowledge-of-exponent” assumption of [?]) do not fit within our taxonomy of intractability assumption. This is because such assumption actually are not *intractability* assumptions: they are *tractability* assumption! Roughly speaking, such assumption stipulate *feasibility* of efficiently performing some particular task (namely “extraction” of some inputs from every machine that wins some game).

Classifying Cryptographic Tasks As mentioned above, the assumption that a particular function is a one-way function can be formalized as a 2-round efficient-challenger assumption; so can the DDH assumption. But also security properties of more elaborate *cryptographic tasks* can also be formalized as intractability assumptions of the above kind:¹²

- For instance, the notion of “IND-CPA security” of an encryption scheme [?] can be formalized as a 4-round efficient-challenger, bounded-communication, assumption using the standard CPA security game. (Recall that the classic notion of Chosen Plain-text Attack (CPA) security considers an attacker that first receives a public key, next picks two messages m_1 and m_2 , then receives an encryption c of a (a randomly) selected choice of these messages m_b ; the attacker wins if he manages to guess the bit b .)
- On the other hand, the security game of a signature scheme [?] requires using an unbounded-round (and thus also communication) challenger, but still has an efficient challenger. (Recall that the standard notion of security for signatures schemes considers an attacker that receives the verification key for a signature scheme, and then gets oracle access of a “signing” oracle that signs any message of the attacker’s choice: the attacker finally wins the game if it manages to come up with a valid signature on a “new” message m on which it has not queried its signature oracle. To model this assumption (C, t) , we simply have the challenger C implement the signing oracle.)
- Also note that non-malleability of a two-round commitment and adaptive soundness of NIZKs can be formalized as bounded-round, bounded-communication assumptions, but they require an inefficient challenger (to check whether the attack was successful).

We refer to the intractability assumption associated with the (game-based) security definition of an instantiation of a cryptographic task as the *trivial* intractability assumption on which it can be based (for instance, clearly, the security of a particular signature scheme can be based on the intractability assumption that the scheme is secure.) Note, however, that not all cryptographic tasks have even a trivial intractability assumption on which they can be based (e.g., it is not clear whether the zero-knowledge property of a protocol can be formalized as a game-based security property).

Many major results in the cryptographic literature demonstrate “jumps” in our taxonomy: we base the security of some cryptographic tasks on an intractability assumption of a more restrictive nature. For instance,

- When we construct a pseudorandom generator from a particular one-way permutation [?, ?] or even a one-way function [?], we base a primitive (the pseudorandom generator) whose trivial associated intractability has 2-rounds, is efficient challenger, and *threshold* of $1/2$, on an intractability assumption that is 2-round, efficient challenger, but threshold 0.¹³
- When we base the security of a signature scheme on one-way functions [?, ?], or when we base pseudorandom functions on one-way functions [?, ?], we base primitives whose associated trivial intractability assumption are unbounded-round, on a 2-round efficient challenger assumption (with bounded communication).

¹²Specifically, this refers to all cryptographic task with “game-based” definition of security (e.g., one-way functions, signatures etc), as opposed to simulation-based definitions of security (e.g. zero-knowledge).

¹³Any threshold t assumption can always be turned into a threshold $t + \delta$ assumption by having the challenger accept with probability δ , and otherwise proceed as before. The other direction is less clear.

In contrast, some intractability assumptions may be unbridgable at least as far as black-box reductions are concerned. Indeed, as mentioned above, the results of [?, ?] yield some results in this direction, separating unbounded-round and bounded-round assumptions [?] and unbounded-challenger and efficient-challenger assumptions [?]. The results in this paper further elucidate this landscape; among other things, separating unbounded challenger and exponential-time challenger assumptions, and exponential-time and efficient-challenger assumptions.

Black-box Reductions We consider probabilistic polynomial time Turing reductions—i.e., *black-box reductions*. A black-box reduction refers to a probabilistic polynomial-time oracle algorithm. Roughly speaking, a black-box reduction for basing the security of a primitive P on the hardness of an assumption C , is a probabilistic polynomial-time oracle machine R such that whenever the oracle O “breaks” P with respect to the security parameter k , then R^O “breaks” C with respect to a polynomially-related security parameter k' such that k' can be efficiently computed given k . We restrict to the case when $k' = k$. This is without loss of generality because we can always redefine the challenger C so that it on input k acts as if its input actually was k' (since k' can be efficiently computed given k). To formalize this notion, we thus restrict to oracle machines R that on input 1^k always query their oracle on inputs $(1^k, \cdot)$.

Definition 3. *We say that R is a security-parameter preserving black-box reduction if R is an oracle machine such that $R(1^k)$ only queries its oracle with inputs of the form $(1^k, y)$, where $y \in \{0, 1\}^*$.*

A more liberal notion of a black-box reduction allows the reduction R to (on input 1^k) query its oracle on multiple security parameters (that are all polynomially related to k). In our eyes, such a liberal notion is less justified from a practical point of view (and as far as we are aware, cryptographic reductions typically do not rely on such liberal reductions); nevertheless, all our proofs directly apply also for such a notion of black-box reductions.¹⁴ i

4 Security of Perfect Adaptive NIZK

We recall the definition of non-interactive proofs in the Common Reference String (CRS) model. For generality (and since we are proving a lower bound) we allow the CRS to be generated by an arbitrary polynomial-time distribution (as opposed to requiring it to be uniformly random). In the adaptively-sound notion of a non-interactive proof/argument, we require that soundness holds even if the attacker may adaptively pick the statement after having seen the CRS. We consider only proofs/arguments for languages in $\mathcal{NP}$ where the *prover is efficient* when given an $\mathcal{NP}$ -witness.

Definition 4 (Non-Interactive Proofs/Arguments). *A triple of algorithms, $(\mathcal{D}, P, V)$, is called a non-interactive proof system (with non-adaptive soundness) for a language L if the algorithm $\mathcal{D}$ (the “CRS generator”) is probabilistic polynomial-time, the algorithm V (the “verifier”) is a deterministic polynomial-time, and P (the “prover”) is probabilistic polynomial-time, such that the following two conditions hold:*

¹⁴In fact, as remarked in [?], in the context of black-box separations, restricting to reductions that only query its oracle on a single security parameter is actually without loss of generality if we consider primitives with “standard” cryptographic definitions where to break security an attacker only needs to be successful for *infinitely many* input lengths.

- Completeness: *There exists a negligible function μ such for every $x \in L$, every $w \in R_L(x)$ and every $k \in N$,*

$$\Pr [\rho \leftarrow \mathcal{D}(1^k, 1^{|x|}); \pi \leftarrow P(1^k, x, w, \rho) : V(1^k, x, \rho, \pi) = 1] \geq 1 - \mu(k)$$

- Soundness: *For every algorithm B and every polynomial q , there exists a negligible function μ such that for every $k \in N$ and every $x \notin L$ such that $|x| \leq q(k)$*

$$\Pr [\rho \leftarrow \mathcal{D}(1^k, 1^{|x|}); \pi' \leftarrow B(1^k, x, \rho) : V(1^k, x, \rho, \pi') = 1] \leq \mu(k)$$

If additionally the following condition holds, then we call $(\mathcal{D}, P, V)$ an adaptively-sound non-interactive proof system:

- Adaptive Soundness: *For every algorithm B and every polynomial q , there exists a negligible function μ such that for every $k \in N, n \in [q(k)]$*

$$\Pr [\rho \leftarrow \mathcal{D}(1^k, 1^n); (x, \pi') \leftarrow B(1^k, 1^n, \rho) : V(1^k, x, \rho, \pi') = 1 \wedge |x| = n \wedge x \notin L] \leq \mu(k)$$

Finally, if the soundness (resp adaptive soundness) condition only holds w.r.t polynomial-time adversaries B , we call $(\mathcal{D}, P, V)$ a non-interactive argument (resp. an adaptively-sound non-interactive argument)).

Let us turn to defining zero-knowledge. Also here there is a non-adaptive and an adaptive version. In the *non-adaptive* definition of zero-knowledge from [?], there is a single simulator, which, after seeing the statement to be proven, generates both the CRS and the proof at the same time. In the *adaptive* definition from [?], there are two simulators—the first of which must output a string before seeing any theorems. The stronger adaptive definition guarantees zero-knowledge even when the statement to be proved is chosen as a function of the CRS. We here focus only on adaptive zero-knowledge.

Definition 5 (Non-Interactive Zero-Knowledge). *Let $(\mathcal{D}, P, V)$ be an non-interactive proof system for the language L . We say that $(\mathcal{D}, P, V)$ is (adaptively) zero-knowledge if there exists two probabilistic polynomial-time simulators S_1 and S_2 such that for every polynomial q , every non-uniform polynomial-time statement-witness choosing algorithm $c(\cdot, \cdot, \cdot)$ that on input $(1^k, 1^n, \rho)$ outputs a n -bit statement x and witness w such that $(x, w) \in R_L$, the following two ensembles are computationally indistinguishable*

$$\begin{aligned} & \{ \rho \leftarrow \mathcal{D}(1^k, 1^n); x, w \leftarrow c(1^k, 1^n, \rho); \pi \leftarrow P(1^k, x, w, \rho) : (\rho, x, \pi) \}_{k \in N, n \in [q(k)]} \\ & \{ (\rho, \text{aux}) \leftarrow S_1(1^k, 1^n); x, w \leftarrow c(1^k, 1^n, \rho); \pi' \leftarrow S_2(1^k, x, \text{aux}) : (\rho, x, \pi') \}_{k \in N, n \in [q(k)]} \end{aligned}$$

We furthermore say that $(\mathcal{D}, P, V)$ is *perfect* (resp. *statistical*) *zero-knowledge* if the above ensembles are identically distributed (resp. statistically close).

We use the (common) acronym “NIZK” to denote a non-interactive zero-knowledge proof or argument. Feige, Lapidot and Shamir and Bellare and Yung [?, ?] (building on [?]) show that the existence of enhanced trapdoor permutations [?] implies that all of $\mathcal{NP}$ has a adaptively-sound NIZK, but the zero-knowledge property is only computational. As mentioned, Groth, Ostrovsky and Sahai [?] show (under some number theoretic assumptions) that all of $\mathcal{NP}$ has a *perfect* NIZK with *non-adaptive* soundness. More recently, Abe and Fehr [?] present a perfect NIZK for $\mathcal{NP}$ also with adaptive soundness but based the soundness property on a “knowledge-extraction”

assumption (similar to the “knowledge-of-exponent” assumption of [?]) rather than an intractability assumption.

We aim to prove limitations of basing notions of adaptive soundness for perfect or statistical NIZK for $\mathcal{NP}$ on intractability assumptions. Let us first explicitly define what it means to break adaptive soundness of a NIZK.

Definition 6 (Breaking Adaptive Soundness). *We say that A breaks adaptive soundness of $(\mathcal{D}, P, V)$ w.r.t the language L on input lengths $q(\cdot)$ with probability $\epsilon(\cdot)$ if for every $k \in N$,*

$$\Pr \left[\rho \leftarrow \mathcal{D}(1^k, 1^{q(k)}); (x, \pi') \leftarrow A(1^k, \rho) : V(1^k, x, \rho, \pi') \wedge |x| = q(k) \wedge x \notin L = 1 \right] \geq \epsilon(k)$$

Let us turn to defining what it means to base adaptive soundness on an intractability assumption C .

Definition 7 (Basing Adaptive Soundness on the Hardness of C). *We say that R is a black-box reduction for basing adaptive soundness of $(\mathcal{D}, P, V)$ w.r.t. L and input lengths $q(\cdot)$ on the hardness of C w.r.t threshold $t(\cdot)$ if R is a valid black-box reduction and there exists a polynomial $p(\cdot, \cdot)$ such that for every probabilistic machine A that breaks adaptive soundness of $(\mathcal{D}, P, V)$ w.r.t L and inputs lengths $q(\cdot)$ with probability $\epsilon(\cdot)$, for every $k \in N$, R^A breaks C w.r.t t with probability $p(\epsilon(k), 1/k)$ on input 1^k .*

Note that we here require that R^O breaks the assumption C on the security parameter k by querying O on the *same* security parameter k . As previously mentioned, a seemingly more general definition would allow R^O to break C on a polynomially-related security parameter k' (which can be efficiently computed given k), but this extra generality does not buy us anything as we can always re-define C to act as its input was k' when getting the input k .

We are now ready to formally state Theorem 1 from the introduction.

Theorem 3. *Assume the existence of non-uniformly hard one-way functions. Then there exists an $\mathcal{NP}$ -language L such that the following holds. Let $(\mathcal{D}, P, V)$ be a statistical non-interactive adaptively zero-knowledge argument for L , let $q(\cdot)$ be a super-constant polynomial, and let (C, t) be any efficient-challenger assumption. If there exists a black-box reduction R for basing adaptive soundness of $(\mathcal{D}, P, V)$ w.r.t L and input lengths $q(\cdot)$ on the hardness of C w.r.t threshold t , then there exists a probabilistic polynomial-time machine B and a polynomial $p'(\cdot)$ such that for infinitely many $k \in N$, B breaks C w.r.t t with probability $\frac{1}{p'(k)}$ on input 1^k . If furthermore assuming the existence of one-way functions secure against non-uniform subexponential-time algorithms, the above holds even if C is subexponential-time computable.*

Let us remark that as shown in [?, ?, ?], any (even computational) NIZK for a hard-on-the-average language (for non-uniform polynomial-time algorithms), implies the existence of non-uniformly hard one-way functions. So the assumption of one-way functions could actually be relaxed to assume that there exists a hard-on-the average language in $\mathcal{NP}$. Let us also remark that under the assumption of one-way functions secure against non-uniform subexponential-time algorithms, Theorem 3 directly extends also to a super-polynomial-time (SPS) [?] relaxation of the notion of a statistical NIZK, where the simulator may run in subexponential time.

Note that in Theorem 3, we rule out statistical NIZK where adaptive soundness only needs to hold w.r.t. statements of a *particular* (polynomial) length $n = q(k)$.

Our next theorem rules out even *exponential-time* challenger assumptions C if the same assumption C can be used to prove adaptive soundness for *any polynomial length statement* (indeed, as far as we know, in all known NIZK constructions the underlying intractability assumption depends only on the security parameter for the NIZK but not on the length of the statement to be proven).

Theorem 4. Assume the existence of one-way functions secure against non-uniform subexponential-time algorithms. Then there exists an $\mathcal{NP}$ -language L such that the following holds. Let $(\mathcal{D}, P, V)$ be a statistical non-interactive adaptively zero-knowledge argument for L , and let (C, t) be any exponential-time challenger assumption. If for every polynomial $q(\cdot)$, there exists a black-box reduction R for basing adaptive soundness of $(\mathcal{D}, P, V)$ w.r.t L and the input length $q(\cdot)$ on the hardness of C w.r.t threshold t , then there exists a probabilistic polynomial-time machine B and a polynomial $p'(\cdot)$ such that for infinitely many $k \in \mathbb{N}$, B breaks C w.r.t t with probability $\frac{1}{p'(k)}$ on input 1^k .

Note that Theorem 4 is weaker than Theorem 3 in one aspect: we require that the same assumption C can be used to prove *any* polynomial-length statement, whereas in Theorem 3 we rule out NIZK where the underlying hardness assumption may depend also on the length of the statement proved. This additional restriction is necessary: the assumption that a particular NIZK is adaptively sound for statements of length $q(k) = k$ can clearly be stated as an exponential-time challenger assumption: the challenger first sends a CRS to the attacker; upon receiving back a statement x and proof π , the challenger outputs 1 iff π is an accepting proof of x (which can be efficiently checked) and $x \in \{0, 1\}^k$ is a false statement (which can be checked in time $2^{\text{poly}(k)}$). Thus, the challenger can be implemented in some fixed exponential time.

4.1 Proof of Theorem 3

We start by considering a simplified case when the zero-knowledge property is *perfect* and the distribution sampled by $\mathcal{D}$ is uniform over $\{0, 1\}^{\text{poly}(k)}$; that is, we consider perfect NIZK in the so-called “Uniform Reference String” (URS) model.

Ruling out Perfect NIZK in the URS Model Let $g : \{0, 1\}^* \rightarrow \{0, 1\}^*$ be a length-doubling PRG (which can be constructed based on one-way functions [?]). Consider the language $L = \{g(s) | s \in \{0, 1\}^*\}$. Assume there exists a perfect NIZK $(\mathcal{D}, P, V)$ for L in the URS model, where the reference string is of length $\ell(k)$ given the security parameter k , and assume there exists a black-box reduction R for basing adaptive soundness of $(\mathcal{D}, P, V)$ w.r.t L and input length $q(\cdot)$ on the hardness of C w.r.t threshold t . That is, there exists a polynomial $p(\cdot, \cdot)$ such that for every probabilistic machine A that breaks adaptive soundness of $(\mathcal{D}, P, V)$ w.r.t L and inputs lengths $q(\cdot)$ with probability $\epsilon(\cdot)$, for every $k \in \mathbb{N}$, R^A breaks C w.r.t t with probability $p(\epsilon(k), 1/k)$ on input 1^k ; that is,

$$\Pr [\langle R^A, C \rangle(1^k) = 1] \geq t(k) + p(\epsilon(k), 1/k). \quad (1)$$

Our overall proof strategy proceeds in two steps:

- We first construct devise a particular *inefficient* attacker A that breaks adaptive soundness of $(\mathcal{D}, P, V)$ with *overwhelming probability*. By Equation 1, it then follows that there exists some polynomial $\tilde{p}(\cdot)$ such that R^A breaks C with probability $t(k) + \frac{1}{\tilde{p}(k)}$ for infinitely many k .
- We then show how to *indistinguishably* simulate A by an efficient “emulator” $\tilde{A}$ ¹⁵. This concludes that R^A breaks C with probability $t(k) + \frac{1}{\tilde{p}(k)} - \mu(k)$ for infinitely many k , where $\mu(k)$ is a negligible function, and thus proves the theorem.

¹⁵We use the term emulator to describe $\tilde{A}$ to not overload the word simulator.

In our actual proof, the above two steps happen somewhat in parallel. We first construct A and $\tilde{A}$, and then prove that $\tilde{A}$ simulates A , and then use this fact to argue that A is a “good” attacker (that with overwhelming probability provides accepting proofs of false statements).

For simplicity of notation, we assume that $q(k) = 2k$; it is easy to see that the same proof works as long as $q(\cdot)$ is any super-constant polynomial.

Constructing the Attacker A We present a *randomized* attacker A , that on each query uses fresh random coins. (Recall that our notion of a black-box reduction requires the reduction to work even if the attacker if A is probabilistic. As we point out in Remark 2, at the cost of a minor complication, the proof can be adapted to work also if only considering reductions that work as long as the attacker is deterministic. To simplify exposition, we consider also randomized attackers.)

On input 1^k and a reference string ρ , attacker A proceeds as follows (letting $n = q(k) = 2k$):

- **Verify CRS:** A first checks that $|\rho| = \ell(k)$; if not, it return $\perp$.
- **Inverting S_1 :** Otherwise, A *uniformly* picks a random tape r such that $S_1(1^k, 1^n)$ outputs ρ, aux given the random tape r , where aux is some arbitrary string. Since, by our assumption, the simulation is *perfect*, every string $\rho \in \{0, 1\}^{\ell(k)}$ is output by $S_1(1^k, 1^n)$ with positive (and the same) probability, so A will succeed in this task. *Note, however, that this step is not necessarily efficient.*
- **Picking FALSE statement x :** Next, A uniformly picks a string $x \in \{0, 1\}^n$. Note that, except with probability 2^{-k} , it holds that $x \notin L$ (there are 2^{2k} strings, and at most 2^k can be in the range of the PRG g).
- **Generate SIMULATED proof π :** Finally, A runs the simulator $S_2(1^k, 1^n, x, \text{aux})$ to produce the proof π , and return (x, π) .

As noted above, with high probability the statement x picked by A is false. But it remains to argue that the proof π of x output by A is accepting (for the reference string ρ). (Very roughly speaking, the intuition for why π ought to be accepting is that the statement x is indistinguishable from a true statements (in the range of the PRG), and for such statements the simulator ought to produce accepting proofs.) As mentioned above, towards formalizing this intuition, we first present an *efficient* emulator, $\tilde{A}$, for A .

Constructing the Emulator $\tilde{A}$ We construct an efficient “emulator”, $\tilde{A}$ that on input 1^k and a reference string ρ , proceeds as follows:

- **Verify CRS:** Just as A , $\tilde{A}$ first check that $|\rho| = \ell(k)$; if not it simply sends back $\perp$.
- **Picking TRUE statement x :** Next, $\tilde{A}$ uniformly picks a string $s \in \{0, 1\}^k$, and lets $x = g(s)$. Note that by definition $x \in L$.
- **Generate HONEST proof π :** $\tilde{A}$ runs the honest prover algorithm $P(1^k, \rho, x, s)$ to produce the proof π , and outputs (x, π) .

Proving that A is a “good” attacker We now turn to proving that A breaks adaptive soundness of $(\mathcal{D}, P, V)$ with overwhelming probability. In particular, we show the following claim.

Claim 1. *There exists a negligible function μ such that for every $k \in N$,*

$$\Pr [\rho \leftarrow \mathcal{D}(1^k, 1^n); (x, \pi') \leftarrow A(1^k, \rho) : V(1^k, x, \rho, \pi') = 1 \wedge x \notin L] \geq 1 - \mu(k)$$

Proof. Let us first note that by the completeness property of $(\mathcal{D}, P, V)$, we have that there exists a negligible function μ' such that for every $k \in N$,

$$\Pr [\rho \leftarrow \mathcal{D}(1^k, 1^n); (x, \pi') \leftarrow \tilde{A}(1^k, \rho) : V(1^k, x, \rho, \pi') = 1] \geq 1 - \mu'(k) \quad (2)$$

$\tilde{A}$, however, clearly does not break (adaptive) soundness of $(\mathcal{D}, P, V)$ as it picks true statements x . As noted above, A does pick false statements (with overwhelming probability). To prove that it also provides accepting proofs of these statements, consider a hybrid attacker A' that performs exactly the same steps as A , but samples a *true* statement $x \in L$ in exactly the same way as the emulator $\tilde{A}$ (in particular, A' still generates proofs π using the NIZK simulator strategies S_1, S_2 , just as A does).

Note that by construction, the following probability distributions are identical (as the only difference between the experiments generating them is the order in which the randomness of the simulator used to generate proofs is sampled).

$$\{(\rho, \mathbf{aux}) \leftarrow S_1(1^k, 1^n); x, \pi' \leftarrow A'(1^k) : (\rho, x, \pi')\}$$

$$\{(\rho, \mathbf{aux}) \leftarrow S_1(1^k, 1^n); s \leftarrow \{0, 1\}^k, x = g(s); \pi' \leftarrow S_2(1^k, x, \mathbf{aux}) : (\rho, x, \pi')\}$$

By the perfect zero-knowledge property of $(\mathcal{D}, P, V)$ (and considering the statement-witness choosing algorithm $c(1^k, 1^n, \rho)$ that picks $s \leftarrow \{0, 1\}^k$ and outputs $(g(s), s)$), it follows that the latter one (and thus also the former one) is identical to the following one.

$$\{\rho \leftarrow \mathcal{D}(1^k, 1^n); s \leftarrow \{0, 1\}^k, x = g(s); \pi \leftarrow P(1^k, x, s, \rho) : (\rho, x, \pi)\}$$

By the completeness of $(\mathcal{D}, P, V)$, we thus have that A' provides convincing proofs (of true statements) with overwhelming probability: there exists some negligible function μ'' such that

$$\Pr [\rho, \mathbf{aux} \leftarrow S_1(1^k, 1^n); (x, \pi') \leftarrow A'(1^k, \rho) : V(1^k, x, \rho, \pi') = 1] \geq 1 - \mu(k) \quad (3)$$

which in turn (by the Perfect NIKZ property) implies that

$$\Pr [\rho \leftarrow \mathcal{D}(1^k, 1^n); (x, \pi') \leftarrow A'(1^k, \rho) : V(1^k, x, \rho, \pi') = 1] \geq 1 - \mu(k) \quad (4)$$

Finally, as the only difference between A and A' is the choice of the statement x (being truly random in the case of A and pseudorandom in the case of A'), it intuitively should follow from the security of the pseudorandom generator g that A also provides convincing proofs (which combined with equation 2 would prove the claim.). But there is a problem: although, the verification algorithm V is efficient, A and A' are not, so *efficiently* contradicting the pseudo-randomness property of g becomes problematic. This, issue, however, can be dealt with by noting that the only inefficient part of A (and A') is the “inverting S_1 step” which happens *before* A chooses the statement x . We can thus non-uniformly fix these inefficient computations, and rely on the fact that the pseudo-randomness property of g holds even against non-uniform polynomial-time algorithms to conclude that there exists a negligible function μ''' such that for every $k \in N$,

$$\Pr [\rho \leftarrow \mathcal{D}(1^k, 1^n); (x, \pi') \leftarrow \tilde{A}(1^k, \rho) : V(1^k, x, \rho, \pi') = 1] \geq 1 - \mu'''(k) \quad (5)$$

which combined with equation 6 concludes the proof of the claim. $\square$

As consequence of Claim 1 and the fact that R is a good reduction (i.e., Equation 1), there exists some polynomial $\tilde{p}(\cdot)$ such that R^A breaks C with probability $t(k) + \frac{1}{\tilde{p}(k)}$ for infinitely many k ; that is,

$$\Pr \left[\langle R^A, C \rangle(1^k) = 1 \right] \geq t(k) + \frac{1}{\tilde{p}(k)} \quad (6)$$

Proving that $\tilde{A}$ is a good emulator for A To conclude the proof of the theorem, we finally show that $\tilde{A}$ is a “good” emulator for A , even if A is repeatedly invoked by R (in an interaction with C).

Claim 2. *For every efficient C and R , there exists a negligible function μ such that for every $k \in N$,*

$$\left| \Pr \left[\langle R^{\tilde{A}}, C \rangle(1^k) = 1 \right] - \Pr \left[\langle R^A, C \rangle(1^k) = 1 \right] \right| \leq \mu(k).$$

Proof. The proof of the claim is similar to that of Claim 1 but more complicated in order to deal with the fact that R can make many queries to its oracle. The key point is that all these queries are answered *independently* (by both A and $\tilde{A}$), and thus we can perform a hybrid argument which essentially reduces us to the situation in Claim 1. Let us point to have this independence property it is crucial that A generates a “fresh” random statement (independent of all earlier statements) on each query it receives. (If, for instance, A had been stateful and had picked a random statement x *once* (before seeing any CRS), and then provided proofs of the *same* x in every query, then this independence property would not hold. This clarifies why our proof does not extend to rule out also reductions proving *non-adaptive* soundness of $(\mathcal{D}, P, V)$.)

We proceed to a formal proof. Assume for contradiction that the claim is false. That is, there exists a polynomial p' such that for infinitely many $k \in N$,

$$\left| \Pr \left[\langle R^{\tilde{A}}, C \rangle(1^k) = 1 \right] - \Pr \left[\langle R^A, C \rangle(1^k) = 1 \right] \right| \geq \frac{1}{p'(k)}.$$

Let $m(k)$ be an upper-bound on the number of oracle queries by R on input 1^k , and fix a canonical k for which the above happens. Consider a sequence of hybrid experiments $H_0, \dots, H_{m(k)}$, where H_i is defined as the output of $C(1^k)$ after communicating with $R(1^k)$ where the first i oracle queries of R are answered by A , and the remaining ones are answered by the efficient $\tilde{A}$. Note that both A and $\tilde{A}$ are *stateless* and thus these hybrid experiments are well defined. Furthermore, note that

- $H_0 = \langle R^{\tilde{A}}, C \rangle(1^k)$, and
- $H_{m(k)} = \langle R^A, C \rangle(1^k)$

It follows that there exists some j such that

$$\left| \Pr [H_{j+1} = 1] - \Pr [H_j = 1] \right| \geq \frac{1}{m(k)p'(k)}.$$

Define another hybrid H'_j which is identically defined to H_j , but where the statement x_{j+1} in the answer to the $j + 1$ th oracle query is selected as a true statement (just as in H_{j+1}) but we still generate the proof π_{j+1} by using the NIZK simulator (as in H_j).

We now have the following Subclaim:

Subclaim 1. *H'_j and H_j are identically distributed.*

Proof. Note that conditioned on the j th query ρ_j being “invalid” (i.e. outside of $\{0, 1\}^{\ell(k)}$), H'_j and H_j proceed identically the same (the j th query is simply answered $\perp$). Conditioned on $\rho_j \in \{0, 1\}^{\ell(k)}$, on the other hand, it follows by the *perfect* zero-knowledge property of $(\mathcal{D}, P, V)$ (in the same way as in the proof of Claim 1) that the output of H'_j is identically distributed to the output of H_{j+1} ; note that, perfect zero-knowledge is important here to ensure that the zero-knowledge simulation works for *every* reference string $\rho_j \in \{0, 1\}^{\ell(k)}$. $\square$

To reach a contradiction, let us finally argue that the output of H_j is indistinguishable to that of H'_j . Note that up until the point when R receives its $(j+1)$ st statement-proof pair (x_{j+1}, π_{j+1}) back from the oracle, the two experiments proceed identically the same. Thus, if they are distinguishable, there exists some prefix τ of the execution of H_j ¹⁶, up until and including the $j+1$ query of R , such that conditioned on this prefix τ , H_j and H'_j are distinguishable. We may also extend τ to also include the string **aux** picked by A in the $j+1$ query, and conclude that there exists some extension τ' such that

- $|\Pr[H_{j+1}|\tau' = 1] - \Pr[H_j = 1]|\tau'| \geq \frac{1}{m(k)p'(k)}$.
- $H_j|\tau'$ and $H'_j|\tau'$ are efficiently computable (given τ').

The only difference between $H_j|\tau'$ and $H'_j|\tau'$ is the choice of the statement x_{j+1} : in the former is chosen as a truly random string $2k$ -bit string, where in the latter as a $g(s)$, where $s \leftarrow \{0, 1\}^k$. We can thus contradict the pseudorandomness property of g (with respect to non-uniform polynomial-time adversaries). Note in this last step, we crucially rely on the fact queries to A and $\tilde{A}$ are answered *independently*, and the statement x_{j+1} is chosen at random independently of everything that has happened up until query $j+1$. (As mentioned above, in the case of a non-adaptive attacker A , this would not be true—it needs to stick to the same statement x —and as such our proof does not extend to deal with non-adaptive soundness.) $\square$

The proof of the theorem (w.r.t. Perfect NIZK in the URS model) is concluded by combining Equation 6 with Claim 2.

We now show how to extend the proof to deal with statistical NIZK in the “general” CRS model (i.e., the reference string need no longer be uniform). We start by dealing with Perfect NIZK in the CRS model, and then further extend the proof to also deal with Statistical NIZK. In both cases, the issue that needs to be handled is how to deal with reductions that query its oracle on “untypical” CRS: in the case of Perfect NIZK in the CRS model, the CRS can be *invalid* (i.e., not in the range of CRS generation algorithm; in the case of statistical NIZK, we also need to deal with “deviating” CRS that are in the range of the CRS generation algorithm, but still lead to a large statistical gap for the zero-knowledge simulation.

Handling Perfect NIZK in the CRS Model We start by considering Perfect NIZK $(\mathcal{D}, P, V)$ in the “general” CRS model. The attacker A described above is not well defined when executed on input a CRS ρ that is not in the range of $\mathcal{D}$ (in particular, the “inverting S_1 ” may fail). To address this issue we, a) remove the “Verify CRS” step from $\tilde{A}$, and b) modify the “Verify CRS” step of A as follows:

- **Verify CRS:** A first checks that ρ is in the range $\mathcal{D}(1^k, 1^{2k})$; if not, let $x, \pi \leftarrow \tilde{A}(1^k, \rho)$ and return x, π .

¹⁶Technically, the prefix includes the random tape of C and R and all the answers to the first j queries by R .

That is, we now let A check whether the reference string ρ is the range of the CRS generating algorithm, and if not, A outputs a honest proof of a *true* statement.

The proof of Claim 1 remains unchanged with respect to this modified A : in a “real” execution, the CRS ρ is always in the range of $\mathcal{D}$ and thus the new execution mode will never be entered; likewise, the change to $\tilde{A}$ will not have any effect.

The proof of Claim 2 remains essentially unchanged; the only (minor) difference is in the proof of Subclaim 1; previously, we argued that H_j and H'_j are identically distributed condition on ρ_j being invalid (i.e., not in the range of the CRS generation algorithm) by noting that in both H_j and H'_j the answer to the $j + 1$ th query is $\perp$. With our modified A , the answer may no longer be $\perp$, but by construction of the modified A , both experiments proceed in exactly the same way conditioned on ρ_j being invalid.

Handling Statistical NIZK in the CRS Model As mentioned, for the case of statistical NIZK, the above modifications to A (and $\tilde{A}$) may not suffice. The problem is that R may query its oracle on a “deviating” CRS for which the zero-knowledge simulation is statistically far from the distribution of honestly generated proof. However, such “deviating” CRS must be “rare”; we can thus afford to have A fail given such deviating CRS.

More precisely, let $\text{Sim}(1^k, \rho)$ denote the output of the following process:

- **Inverting S_1 :** Pick a random tape r such that $S_1(1^k, 1^{2k})$ outputs ρ, aux given the random tape r , where aux is some arbitrary string.
- **Picking TRUE statement x :** Pick a string $s \in \{0, 1\}^k$, and let $x = g(s)$.
- **Generate SIMULATED proof π :** Run the simulator $S_2(1^k, 1^{2k}, x, \text{aux})$ to produce the proof π , and return (x, π) .

We say that a CRS ρ is α -deviating if following distributions are α -far in statistical distance

$$\begin{aligned} & \{s \leftarrow \{0, 1\}^k, x = g(s); \pi \leftarrow P(1^k, \rho, x, s) : (\rho, x, \pi)\} \\ & \{x, \pi' \leftarrow \text{Sim}(1^k, \rho) : (\rho, x, \pi')\} \end{aligned}$$

Claim 3. *There exists a negligible function μ' such that for every $k \in N$,*

$$\Pr [\rho \leftarrow \mathcal{D}(1^k, 1^n) : \rho \text{ is } \mu'(k)\text{-deviating}] \leq \mu'(k)$$

Proof. Assume for contradiction that there exists some polynomial $p(\cdot)$ such that for infinitely many k , with probability at least $\frac{1}{p(k)}$ over the choice of ρ , the statistical distance between the above distributions (w.r.t. ρ) is at least $\frac{1}{p(k)}$. It follows that the statistical distance between the following distributions is at least $\frac{1}{p(n)^2}$:

$$\begin{aligned} & \{\rho \leftarrow \mathcal{D}(1^k, 1^{2k}); s \leftarrow \{0, 1\}^k; x = g(s); \pi \leftarrow P(1^k, \rho, x, s) : (\rho, x, \pi)\} \\ & \{\rho \leftarrow \mathcal{D}(1^k, 1^{2k}); x, \pi' \leftarrow \text{Sim}(1^k, \rho) : (\rho, x, \pi')\} \end{aligned}$$

By the statistical ZK property of $(\mathcal{D}, P, V)$, there exists some negligible function μ'' such that the latter distribution is $\mu''(k)$ close to the following distribution

$$\{\rho \leftarrow \mathcal{S}_1(1^k, 1^{2k}); x, \pi' \leftarrow \text{Sim}(1^k, \rho) : (\rho, x, \pi')\},$$

Approved for Public Release; Distribution Unlimited.

which in turn is identical to

$$\{\rho, \text{aux} \leftarrow \mathcal{S}_1(1^k, 1^{2k}); s \leftarrow \{0, 1\}^k; x = g(s); \pi \leftarrow \mathcal{S}_2(1^k, x, \text{aux}) : (\rho, x, \pi) \}$$

But this contradicts the statistical ZK property of $(\mathcal{D}, P, V)$. $\square$

Given Claim 3, we now modify the “Verify CRS” step of A to run $\tilde{A}$ not only when the CRS is invalid, but also when the CRS ρ is $\mu'(k)$ -deviating (this may not be efficiently checkable, but it is well defined). Given this new A , the proof of Claim 1 goes through exactly as before if we restrict all experiments to conditioning the CRS on *not* being $\mu'(k)$ -deviating (and relying on the simulation-closeness requirement guaranteed by the definition of a $\mu'(k)$ -deviating CRS instead of appealing perfect zero-knowledge). Since by Claim 3, an honestly generated CRS is $\mu'(k)$ -deviating with negligible probability, it follows by a union bound that Claim 1 still holds with respect to the modified A .

The proof of Claim 2 goes through essentially unchanged: we simply need to weaken Subclaim 1 to the following new subclaim that suffices to conclude Claim 2.

Subclaim 2. H'_j and H_j are $\mu'(k)$ close in statistical distance.

Proof. Conditioned on the $(j+1)$ th query ρ_j being “invalid” (i.e. outside the range of $\mathcal{D}(1^k, 1^{2k})$) or $\mu'(k)$ -deviating, H'_j and H_j proceed identically the same (by construction of A).

Conditioned on ρ_j being in the range of $\mathcal{D}$ and not $\mu'(k)$ -deviating, it follows by definition of $\mu'(k)$ -deviating that the output of H'_j is $\mu'(k)$ -close to the output of H_{j+1} . $\square$

This concludes the proof of Theorem 3.

Ruling out Subexponential-time Challenger Assumptions. If the challenger C is not efficient, then in the above hybrid argument, when switching the statement $x = g(s)$ from being pseudorandom to being truly random, we can no longer directly argue that the probability of C outputting 1 does not change by much. However, if we could ensure that the pseudo-randomness property held against subexponential-size circuits, then the same proof would go through as long as C is a subexponential-size circuit. Thus, if we assume the existence of one-way functions secure against subexponential-size circuit, we prove the theorem also when C is a subexponential-size circuit.

Remark 1. On Adaptive Culpable Soundness *Groth, Ostrovsky and Sahai [?] also define a weaker definition of adaptive soundness which they call adaptive culpable soundness. Roughly speaking, 1) we restrict to languages in $\mathcal{NP} \cap \text{coNP}$, and 2) require a successful attacker to not only prove a false statement, but also provide a $\mathcal{NP}$ proof of the fact that the statement is true.*

Let us remark that simply restricting to languages in $\mathcal{NP} \cap \text{coNP}$ does not suffice for overcoming our lower-bound: assuming the existence of one-way permutations, our impossibility result rules out statistical NIZK with adaptive soundness also for $\mathcal{NP} \cap \text{coNP}$: by the Blum-Micali-Goldreich-Levin [?, ?] construction of a pseudo random generator (see [?]), the existence of one-way permutations implies the existence of a hard-on-the-average language in $\mathcal{NP} \cap \text{coNP}$, which suffices to concluding the theorem.

It is also worthwhile to note why our proof strategy breaks down in case we required the attacker to also provide a witness for false statements (as in the definition of adaptive culpable soundness): a key component in our proof is that the attacker chooses statements that are false but look indistinguishable from true statements (and this is exactly the same method as the one used in [?]). In case the attacker also needs to provide an $\mathcal{NP}$ witness for the statement being false, we can no longer simulate such an attacker by using a true statement.

Remark 2. Deterministic v.s. Randomized Attackers. *In the above proof, we consider a randomized oracle A , and thus only rule out reductions that work for randomized attackers. Following [?], we can easily extend the proof to also rule out reductions that only work for deterministic attackers. First, if we consider a deterministic attacker, we may assume w.l.o.g. that R never asks the same query $(1^k, \rho)$ twice (since we can internally emulate responses to all repeated queries). Next, we define a deterministic attacker A^f that on input $(1^k, \rho)$ selects the random tape of A as $f(1^k, \rho)$ and next executes $A(1^k, \rho)$, as defined above, using the pre-selected random tape. Let $RO : \{0, 1\}^* \rightarrow \{0, 1\}^\infty$ be a uniformly distributed random oracle. Note that A^{RO} acts exactly as the attacker A defined in our proof as long as we never ask the attacker the same query twice. It follows that R 's view when talking to A^{RO} and A are identical. We can thus replace A with A^{RO} in Claim 2. Finally, by an averaging argument, with overwhelming probability over the choice of a random oracle $f \leftarrow RO$, A^f breaks adaptive soundness of $(\mathcal{D}, P, V)$ with overwhelming probability, and for each such “good” choice of f we have that $R^{A^f}(1^k)$ breaks C with advantage $1/p(k)$, where $p(\cdot)$ is a polynomial; thus $R^{A^{RO}}(1^k)$ also breaks C with advantage negligibly close to $\frac{1}{p(k)}$.*

Remark 3. On Non-uniform Reductions *Our current lower bound only considers uniform security reductions R . However, by using the recent techniques of [?] (and relying on the fact that we can rule out reductions that only need to work for deterministic attackers), it readily extends also to rule out non-uniform reductions. We refer the reader to [?] for further details.*

4.2 Proof of Theorem 4

The proof of theorem 4 is essentially identical to the proof of theorem 3. The only obstacle we need to deal with is to ensure that true and false statements are indistinguishable for time 2^k . This is easily achieved if 1) assuming the existence of a pseudorandom generator secure against time 2^{n^ϵ} where n is its output length, and 2) letting A prove statements of length $q(k) = k^{\frac{1}{\epsilon}}$. Note that we here rely on the fact that the same assumption C (with the same security parameter k) can be used to prove adaptive soundness of *any* polynomial length statement.

5 Security of Non-interactive Non-malleable Commitments

Commitment schemes are used to enable a party, known as the *sender*, to commit itself to a value while keeping it secret from the *receiver* (this property is called **hiding**). Furthermore, the commitment is **binding**, and thus in a later stage when the commitment is opened, it is guaranteed that the “opening” can yield only a single value determined in the committing phase. In this work, we consider commitment schemes that are **statistically-binding**, namely while the hiding property only holds against computationally bounded (non-uniform) adversaries, the binding property is required to hold against unbounded adversaries. We refer the reader to [?] for a formal definition. In the rest of the paper, a commitment scheme always refers to a statistically-binding commitment.

Following [?, ?], we consider *tag-based commitment schemes* where, in addition to the security parameter, the committer and the receiver also receive a “tag”—a.k.a. the identity— id as common input.

Let us turn to defining non-malleable commitments. We use a definition essentially due to [?, ?], which follows the earlier definition from [?]. Let $\langle C, R \rangle$ be a tag-based commitment scheme, and let $k \in N$ be a security parameter. Consider a man-in-the-middle adversary A that, on inputs n and z (where z is received as an auxiliary input), participates in one “left” and one “right” interaction. In the left interaction, the man-in-the-middle adversary A interacts with C , receiving

a commitment to the value v using an identity id of length $\ell(k)$ of its choice. In the right interaction A interacts with R attempting to commit a value $\tilde{v}$, again an identity $\tilde{\text{id}}$ of length $\ell(k)$ of its choice. If the right commitments is invalid, or undefined, its value is set to $\perp$; furthermore, if the adversary uses the same identity on the left as on the right, the right interaction is considered invalid. Let $\text{MIM}^\Pi(A, \ell, v, z)$ denote a random variable that describes the value $\tilde{v}$ in the above experiment.¹⁷

Definition 8. A commitment scheme Π is said to be *non-malleable* (with respect to itself) for identities of length $\ell(\cdot)$ if for every PPT man-in-the-middle adversary A the following ensembles are computationally indistinguishable.

$$\begin{aligned} & \{\text{MIM}^\Pi(A, \ell, v, z)\}_{k \in N, v \in \{0,1\}^n, z \in \{0,1\}^*} \\ & \{\text{MIM}^\Pi(A, \ell, v, z)\}_{k \in N, v \in \{0,1\}^n, z \in \{0,1\}^*} \end{aligned}$$

We say that Π is *non-malleable* (with respect to itself) if it is non-malleable for identities of length $\ell(k) = k$.

Our lower bound will apply to non-malleability with respect to identities of length 1, and even if only considering the messages 0^k and 1^k . We now explicitly define what it means to break non-malleability in this particular way.

Definition 9 (Breaking Non-malleability). Let Π be a tag-based commitment scheme. We say that A breaks non-malleability of Π with probability $\mu(\cdot)$ if for every $n \in N$,

$$|\Pr[\text{MIM}^\Pi(A, \ell, 0^k, \perp) = 0^k] - \Pr[\text{MIM}^\Pi(A, \ell, 1^k, \perp) = 0^k]| > \mu(k)$$

where $\ell(k) = 1$. We furthermore say that A breaks one-sided non-malleability of Π with probability $\mu(\cdot)$ if the above holds and A always picks identity 0 for the left interaction and identity 1 for the right one.

We call a commitment *weakly non-malleable* if no attacks of the above kind exist.

Definition 10 (Basing Weak Non-malleability on the Hardness of C). We say that R is a $T(\cdot)$ -black-box reduction for basing weak non-malleability of Π (resp. weak one-sided non-malleability) on the hardness of C w.r.t threshold $t(\cdot)$ if R is a time $T(\cdot)$ a probabilistic oracle machine and there exists a polynomial $p(\cdot, \cdot)$ such that for every probabilistic machine A that breaks non-malleability (resp. one-sided non-malleability) of Π with probability $\mu(\cdot)$, for every $k \in N$, R^A breaks C w.r.t t with probability $p(\mu(k), 1/k)$ on input 1^k .

We are now ready to state our first lower bound for non-malleable commitments.

Theorem 5. Let Π be a two-round tag-based commitment scheme (i.e., the commit-phase consists of a single message from the receiver, followed by a single message from the committer), and let (C, t) be any efficient challenger assumption. If there exists a probabilistic polynomial-time black-box reduction R for basing weak one-sided non-malleability of Π on the hardness of C w.r.t threshold t , then there exists a probabilistic polynomial-time machine B and a polynomial $p'(\cdot)$ such that for infinitely many $k \in N$, B breaks C w.r.t t with probability $\frac{1}{p'(k)}$ on input 1^k .

¹⁷We note that more recent definitions of non-malleability [?, ?] additionally output the view of A in the above experiment. Since we are proving a lower-bound we simply state the weaker definition.

Proof. Consider an unbounded attacker A that chooses identity 0 in the left interaction and 1 in the right and upon receiving a commitment to the value b^k commits to b^k , and otherwise (e.g., if the commitment is invalid) simply commits to 0^k . (Note that implementing A requires super-polynomial time by the hiding property of Π .) Now consider $\tilde{A}$ that picks the same identities, but simply commits to 0^k in the right interaction (no matter what messages it receives on the left). We claim that $R^{\tilde{A}}$ breaks C w.r.t t with inverse polynomial probability for infinitely many k . Assume not; that is, there exists a negligible function μ such that for every $k \in N$, $R^{\tilde{A}}$ breaks C w.r.t t with probability at most $\mu(k)$. Consider a fixed k , and hybrids $H_1, \dots, H_{m(k)}$ where H_i denotes the success probability of R when the first i queries are answered by A and the remaining ones answered by $\tilde{A}$. By the triangle inequality there exists some i such $|H_i - H_{i+1}|$ is inverse polynomial (where the polynomial only depends on R). Since H_i and H_{i+1} proceed identically up until query i , this means there exists some prefix ρ of the executions up until query i such that 1) conditioned on ρ the gap in probability is as high, and 2) the executions of H_i and H_{i+1} are efficiently computable conditioned on this prefix (recall that $\tilde{A}$ is efficient, so once we fix all the answers of A in all the first $i + 1$ queries, the rest of the experiment is efficient). Since the only difference between the hybrids is the commitment received by R in round $i + 1$, we contradict the *non-uniform* hiding of Π . Note that we here rely on the fact that Π only has two rounds (or else hiding may no longer hold since R could be rewinding its oracle), and that C is polynomial-time computable. $\square$

We remark that the above theorem is tight; we cannot hope to rule out also super-polynomial-time reductions for one-sided non-malleability: Liskov et al [?] present a construction of such commitments assuming the existence of one-way permutations with subexponential security.

Let us now turn to ruling out also super polynomial-time reductions for “two-sided” non-malleability (i.e., where the attacker may choose which identity to use).

Theorem 6. *Let Π be a two-round tag-based commitment scheme (i.e., the commit-phase consists of a single message from the receiver, followed by a single message from the committer), and let (C, t) be an $T(\cdot)$ -size challenger intractability assumption. If there exists a $T(\cdot)$ -size randomized black-box reduction R for basing weak non-malleability of Π on the hardness of C w.r.t threshold t , then there exists a $\text{poly}(T(\cdot))$ -sized attacker B and a polynomial $p'(\cdot)$ such that for infinitely many $k \in N$, B breaks C w.r.t t with probability $\frac{1}{p'(k)}$ on input 1^k .*

Proof. Consider the attacker $A, \tilde{A}$ from the proof of Theorem 5. We consider two cases: Either $R^{\tilde{A}}$ breaks C w.r.t t with inverse polynomial probability for infinitely many k (as in the proof of Theorem 5) in which case we are done. Or, there exists a negligible function μ such that for every $k \in N$, $R^{\tilde{A}}$ breaks C w.r.t t with probability at most $\mu(k)$. As shown in the proof of Theorem 5, in this case R together with C and using a polynomial-size advice string may distinguish commitments to 0^k and 1^k with inverse polynomial probability for infinitely many k ; let D denote this “commitment distinguisher” (outputting a bit $b \in \{0, 1\}$). Now consider a third attacker A' that chooses identity 1 on the left and 0 on the right (instead of using 0 on the left and 1 on the right as A and $\tilde{A}$) and next applies D to the commitment it receives on the left in order to decide what value to commit to on the right. More formally, $A'(1^k)$ proceeds as follows:

- A' internally emulates $D(1^n)$, outputs the first output of D (the first message of the commitment¹⁸);

¹⁸Recall that D is a commitment distinguisher for two-round commitment, and thus must first specify the first message of the commitment.

- Upon receiving a commitment c on the left, and a first message r on the right, A' simply forwards c to D ;
- Let b denote the final bit output by D ; output on the right a commitment to b^n (i.e., 0^n if $b = 0$ and 1^n if $b = 1$) using r as a first message.

It directly follows from the fact that D is a good distinguisher that A' breaks non-malleability of Π , and thus there exists a polynomial $p'(\cdot)$ such that for infinitely many $k \in N$, $R^{A'}$ breaks C w.r.t t with probability $\frac{1}{p'(k)}$ on input 1^k . The theorem follows by noting that both R and D are computable in size $\text{poly}(T(\cdot))$. $\square$

A Remark on Deterministic Attackers and Non-uniform Reductions Just as the proof of Theorem 3, the proofs of the above theorem readily extends to rule out reductions that only work with deterministic attackers, and, relying on the techniques from [?], also non-uniform reductions.

6 Acknowledgements

I am extremely grateful to Kai-min Chung and Mohammad Mahmoody for many helpful comments and definitional discussions.

Black-Box Proof of Knowledge of Plaintext and Multiparty Computation with Low Communication Overhead

Steven Myers¹ *, Mona Sergi², and abhi shelat²

¹ Indiana University, Bloomington, IN, USA

² University of Virginia, Charlottesville, VA, USA

Abstract. We present a 2-round protocol to prove knowledge of a plaintext corresponding to a given ciphertext. Our protocol is black-box in the underlying cryptographic primitives and it can be instantiated with almost any fully homomorphic encryption scheme.

Since our protocol is only 2 rounds it cannot be zero-knowledge [GO94]; instead, we prove that our protocol ensures the semantic security of the underlying ciphertext.

To illustrate the merit of this relaxed proof of knowledge property, we use our result to construct a secure multi-party computation protocol for evaluating a function f in the standard model using only *black-box access* to a threshold fully homomorphic encryption scheme. This protocol requires communication that is *independent of $|f|$* ; while Gentry [Gen09a] has previously shown how to construct secure multi-party protocols with similar communication rates, the use of our novel primitive (along with other new techniques) avoids the use of complicated generic white-box techniques (cf. PCP encodings [Gen09a] and generic zero-knowledge proofs [AJLA⁺12,LATV11].)

In this sense, our work demonstrates in principle that *practical* TFHE can lead to reasonably practical secure computation.

Keywords: Fully Homomorphic Encryption, Threshold Encryption, Secure Multi-Party Computation, Communication and Round Complexity, Proof Of Knowledge

1 Introduction

The main technical contribution of this paper is a novel proof of knowledge of a plaintext protocol and its demonstrated use in the construction of a fully black-box multi-party computation protocol with low communication overhead. We briefly describe the motivation behind our work.

* This work, and the authors are sponsored by NSF Grant 0939718, and DARPA and Air Force Research Laboratory under Grant FA8750-11-C0080. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the Defense Advanced Research Projects Agency or the US government.

Communication Secure computation with an honest majority can be accomplished without any cryptographic assumptions, but the best such protocol requires the parties to communicate $|f| \log |f| + d^2 \cdot \text{poly}(n, \log |f|)$ bits [DIK10] and at least d rounds. Here $|f|$ is the size of the function being computed and d is the circuit depth of f , and thus the communication of the protocol is super-linearly related to the number of gates in f . Until recently, even the use of cryptographic assumptions for secure computation required $\text{polylog}(\lambda)$ communication overhead per gate [DIK10] where λ is a security parameter.

Gentry [Gen09a] circumvents per-gate overhead as follows: the honest-but-curious parties use secure multi-party computation to generate an FHE key, each party encrypts its input, and sends the resulting ciphertext and proof to other parties. Once all parties have encryptions of everyone’s inputs, they compute the function of interest locally using the evaluation procedure of the FHE. Finally, to use the resulting ciphertexts as inputs to a secure multi-party computation which computes the decryption of the majority input. In order to be secure against malicious adversaries, the Naor and Nissim compiler [NN01], which makes use of the PCP theorem, can be applied. The use of the PCP theorem in the SMC steps makes the approach impractical, even when presented with a practical FHE scheme.

The motivation behind our work is to remove any use white-box techniques, such as the PCP theorem or generic ZK or NIZK, from the above framework for constructing communication-efficient secure protocols. These techniques have historically been inefficient. In other words, we seek a black-box transformation from TFHE to secure computation.

First Contribution The main technical hurdle in devising a black-box transformation from TFHE to secure computation is to implement the requirement for each player to prove that they “know the plaintext” corresponding to the encrypted input that they have broadcast. This step is essential because it prevents one player from copying (or mauling via the homomorphism) the input of a player who has acted earlier. To handle this step, we show how to construct a two-round black-box proof of knowledge of an encrypted bit for any circuit private FHE scheme using only the encryption scheme. Since our protocol is only two rounds, it is not zero-knowledge (cf. [GO94]), but can provably keep the encrypted bit hidden. Our POK requires that the public-key contain a labeled encryption of 0 and 1, which given all known FHE schemes seems to be a natural modification.³ For traditional FHE schemes, the POK can be used completely black-box, without even the need for the modification.

The basic idea of our proof of knowledge protocol is to first modify the encryption scheme so that the message is encoded using an error-correcting code (ECC) based verifiable secret sharing (VSS) scheme. To encrypt a message we first

³ Since all current schemes contain bit-wise encryptions of their own secret-keys which are random bit strings, and a natural extension of any protocol that provides encryptions of one’s own secret-key can be used to derive a labeled encryption of 0 and 1 which we describe.

generate its secret shares, and encrypt them independently using fresh randomness. A verifier now requests the Prover to reveal the randomness used to encrypt a sub-threshold number of the shares. The verifier then does a consistency check, based on the ECC underlying the scheme, to ensure that the shares were encoded properly. In particular, the error-correcting code we choose offers a property that allows one to check whether local parts of the codeword are error-free. The verifier accepts if everything appears to be properly coded. Since the number of shares revealed is less than the threshold, it does not leak any information about the original message. To show a proof of knowledge property, we argue that an extractor can rewind the Prover and ask for another set of shares to be opened. With high probability, this second transcript provides enough new shares to run the VSS recover algorithm, and recover the original message. The one issue with this approach is that the Prover must reveal the randomness used to encrypt some of the shares. The semantic security of an encryption scheme does not guarantee any security when these random bits are revealed—in particular, the security of the rest of the unopened encryptions are not guaranteed. Instead, we require the encryption scheme to be secure against a selective opening attack (SOA). Fortunately, a result of Hemenway et al. [HLOV11] can be generalized to show that any circuit private homomorphic encryption scheme can be made into an SOA-secure one.

We point out that our proof of knowledge requires the encryption scheme to be homomorphic and circuit-private. Recently, Damgård et al. [DPSZ12] demonstrates a three-round Σ -protocol for knowledge of plaintext, but their protocol *requires* the underlying encryption scheme to also be homomorphic on the random coins used to encrypt. Although many FHE schemes support this property on their random coins, it is certainly not specified in the definition of FHE. In contrast, circuit privacy has been independently defined and seems to be a naturally weaker property.⁴ Moreover, their scheme requires the message space for the FHE to be over $\mathbb{Z}_N$ for N related to the security parameter. While in general, single-bit FHE implies many-bit FHE, we are not aware of any such transformation that *also* preserves the homomorphism over the random coins as required by their protocol. Thus, the requirement for large message space *and* homomorphism over the random coins seem to be extra assumption which our work can avoid (our protocol also works on single-bit FHE). Finally, the Σ -protocol from [DPSZ12] must be compiled into a full zero-knowledge protocol using standard techniques which add round complexity and/or setup assumptions; we show that our two-round protocol with its hidden-bit property suffices for our secure computation protocol.

Second Contribution By combining our result with almost any TFHE scheme, we construct a secure multi-party protocol that avoids *both* per-gate communication

⁴ Even though current schemes achieve circuit privacy via randomness homomorphisms, it is certainly plausible for future constructions to achieve circuit privacy in other ways. Moreover, there do not seem to be any natural ways to transform a circuit private scheme to one with a randomness homomorphism, and thus we feel it is a weaker notion.

complexity *and* white-box techniques such as the PCP theorem or Zero-Knowledge. The communication complexity of our protocol is $O(\lambda^c \cdot n^2)$ where λ is a security parameter and c is a small constant for the TFHE scheme and is thus independent of $|f|$. Our black-box transformation is particularly important because if practical FHE (and TFHE) can be constructed, our transformations will result in practical SFE. Our work is in the standard model and does not require trust assumptions such as the common reference string, a random oracle or public-key setup.

Final Contribution For completeness, we also construct a *threshold* fully homomorphic public-key encryption scheme (TFHE) based on the Approximate GCD problem and the fully homomorphic encryption scheme presented by van Dijk et al. [vDGHV10], and our result was the first to demonstrate the feasibility of directly achieving this threshold primitive for FHE. Since our original eprint submission, [AJLA⁺12] and [LATV11] present more efficient TFHE constructions based on LWE-style assumptions. The point of this construction is to demonstrate feasibility of TFHE under different complexity assumptions.⁵

We present our protocols in the information-theoretic model over secure point-to-point channels, and thus our protocols are secure in the presence of an honest majority. Thus, when used with our transformation, the resulting protocol is also only secure with an honest majority. By using another TFHE that tolerates a dishonest majority, our transformation results in a secure computation protocol that also tolerates the same.

The TFHE scheme provides a constant-round protocol for n players to generate a public-key and distribute private shares of the corresponding secret-key of a fully homomorphic encryption scheme. This step itself is non-trivial since the generation of the public-key for an FHE scheme (that is based on bootstrapping) requires encryption of the secret-key. Later, a majority of players can cooperatively decrypt a ciphertext by running a constant-round protocol on their private shares and a public ciphertext. We also provide methods for distributed encryption and for proving knowledge of an encrypted value.

We note that both our TFHE key generation and decryption protocols are more efficient than generically applying secure function evaluation techniques to the key generation or decryption algorithms of an FHE scheme. For example, with the right set of the parameters, our decryption protocol requires only a constant number of share multiplications, whereas generic techniques would require $O(\text{poly}(\lambda))$ such multiplications. We heavily exploit the linear nature of the operations involved in key generation, encryption and decryption for the particular FHE scheme of van Dijk et al. For key generation and decryption, we develop specific multiparty computation protocols that evaluates an arithmetic circuit using verifiable secret sharing techniques, that is more efficient than the application of generic techniques.

⁵ We note that historically, threshold encryption has been presented where the key-generation algorithm and decryption algorithms are single algorithms, or they are multi-party protocols. We present multi-party protocols.

Comparison With Other FHE-based Secure Computation Protocols Gentry's [Gen09a] secure computation protocol was the first to achieve communication complexity that is independent of $|f|$ by using the PCP theorem in several steps.

Asharov, Jain and Wichs [AJLA⁺12] and López-Alt, Tromer, and Vaikuntanathan [LATV11] have constructed more efficient TFHE schemes based on LWE and the closely related RLWE assumption, which can be reduced to varying degrees to worst-case lattice problems. Their approaches rely on the ability to construct an FHE that also has a homomorphism on the secret-keys, and can also be used to achieve secure computation with communication that is independent of $|f|$. Together, our results demonstrate that the TFHE primitive can be developed from reductions to different classes of hardness assumptions, and therefore TFHE is not simply a consequence of a specific hardness property.

To achieve security against malicious adversaries, López-Alt et al. rely on a common reference string setup so that players can use a NIZK to prove to each other that their keys and their input ciphertexts are well-formed. The use of such NIZK also requires additional hardness assumptions, since (T)FHE is not known to imply NIZK. They can also instantiate their ideas in the standard model by replacing these NIZK proofs with traditional interactive ZK proofs; but in either case, the generic (NI)ZK techniques used require non-blackbox use of the underlying TFHE scheme.⁶ By choosing the CRS model, the authors observed that by using a more expensive simulation-sound NIZK, their protocols can also achieve UC-security. Our protocols only claim standard security, but it has been pointed out to us that it is likely that we can state some of our results as UC in a TFHE-hybrid model.

Asharov et al. use efficient Σ -Protocol constructions to prove well-formedness; these make heavy use of the underlying mathematical structure of the LWE assumption. In order to have efficient NIZK proofs, they must rely on the use of the Random Oracle model, and the use of the Fiat-Shamir heuristic to transform the Σ -protocols into NIZK proofs. In any case, due to the black-box nature of our SMC construction, with simple modifications to the public-key to include labeled ciphertexts representing encryptions of 0 and 1, either of the López-Alt et al. or Asharov et al. TFHE schemes can be plugged in to our construction to achieve security against an arbitrary number of malicious adversaries, with abort. In contrast, with our scheme we are guaranteed output delivery, but need an honest majority of players.

The protocols of Damgård et al. [DPSZ12] and Bendlin et al. [BDOZ11] use a different approach to constructing secure computation protocols from traditional homomorphic encryption. Their schemes rely on the idea from Beaver [Bea91] for circuit randomization. First, they use an offline phase in which the parties use a somewhat homomorphic encryption primitive to create shares of triples (a, b, c) such that $a \cdot b = c$. One triple is required for each multiplication gate in f that is to be evaluated and requires approximately $O(n/s)$ "heavy" cryptographic

⁶ In other words, the encryption algorithm of the TFHE will need to be expressed in terms of a graph-coloring instance (or Hamiltonicity, circuit-sat, etc...). As far as we know, this transformation requires a high-order polynomial overhead.

operations to generate. Next, after such triples have been created, the parties use only information-theoretic methods to evaluate the circuit. This approach results in admirable communication parameters for small circuits (as they have also run practical examples); nonetheless, the approach requires linear communication for each gate in $|f|$, and thus does not achieve our main aim of eliminating this relationship.

Finally, these prior results are all in a model in which n parties are computing, and the protocols can tolerate up to $n - 1$ malicious parties. In contrast, our protocols require an honest majority. The relative incomparability of these models is well understood. In particular, in the model that tolerates up to $n - 1$ malicious adversaries, if any one party deviates from the protocol or fails, then all parties output $\perp$. Alternately, with an honest majority, all parties can output an effective output, as supported by our protocol. For a discussion of the relative merits of the two models, and the impossibility of having protocols that achieve the best of both worlds for general functionalities, see the work of Ishai et al. [IKK⁺11].

In summary, all of these recent works have advantages and disadvantages of their own; our major contribution is the black-box transformation and the independent hardness assumption.

Related work Cramer, Damgård and Nielsen [CDN01], along with Jakobbsson and Juels [JJ00] show how to use threshold cryptography to construct secure multiparty computation protocols. In more detail, we use many ideas from [CDN01] which shows how a homomorphic threshold cryptosystem can be used to achieve general multiparty computation protocols. The notion of using secret-sharing to encode encryptions, as we will do, was first seen in [CDSMW08] and has recently been extended in [GLOV12], although these works use the technique to ensure consistency, and not a proof-of-knowledge, as pursued here.

2 Preliminaries and Notation

A 4-tuple of protocols and algorithms $(\mathbf{Gen}, \mathbf{Enc}, \mathbf{Dec}, \mathbf{Eval})$ is a (t, n) -threshold fully homomorphic encryption scheme if the following hold:

Key Generation An n -party protocol $\mathbf{Gen}$ that at each invocation returns a new public-key $\mathbf{PK}$ and the secret-key $(\mathbf{SK}_1, \dots, \mathbf{SK}_n)$, where $\mathbf{SK}_i$ is the share of the secret-key for Player_i .

Encryption A PPT algorithm $\mathbf{Enc}_{\mathbf{PK}}(m, r)$ that returns the encryption of the plaintext m under the public-key $\mathbf{PK}$ with random coins r .

Decryption There exists a PPT n -party protocol $\mathbf{Dec}(c, \mathbf{SK}_1, \dots, \mathbf{SK}_n)$, which returns the plaintext m using the shares $\mathbf{SK}_i$ held by honest party Player_i , where $c = \mathbf{Enc}(m, r)$ for some random r .

$f_{\mathbf{PK}}$ -homomorphic There exists a PPT algorithm $\mathbf{Eval}$ which given a polynomial f , ciphertexts $c_1 \in \mathbf{Enc}_{\mathbf{PK}}(m_1), \dots, c_k \in \mathbf{Enc}_{\mathbf{PK}}(m_k)$ for some k and a public-key $\mathbf{PK}$, outputs $c \in \mathbf{Enc}(f(m_1, \dots, m_k))$.

The natural notion of chosen plaintext attack indistinguishability needs to be modified in the venue of threshold cryptography to take into account the fact that the adversary has access to shares of the secret-key. The appropriate corresponding and natural definition is given in [CDN01], and full version of our paper [MSas11]. Standard security notions for secure multi-party computation protocols can be used to define the security for the protocols **Gen** and **Dec** in any given instantiation of a TFHE (e.g., we can consider security in the real/ideal standalone paradigm, the UC framework, etc..)

Next, we present the notion of bootstrapping a ciphertext. Gentry developed the notion of Bootstrapping to reduce noise in a somewhat fully homomorphic encryption scheme, in order to achieve a fully homomorphic scheme. In contrast, we assume the existence of an FHE and simply use it to reduce noise produced in ciphertexts generated in our selective opening attack secure scheme that we introduce later.

Definition 1. (*Bootstrapping a Ciphertext*) For a FHE scheme $\Pi = (G, E, D, \mathbf{Eval})$ and the security parameter k , let D_Π be Π 's decryption circuit, which takes a secret-key and s ciphertext as input. Given a ciphertext C encrypted with respect to a public-key PK and secret-key $SK = (SK_1, \dots, SK_\ell)$ we require that PK contains a bit-wise encryption of SK , denoted $s_1, \dots, s_\ell$ where $s_i = E(PK, SK_i)$. Let $(C_1, \dots, C_n)$ denote the bits of C , and generate $c_i = E(PK, C_i)$. We say that the value $C^\dagger = \mathbf{Eval}(PK, D_\Pi, s_1, \dots, s_\ell, c_1, \dots, c_n)$ (which homomorphically evaluates $D(SK, C)$) is the result of bootstrapping C .

2.1 Selective Opening Security

In our construction, we will need to refer to encryption schemes where messages that are encrypted remain secure, even after the randomness used to encrypt related messages is revealed. This notion of security is called Selective Opening Security.

Definition 2 (IND-SO-SEC Encryption Security). A public-key encryption scheme $\Pi = (G, E, D)$ is Indistinguishable Selective Opening secure if, for any message sampler M that supports efficient conditional resampling, and any ppt adversary $A = (A_1, A_2)$ there exists a negligible function μ such that for all sufficiently large k :

$$\left| \Pr[A_\Pi^{\text{Ind-SO-Real}}(1^k) = 1] - \Pr[A_\Pi^{\text{Ind-SO-Ideal}}(1^k) = 1] \right| \leq \mu(k).$$

A message sampler M is a PPT algorithm that outputs a vector $\mathbf{m}$ of n messages from a given distribution. It is an efficient conditional resampler if, when given two auxiliary inputs, a set of indices $I \subseteq [n]$, and a vector of messages $\mathbf{m} = (m_1, \dots, m_n)$, M samples another vector $\mathbf{m}' = (m'_1, \dots, m'_n)$ conditioned on $m_i = m'_i$ for each $i \in I$. We define the experiments Ind-SO-Real and Ind-SO-Ideal as follows.

$\text{Ind-SO-Real}(1^k, A)$ $(PK, SK) \leftarrow G(1^k)$ $\mathbf{m} = (m_1, \dots, m_n) \leftarrow M$ $r_1, \dots, r_n \leftarrow R$ $(I, \sigma) \leftarrow A_1(PK, E_{PK}(m_1, r_1), \dots, E_{PK}(m_n, r_n))$ $\text{Output } A_2(\sigma, (m_i, r_i)_{i \in I}, \mathbf{m})$
$\text{Ind-SO-Ideal}(1^k, A)$ $(PK, SK) \leftarrow G(1^k)$ $\mathbf{m} = (m_1, \dots, m_n) \leftarrow M$ $(I, \sigma) \leftarrow A_1(PK, E_{PK}(m_1, r_1), \dots, E_{PK}(m_n, r_n))$ $\mathbf{m}' = (m'_1, \dots, m'_n) \leftarrow M_{ I, m[I]}$ $\text{Output } A_2(\sigma, (m_i, r_i)_{i \in I}, \mathbf{m}')$

2.2 Circuit Privacy

Definition 3. (*(Statistical) Circuit Private Homomorphic Encryption*). A homomorphic encryption scheme $\varepsilon = (\mathbf{Gen}, \mathbf{Enc}, \mathbf{Dec})$ is circuit-private for circuits in a set C_ε if, for any key pair (PK, SK) output by $\mathbf{Gen}(\lambda)$, any circuit $C \in C_\varepsilon$, and any fixed ciphertext $\psi = \langle \psi_1, \dots, \psi_t \rangle$ that are in the image of $\mathbf{Enc}$ for plaintexts $\pi_1, \dots, \pi_t$, the following distributions (over the random coins in $\mathbf{Enc}, \mathbf{Eval}$) are (statistically) indistinguishable:

$$\mathbf{Enc}_{PK}(C(\pi_1, \dots, \pi_t)) \approx \mathbf{Eval}_{PK}(C, \psi)$$

In the original schemes first presented by both Dijk et al. [vDGHV10] and Gentry [Gen09a], the initial evaluation functions are deterministic and not circuit-private. In order to overcome this problem, both works introduce a method for adding random noise to encryptions, whether they are output from $\mathbf{Eval}$ or $\mathbf{Enc}$, and thus in some sense rerandomizing them. This is done by adding an ‘encryption’ of 0 to the ciphertext in question, but where the ‘encryption’ has significantly more noise than would be generated by either the legitimate encryption or evaluation process. Specifically, they introduce ppt algorithms labeled $\text{CircuitPrivacy} : \mathcal{C}_b \rightarrow \mathcal{C}'_b$, where $\mathcal{C}$ consists of all the ciphertexts that are output from $\mathbf{Enc}_{PK}(b)$ or a call to $\mathbf{Eval}$ with an encrypted output bit of b . It is the case that for any b and any $c_{b,0}, c_{b,1} \in \mathcal{C}_b$.

$$\text{CircuitPrivacy}(c_{b,0}) \approx_s \text{CircuitPrivacy}(c_{b,1}).$$

3 Proof of Knowledge of an Encryption

As noted in the Introduction, the method of Cramer, Damgård, and Nielsen [CDN01] requires an honest-verifier zero-knowledge proof of knowledge of encrypted values for the threshold schemes that they employ. We provide a weaker 2-round solution to that requirement, which alas, is not zero-knowledge, but also does not

release any information about the bit being discussed (we formalize this below). Moreover, our construction is black-box in the underlying circuit-private FHE scheme.

We construct this proof through a two-step process. At a high-level, instead of encrypting a bit b , we use a specific $(n, n/2 + 2)$ verifiable secret sharing scheme to generate n shares of b and encrypt those shares.⁷ In order to give a proof of knowledge of the encryption of b , we allow a verifier to select $n/2 + 1$ of the encryptions of shares of b , and then direct the Prover to reveal the randomness used to encrypt those shares. To extract the bit, our extractor rewinds the proof and selects an alternate $n/2 + 1$ shares, so that with high probability, it can use $n/2 + 2$ shares to reconstruct b , and only b due to the verifiability of the secret sharing scheme. The problem with this approach is that revealing the randomness for an encryption raises selective decommitment issues. We use techniques from Hemenway et al. [HLOV11] to construct a bit-wise Indistinguishable Selective-Opening Secure encryption scheme from our threshold fully-homomorphic scheme. We can then use it to bitwise encrypt the VSS shares.

We note that the encryptions of the shares under the bit-wise Indistinguishable Selective-Opening Secure scheme, is not itself a homomorphic encryption scheme. For example, we cannot multiply directly two sets of shares encoding b_0 and b_1 and expect the result to encode $b_0 \cdot b_1$. However, the individual encrypted bits are still properly encoded ciphertexts under the FHE scheme that have a circuit-privacy evaluation function applied to them. Intuitively, therefore, we can homomorphically evaluate the reveal function of the secret sharing scheme to get a single encryption representing the reconstituted bit. This encryption can then be used to homomorphically evaluate the function as in Cramer et al. [CDN01]. There is however a snag: in principle, once the circuit-privacy function has been applied to a ciphertext, it may no longer be able to have homomorphic operations applied to it, as this is not guaranteed by the definition.⁸ However, this problem is easily surmounted by applying Gentry’s bootstrapping technique (cf. Defn 1) to re-encode the selective-opening secure schemes into ciphertexts which can have homomorphic operations applied to them, and thus the VSS’s reveal algorithm can be applied to the individual bits of the shares, resulting in ciphertext of the encoded bit, which is in the ciphertext space of the TFHE scheme.

Using FHE to construct a Selective Opening Encryption Scheme Hemenway et al. [HLOV11] show how any re-randomizable encryption scheme can be used to construct a natural lossy encryption scheme and thus, by the result of Bellare et al. [BHY09], is secure against indistinguishable selective opening attacks.

Since the Hemenway and Ostrovsky construction relies on re-randomization, they suggest that the distribution of a “fresh” encryption of a message should be

⁷ We use a verifiable secret sharing scheme with a $n/2 + 2$ threshold to simplify the proof of the VSS, thus $|T| = n/2 + 1$ is chosen to be right under the threshold of the VSS, as one might expect.

⁸ Further, in practice, with known schemes, these ciphertexts have too much noise in them to allow further homomorphic operations without sacrificing decryption correctness.

statistically close to a rerandomization of a fixed message. They point out that all homomorphic encryption schemes up to that point achieved this property by adding an encryption of 0 to the current message. While this property was true of all schemes at the time, it is not actually true of the known fully homomorphic encryption schemes, because each time we add an encrypted message to another we increase the amount of noise that is embedded in the ciphertexts, and thus fresh encryptions have less noise than encryptions that have had operations (such as addition) applied to them. Fortunately, the property they state is overly strong, and a simple observation shows that for their construction to go through they only require that the distributions

$$\{r \leftarrow R : E_{pk}(0, r) \boxplus E_{pk}(m, r_0)\} \approx_s \{r \leftarrow R : E_{pk}(0, r) \boxplus E_{pk}(m, r_1)\},$$

for all public-keys pk , messages m and random strings r_0 and r_1 where $\boxplus$ is the homomorphic addition operation. However, it is simple to see that even these two distributions are not statistically close for the fully homomorphic encryption schemes that have been proposed. Fortunately, both schemes under consideration have rerandomization functions built to ensure *Circuit-Privacy*, as is defined in [Gen09b] and Def. 3.

Construction of a SOA from Lossy We generate a public-key for the Lossy scheme by generating a traditional public-key and secret-key for the TFHE, and then we augment the public-key with two labeled ciphertexts c_0 and c_1 , representing encryptions of 0 and 1. Now, to encrypt a bit b , we take c_b , and rerandomize it using the circuit-privacy function (In comparison, Hemenway and Ostrovsky add an encryption of the bit 0). Decryption works as it does in the FHE scheme. The lossy key generator simply has c_1 represent an encryption of 0 instead of 1. By the IND-CPA security of the TFHE scheme, the keys are indistinguishable. The scheme is formally described below.

Key Generation $G'(1^k, b), b \in \{\text{INJ}, \text{LOSSY}\}$: Let $(\text{PK}, \text{SK}) \leftarrow G(1^k)$, $c_0 \leftarrow E(\text{PK}, 0)$, $c_1 \leftarrow E(\text{PK}, 1)$ and $c'_1 \leftarrow E(\text{PK}, 0)$. If $b = \text{INJ}$ Output $\text{PK}' = (pk, c_0, c_1)$ and $\text{SK}' = \text{SK}$, else when $b = \text{LOSSY}$ output $\text{PK}' = (\text{PK}, c_0, c'_1)$ and $\text{SK}' = \text{SK}$.

Encryption $E'(\text{PK}' = (\text{PK}, c_0, c_1), b)$: Output $\text{ReRand}(c_b)$.

Decryption $D'(\text{SK}, c)$: Output $D(\text{SK}, c)$.

Theorem 1. *If (G, E, D) is a circuit-private FHE, then the blackbox construction (G', E', D') described above is an IND-SO-SEC secure encryption scheme.*

Proof. Follows from [HLOV11] and [BHY09].

Modifying the SOA-secure Encryption Scheme to Support POKs Again, in order to be able to provide a proof of knowledge that a party has knowledge of the value encrypted, we need to provide a POK. We will show a 2-round public-coin proof of knowledge of the encrypted bit based on any selective opening secure scheme. The protocol is neither zero-knowledge nor witness indistinguishable, but does

maintain secrecy of the encrypted bit. First, we encrypt bits using the following protocol. Let $\Pi' = (G', E', D')$ be the selective-opening attack secure scheme described in Thm. 1. We construct a new encryption scheme $\hat{\Pi} = (\hat{G}, \hat{E}, \hat{D})$ to encode bits as follows. We define $\hat{G} = G'$, and present the algorithms for $\hat{E}$ and $\hat{D}$ below. Refer to a full version of our work [MSas11] for the standard definitions of the Verifiable Secret Sharing algorithms.

$\hat{E}(\text{PK}, b, r)$ $(s_1, \dots, s_n) \leftarrow \text{VSShare}_{(n, n/2+2)}(b)$ Let M be the $n \times n$ matrix representation of shares $(s_1, \dots, s_n)$ $c_{i,j} = E'(\text{PK}, M_{i,j}, r_{i,j})$ Output $\mathbf{C} = \{c_{i,j}\}_{i,j \in n}$	$\hat{D}(\text{SK}, \mathbf{C})$ $M = \{M_{i,j}\}_{i,j \in [n]} \leftarrow D'(\text{SK}, \mathbf{C})$ Let $(s_1, \dots, s_n)$ be the shares corresponding to matrix M . $T' = \{t \mid 1 \leq t \leq n \text{ share } s_t$ is $n/2 + 2$ -consistent} If $ T' < n/2 + 2$ output $\perp$. Let $T \subseteq T'$ s.t. $ T = n/2 + 2$. Output $\text{VSReveal}_{(n, n/2+2)}(\{s_{t_i}\})_{t_i \in T}$

Hidden Bit POK Given a ciphertext $\mathbf{C} = \{c_{i,j}\}_{i,j \in n}$ output by encryption algorithm $\hat{E}$ and the random coins r used to generate it, we show how to perform a two-round proof of knowledge of the encrypted bit $\hat{D}(\text{SK}, \mathbf{C})$. P will prove that it has knowledge of the underlying shares of the verifiable secret-sharing scheme that have been encrypted. In order to do this, the verifier sends a random challenge of indices $T \subset [n]$, where $|T| = n/2 + 1$. The encryptor then decommits to these encryptions by providing the random-bits used to encrypt each share of the bit. If each bit decommits successfully, and the result is $n/2 + 1$ valid shares to the VSS, then the verifier accepts.

Prover($\text{PK}, \mathbf{C} = \{c_{i,j}\}_{i,j \in [n]}$ $= \hat{E}(\text{PK}, b, r), M, r$) Let $c_{i,j} = E'(\text{PK}, M_{i,j}, r_{i,j})$	$\xleftarrow{T}$ $\{M_{i,x}, r_{i,x}, M_{x,i}, r_{x,i}\}_{i \in T}$ $\xrightarrow{x \in [n]}$	Verifier($\text{PK}, \mathbf{C} = \{c_{i,j}\}_{i,j \in [n]}$) $T \leftarrow \{S \mid S \subset [n] \wedge S = \frac{n}{2} + 1\}$ if $\exists i, j: c_{i,j} \neq E'(\text{PK}, M_{i,j}, r_{i,j})$, output $\perp$. Output 1.

$\text{Extractor}(\mathbf{C}, \text{PK}, U_1 = \{M_{i,x}, r_{i,x}, M_{x,i}, r_{x,i}\}_{i \in T_1, x \in [n]}, U_2 = \{M_{i,x}, r_{i,x}, M_{x,i}, r_{x,i}\}_{i \in T_2, x \in [n]})$ Let $T = T_1 \cup T_2, U = U_1 \cup U_2$ If $ T < n/2$ output $\perp$. If $\exists i \in T, x \in [n]$ s.t. $E'(\text{PK}, M_{i,x}, r_{i,x}) \neq c_{i,x}$ or $E'(\text{PK}, M_{x,i}, r_{x,i}) \neq c_{x,i}$ output $\perp$. For each $i \in T$ reconstruct its corresponding share s_i . Output $\text{VSReveal}_{(n, n/2+2)}(s_{r_1}, \dots, s_{r_{\frac{n}{2}}})$, where $r_1, \dots, r_{\frac{n}{2}}$ are the smallest indices in T .

Completeness Follows by inspection.

Extractability (Soundness) Soundness follows from an extractor.

Theorem 2. For all sufficiently large n , for all $d > 0$, for all $(SK, PK) \leftarrow \hat{G}$, for all ‘ciphertext’ inputs C , and provers P' , if $(P', V)(C = \{c_{i,j}\}_{i,j \in [n]}, PK)$ accepts with probability $1/n^d$, then there exists a probabilistic polynomial time extractor that, with all but negligible probability, outputs a set of decommitments to all ciphertexts for a given set of indices $L = \{\ell_1, \dots, \ell_{n/2+2}\} \subseteq [n]$ that constitute shares $S = \{s_{\ell_1}, \dots, s_{\ell_{n/2+2}}\}$ such that $VSReveal_{(n, n/2+2)}(s_{\ell_1}, \dots, s_{\ell_{n/2+2}}) = \hat{D}(SK, C)$.

Definition 4. We say an $n \times n$ matrix representation of shares has t -consistent indices if there is a set S of size t such that for each $i \in S$, each row i and column i is $n/2 + 2$ consistent.

Proof. Given the ability to rewind the prover-verifier protocol, we can extract the encrypted bit by recovering enough shares of the VSS scheme. We continue to execute the prover/verifier protocol until we get two distinct separate accepting proofs. It is a simple observation that except with exponentially small probability, we will succeed in $O(n^{d+1})$ rewinds. Let (T_1, U_1) and (T_2, U_2) be the flows in the first and second accepting proofs, respectively. By the security of the commitment scheme (Here we are using our encryption scheme as a simple commitment scheme), the probability that there is a ciphertext $c_{i,j}$ that is ever decommitted to in two distinct fashions is negligible.

We feed these inputs in to *Extractor*. If there is not a valid encryption of a bit (fewer than $n/2 + 2$ committed and consistent shares), then by Lemma 1, the probability that the verifier outputs anything other than $\perp$ is less than $\frac{1}{\binom{n}{n/2+2}}$ which grows exponentially small.

Given the decommitments of the shares $\{s_i\}_{i \in T_i}$ for different randomly chosen set of indices T_1 and T_2 , note these sets are not the same by selection, and therefore there is no chance that $\perp$ is output by the extractor. Next the extractor executes a $VSReveal_{(n, n/2+2)}$ command. However, this is not necessarily over the same shares as would be revealed in a legitimate decryption. We need to ensure that no matter which of the rewound and newly played legitimate traces we receive, we are going to reveal the same encrypted bit, with all but negligible probability. That is, we need to ensure that $VSReveal_{(n, n/2+2)}(s_{r_1}, \dots, s_{r_{n/2}}) = VSReveal_{(n, n/2+2)}(s_1, \dots, s_{n/2})$. This is the case, as shown in Lemma 2 because of the verifiable properties of the secret sharing scheme ensures that even in the case of a corrupted dealer (improper ciphertext encoding of shares) then all honest players will reveal the same value, with all but negligible probability. Therefore, with all but negligible probability we have that the extractor outputs the same value as $D(SK, c)$.

Lemma 1. Let M be an $n \times n$ matrix with at most $n/2 + 1$ consistent indices. The probability that any $n/2 + 1$ randomly selected indices (without replacement) choose a set of $n/2 + 1$ consistent indices is no more than

$$1/\binom{n}{n/2+1}.$$

Proof. There can be at most 1 set of size $(n/2 + 1)$ that is $(n/2 + 1)$ consistent in an $n \times n$ matrix. The lemma follows by computing the probability of choosing this one set from a set of n objects.

Lemma 2. *Let M be $n \times n$ matrix representation of shares. Let $S, T \subseteq [n]$, $|S| = |T| = n/2 + 2$, $S \neq T$, and the rows $R_S = \{r_i\}_{i \in S}$, $R_T = \{r_i\}_{i \in T}$ and columns $C_S = \{c_i\}_{i \in S}$, $C_T = \{c_i\}_{i \in T}$ are all $n/2 + 2$ -consistent. Let $s = (s_1, \dots, s_{n/2+2})$ and $t = (t_1, \dots, t_{n/2+2})$ be the shares drawn from M corresponding to the sets of indices S and T respectively. Then*

$$VSReveal_{(n, n/2+2)}(s_1, \dots, s_{n/2+1}) = VSReveal_{(n, n/2+2)}(t_1, \dots, t_{n/2+1})$$

Proof. Note that $VSReveal_{(n, n/2+2)}$ will never output $\perp$ under our conditions, so all that we need do is show that f will interpolate to the same value in both cases.

We know that the rows $R_T = \{r_i\}_{i \in T}$ and columns $C_R = \{c_i\}_{i \in T}$ are all $(n/2 + 2)$ -consistent. Choose any $j \in S \setminus T$. Let $T = \{t_1, \dots, t_{n/2+2}\}$. Consider $c_j = (c_{1,j}, c_{2,j}, \dots, c_{n,j})^T$. Since c_j is $n/2 + 2$ -consistent, the points $(c_{t_1,j}, t_1), \dots, (c_{t_{n/2+2},j}, t_{n/2+2})$, interpolate to a unique univariate degree $n/2 + 1$ polynomial (i.e. $f(x, j)$). This defines $(c_{1,j}, c_{2,j}, \dots, c_{n,j})^T$, so the column j must be consistent with T . Since the j th column was an arbitrary column in S different from those in T , all such columns must be consistent with the rows defined by T . A symmetric argument shows that rows selected by S must be consistent with the columns selected by T . Therefore, both sets are consistent in that they define the same polynomials. Therefore, interpolation in $VSReveal_{(n, n/2+2)}$ will result in the same output.

Hidden Bit We show that no efficient cheating verifier can predict the bit b , when given $C = \hat{E}(\text{PK}, b, r)$ as a theorem for which we are engaging in a POK.

Theorem 3. *For every P.P.T. adversary $A = (A_1, A_2)$, there exists a negligible function μ such that $\Pr[HB_A(1^k) = 1] \leq 1/2 + \mu(k)$, where HB_A is defined below:*

$HB_A(1^k)$

$(PK, SK) \leftarrow \hat{G}(1^k)$

$b \in \{0, 1\}$

$C = \{c_{i,j}\}_{i,j \in [n]} = \hat{E}(PK, b)$ where $c_{i,j} = E'(PK, M_{i,j}, r_{i,j})$ are SOA-sec.

$(T, \sigma) \leftarrow A_1(PK, C)$ where $T \subset [n]$, $|T| = n/2 + 1$.

$b' \leftarrow A_2(\sigma, (M_{i,j}, r_{i,j})_{i,j \in T})$

Output 1 iff $b = b'$

Proof. This follows directly from the IND-SO-SEC security of $\Pi' = (G', E', D')$. Suppose an adversary $A = (A_1, A_2)$ breaks the hidden bit security of the protocol. That is for some $d > 0$ and infinitely many k : $\Pr[HB_A(1^k) = 1] \geq 1/2 + 1/k^d$. We use it to build an adversary $B = (B_1, B_2)$ and message selector M that breaks the IND-SO-SEC security (cf. Defn. in [BHY09] or [MSas11]) of $\Pi' =$

(G', E', D') . The message selector M chooses a random bit b , let $(s_1, \dots, s_n) \leftarrow VSShare_{(n, n/2+2)}(b)$, and let $\mathbf{M}$ be the $n \times n$ matrix that represents the shares $(s_1, \dots, s_n)$ according to the ECC representation of the VSS. Output $\mathbf{M}$.

The adversary $B_1 \left(\text{PK}, (E(\text{PK}, \mathbf{M}_{i,j}, \mathbf{r}_{i,j}))_{i,j \in n} \right)$ for the IND-SO-SEC experiment simulates $(T, \sigma) \leftarrow A_1(\text{PK}, C = (E(\text{PK}, \mathbf{M}_{i,j}, \mathbf{r}_{i,j})))$, and outputs $I = \{(i, j) | i, j \in n, i \in T \text{ or } j \in T\}$ and $\sigma' = (T, \sigma)$. Recall by the definition of A_1 , $|T| = n/2 + 1$.

The conditional message selector $M_{I, \mathbf{m}[I]}$ from the SOA security definition finds a random bi-variate polynomial of degree $n/2 + 1$ in each variable over the field F such that $f(0, 0) \in \{0, 1\}$ and for each $(i, j) \in I$, it holds that $f(i, j) = \mathbf{M}_{i,j}$. Since $|T| = n/2 + 1$, and thus we have effectively release $n/2 + 1$ shares for a VSS scheme that requires $n/2 + 2$ for reconstruction, the information secrecy property of the VSS guarantees there are exactly the same number of such selections for the case $f(0, 0) = 0$ and $f(0, 0) = 1$. $M_{I, \mathbf{m}[I]}$ outputs $\{f(i, j)\}_{1 \leq i, j \leq n}$.

The adversary $B_2(\sigma, (M_{i,j}, r_{i,j})_{(i,j) \in I}, \mathbf{M}^*)$ computes the shares $(s_1^*, \dots, s_n^*)$ that correspond to $\mathbf{M}^*$, and runs $VSReveal_{(n, n/2+2)}(s_1^*, \dots, s_n^*) = b'$, it then executes $b \leftarrow A_2(\sigma, (m_{i,j}, r_{i,j})_{(i,j) \in I})$ and outputs 1 iff $b = b'$.

Now consider $\Pr[B_H^{\text{Ind-SO-Real}}(1^k) = 1]$, this is a perfect simulation of $HB_A(1^k)$, and therefore by the assumption that A breaks the hidden-bit security, the term must exceed $1/2 + \epsilon$, where $\epsilon \geq 1/k^c$. In contrast, consider $\Pr[B_H^{\text{Ind-SO-Ideal}}(1^k) = 1]$. In the case that $VSReveal_{(n, n/2+2)}(s_1^*, \dots, s_n^*) = VSReveal_{(n, n/2+2)}(s_1, \dots, s_n)$, which occurs with probability exactly $1/2$, it is again a perfect simulation of $HB_A(1^k)$, and so the experiment outputs 1 with probability $1/2 + \epsilon$. In contrast, when $VSReveal_{(n, n/2+2)}(s_1^*, \dots, s_n^*) \neq VSReveal_{(n, n/2+2)}(s_1, \dots, s_n)$, then we know that A_2 outputs $VSReveal_{(n, n/2+2)}(s_1, \dots, s_n)$ with probability $1/2 + \epsilon$, and so B_2 outputs 1 with probability $1 - (1/2 + \epsilon) = 1/2 - \epsilon$. Therefore, $\Pr[B_H^{\text{Ind-SO-Ideal}}(1^k) = 1] = (1/2)(1/2 + \epsilon + 1/2 - \epsilon) = 1/2$. Therefore, $\Pr[B_H^{\text{Ind-SO-Real}}(1^k) = 1] - \Pr[B_H^{\text{Ind-SO-Ideal}}(1^k) = 1] = 1/2 + \epsilon - 1/2 \geq 1/k^c$, breaking IND-SO-SEC security.

Using the SOA Ciphertexts in a Secure Multiparty Computation Protocol In our SMC construction, we encode all users' inputs using the POK scheme above. The encrypted inputs are sent to the other parties. After each party's input has been confirmed with a proof of knowledge, the parties homomorphically evaluate the different ciphertexts to get an appropriate encrypted output. However, as explained before, the POK encryptions are not themselves homomorphic. To solve this problem we use Gentry's bootstrapping technique. Bootstrapping lets us take a ciphertext in an FHE scheme with any amount of noise that still allows for proper decryption (specially, this is potentially more noise than is permissible to perform any extra homomorphic operations without destroying the correctness of the ciphertext), and output a new ciphertext in the FHE scheme, of the same value, but with a small enough amount of noise that it can be properly computed on through the use of the FHE's evaluation function. Given a ciphertext $C = \{c_{i,j}\}_{i,j \in [n]}$ in the POK scheme, each $c_{i,j}$ is a ciphertext

from a lossy encryption scheme. To convert C into a corresponding encryption $c^\dagger$ in the TFHE scheme we do the following: We bootstrap each $c_{i,j}$ which is simply a TFHE ciphertext that has had the circuit-privacy function applied to it—thus containing potentially too much noise to apply further homomorphic operations to, but not so much that it decrypts improperly— to receive the corresponding lower-noise TFHE ciphertext $c'_{i,j}$. The c' ciphertexts can now be evaluated in the THFE eval function, and in particular we can use the TFHE eval function, to evaluate $VSReveal_{(n,n/2+2)}$. The result of this evaluation is the ciphertext $C^\dagger$ corresponding to the output.

Protocols vs. Algorithms We note that there is one technical issue that needs to be resolved, which is that in this section we have described the key generation and decryption algorithms as stand-alone algorithms, rather than protocols. For our purposes, we need a joint protocol for key generation and decryption. For this reason, we need to modify our key generation algorithm in the TFHE scheme to include an encryption of the bits 0 and 1 in the public-key. These values allow the parties to encrypt under the SOA secure encryption scheme $\hat{\Pi}$. The SOA secure scheme does not modify the decryption algorithm, so there is no need for modification to the decryption protocol.

4 Secure Multiparty Computation

We follow the Cramer et al. [CDN01] approach for constructing a multi-party computation protocol based on threshold cryptography. Our biggest changes are that we do not need a protocol for multiplication, we use a different approach for proving knowledge of encryption, and we explicitly describe a key generation phase whereas it is assumed as an external setup in [CDN01]. Since our solution requires less interaction among the parties, our simulation argument is simpler than the argument from [CDN01].

We use the standard simulation-based definition of stand-alone secure multi-party computation. We assume the existence of a standard n -party CoinFlipping protocol which guarantees soundness in the presence of $< n/2$ adversaries: namely, for any minority set of adversaries, the protocol guarantees that the distribution is still statistically close to uniform. Such a protocol can be easily constructed based on the existence of hiding commitments. (Unlike [CDN01], we do not need this coin flipping protocol to be simulatable.). See our full version [MSas11] for a definition of the real/ideal paradigm for secure multi-party computation from [CDN01] and [IKK⁺11]. In this section the TFHE scheme used is denoted $\tilde{\Pi} = (\tilde{G}, \tilde{E}, \tilde{D}, \mathbf{Eval})$.

We assume that the players can communicate via an authenticated broadcast channel and via point-to-point private and authenticated channels (which may in turn be implemented using signatures, public-key encryption, etc.)

Protocol 1 . Each party holds private input x_i ; the parties jointly compute $f(x_1, \dots, x_n)$.

- 1: Party P_i receives as input $(1^k, n, x_i)$. (We assume the adversary receives as input $1^k, n$, a set of corrupted parties C and the inputs $\{x_c\}_{c \in C}$ for the corrupted parties, and auxiliary information.)
- 2: Players run the TFHE key generation subprotocol $\tilde{G}(\eta, \tau, \rho, \theta, \Theta, \kappa)$ to generate a public-key $\tilde{\text{PK}}$ and shares of the secret for the threshold scheme $\tilde{\Pi}$. At the end of this step, player p_i holds share SK_i of the secret-key SK . If the sub-protocol halts prematurely, then players halt and output $\perp$.
- 3: The players take sequential turns sharing their input using the encryption scheme $\hat{\Pi}$ that is constructed from Π (see §3). More specifically, for $i \in [n]$, player P_i broadcasts $c_{i,j} \leftarrow \hat{E}(\tilde{\text{PK}}, x_{i,j})$. Then all of the players run a standard CoinFlipping protocol to generate a random string r_i . Player P_i now interprets r_i as n strings $r_{i,1}, \dots, r_{i,n}$ and uses coins $r_{i,j}$ as the random coins to run $\text{Verifier}(\text{PK}, c_{i,j})$ (see §3) of the Hidden Bit POK protocol on input $c_{i,j}$ for each bit $j \in [n]$ of input x_i . Player P_i runs the corresponding Prover algorithm on $c_{i,j}$ using the random coins used to generate $c_{i,j}$ as the witness, and broadcasts the Prover message. The remaining players also execute the Verifier algorithm using the same random coins and verify that the first message is consistent and the second message is accepted. If player P_i fails the POK protocol, then P_i is excluded from the rest of the protocol, and the remaining players that have not been excluded use a canonical encryption of 0 as the input for P_i (e.g., they use $\tilde{E}(\tilde{\text{PK}}, 0; 0)$ as each input bit).
- 4: The players that have not been excluded locally run $\mathbf{Eval}(\tilde{\text{PK}}, c_{1,1}, \dots, c_{n,n}, \tilde{f})$ where the function $\tilde{f}$ first transforms the input ciphertexts encrypted under $\hat{\Pi}$ into ones for scheme $\tilde{\Pi}$. This is done by homomorphically evaluating the decryption procedure described in §3 (i.e. bootstrapping, see Defn. 1). (Note: All of the ciphertexts in $c_{i,j}$ have a large degree of noise in them due to the circuit-privacy call that was used to rerandomize the ciphertexts. Therefore, the first thing that is done is that the ciphertexts are re-encoded with less noise using the same procedure as FHE bootstrapping.) Next, compute ciphertext z_i of the result $f(x_1, \dots, x_n)$. Note that each player can complete this step using only local information (since the public-key for the FHE includes all the information needed for evaluation).
- 5: Each player P_i that has not been excluded broadcasts the ciphertext z_i computed in the previous step. Each player then locally computes the majority of the broadcasts as ciphertext z' . A majority is guaranteed to exist since the malicious players form a minority and $\mathbf{Eval}$ is deterministic. Any player whose broadcast differs from the majority is excluded from the remaining portion of the protocol.

- 6: Players p_i that have not been excluded run the distributed subprotocol $\tilde{D}(z', \text{SK}_1, \dots, \text{SK}_n)$ using input z' and their local share SK_i . The output of the protocol is taken as the output.

Theorem 4. *Let π be Protocol 8 for a function f , and fix $s \in \{1, \dots, n/2\}$. If Π is a circuit-private TFHE encryption scheme, then for any ppt adversary A , there exists a ppt adversary A' such that for every polynomial-size circuit family $Z = Z_k$ corrupting a minority of parties the following is negligible:*

$$|\Pr[\text{REAL}_{\pi, A, Z}(k) = 1] - \Pr[\text{IDEAL}_{f, A', Z}(k) = 1]|.$$

See full version for details.

5 Threshold FHE for the Integers

In this section we briefly highlight the construction of a TFHE scheme $\tilde{\Pi} = (\tilde{G}, \tilde{E}, \tilde{D}, \mathbf{\tilde{Eval}})$ from the FHE scheme $\Pi = (G, E, D, \mathbf{Eval})$ based on the Approximate-GCD problem described by [vDGHV10]. The details are presented in our full version online. We point out that in any such transformation $\tilde{E} = E$ and $\mathbf{\tilde{Eval}} = \mathbf{Eval}$, and thus we only need to describe protocols for computing $\tilde{G}$ and $\tilde{D}$.

Sharing the Public and Secret-key Recall the secret-key p for the “somewhat homomorphic encryption scheme” is an odd η -bit integer. To sample p in a distributed fashion, we notice that the bits p_0 and $p_{\eta-1}$ should be 1 whereas the rest of the bits $p_1, \dots, p_{\eta-2}$ should be randomly shared. At the end, each player holds a share of p . We then extend techniques from [KLML05] to allow multiple parties who hold shares of p to compute shares of $1/p$ and $x_p = \lfloor 2^\kappa/p \rfloor$.

Recall that the secret-key for Π consists of a Θ -bit vector $\mathbf{s}$ with Hamming weight θ . Our first modification to Π is to note that instead of θ , it suffices to select a vector with Hamming weight in the interval $\theta \pm \theta/4$. To verify this, note that the sparse subset-sum problem is assumed to be hard for $\theta = \Theta^\epsilon$ for $0 < \epsilon < 1$; our change does not violate this condition. Also, our new range of settings for θ does not increase the total degree of the decryption circuit by more than a factor of 2 and thus the condition that the decryption protocol is admissible is maintained (and thus the scheme is bootstrappable. See the computation on p.18 [vDGHV10].) Our approach for producing $\mathbf{s}$ is to securely generate a random number r_i in the range $[0, \Theta]$ for each s_i and setting $s_i = 1$ if $r_i \leq \theta$ and 0 otherwise.

The public-key consists of the vectors $\mathbf{x}$ and $\mathbf{u}$. Using $\mathbf{s}$ and x_p , we compute the vector $\mathbf{u}$ using the formula $\mathbf{u} = \sum_i s_i \cdot u_i \bmod 2^{\kappa+1}$. These shares can be used to compute the vector $\mathbf{y}$.

Using bits of $1/p$ computed in previous steps, we generate the x_i 's. Recall from the original public-key generation algorithm that we need to sample $x_i \leftarrow D_{\gamma, \rho}(p)$

for $i = 0, \dots, \tau$. Intuitively, these x_i represent random encryptions of 0 that get added to our base encryption in the homomorphic scheme. Further, recall that

$$D_{\gamma, \rho}(p) = \{\text{choose } q \leftarrow Z \cap [0, 2^\gamma/p], r \leftarrow Z \cap (-2^\rho, 2^\rho) : \text{output } x \leftarrow pq + r\}.$$

After sampling, the list should be relabeled so that x_0 is the largest. The key-generation process requires that the process is restarted if either x_0 is even or $x_0 - \lfloor x_0/p \rfloor \cdot p$ is odd. Since $x_0 = pq + r$ is generated as directed for some random q and r and since p is an odd number, the requirement that x_0 is odd can be checked by inspecting the least significant bits of the q and r : If $q_0 + r_0 = 1$, then x_0 satisfies the first condition. To check the second condition, that $x_0 - \lfloor x_0/p \rfloor \cdot p$ is an odd number, we observe that because of the constraints $-2^\rho < r < 2^\rho$ and $2^{\eta-1} \leq p < 2^\eta$, it follows that $-2^{\rho-\eta+1} < r/p < 2^{\rho-\eta+1}$.

Since $\rho = \lambda$ and $\eta = \tilde{O}(\lambda^2)$, therefore for all sufficiently large λ (if $\eta = \lambda^2$, then for $\lambda > 2$), $\lfloor r/p \rfloor = 0$ and as a result r can be ignored. That is $\lfloor x_0/q \rfloor = \lfloor pq + r/q \rfloor = q + \lfloor r/q \rfloor = q$. So $x_0 - \lfloor x_0/p \rfloor \cdot p = x_0 - q \cdot p$. Because x_0 and p are both odd, q must be odd to make the term $x_0 - \lfloor x_0/p \rfloor \cdot p$ even. These constraints imply that for x_0 to be odd and $x_0 - \lfloor x_0/p \rfloor \cdot p$ to be even, then q must be even and r must be odd.

Computing encryptions of s One step in Gentry's paradigm for FHE construction requires the public-key to contain an encryption of the secret-key. We assume circular security of the underlying encryption scheme, as do van Dijk et al. [vDGHV10] and Gentry [Gen09b]. Towards this goal, we design a protocol that enables players who hold private shares of the secret-key (as well as the entire public-key) to compute an encryption of the secret-key under the public-key. Note this *cannot* be done trivially with homomorphic evaluation because the encrypted secret-key is in fact necessary to homomorphically evaluate circuits of an arbitrary depth, resulting in a circular requirement.

Recall that in Dijk et al. [vDGHV10], the encryption of m under the public-key $\langle x_0, \dots, x_\tau \rangle$ computes as $[m + 2r + 2 \sum_{i \in S} x_i]_{x_0}$, where $r \in (-2^{\rho'}, 2^{\rho'})$ and $S \subseteq \{1, \dots, \tau\}$ is a random subset. Since both the x_i 's and r can take negative values (as integers) whereas the computation is in a finite field, we need to somehow make sure the computation in the finite field result in the same integer value of the encryption of m . To resolve this issue, we compute the value $\min$ which is a unique value that satisfies the following two properties: 1) $\min = 0 \bmod x_0$, and 2) for an arbitrary S and for our set of x_i 's and any value of r , it would make the summation $m + 2r + 2 \sum_{i \in S} x_i$ positive. Because the range of values that r can take is public, all users can compute $\min$ locally and agree on respective shares. Next, to encrypt the secret-key, all users generate shares for a set S and the shares for a value r . All users then add their shares of r , use shares in S to add in appropriate x_i 's, and add $\min$. See the full version for details.

Computing encryptions of 0 and 1 for PK The same techniques from the previous step can be used to produce encryptions of random bits. These encryptions can then be collaboratively decrypted until both an encryption of 0 and an encryption

of 1 are identified. These two ciphertexts can then be adjoined to the public-key—they are guaranteed to be well-formed and have the right amount of noise.

References

- AJLA⁺12. Gilad Asharov, Abhishek Jain, Adriana López-Alt, Eran Tromer, Vinod Vaikuntanathan, and Daniel Wichs. Multiparty computation with low communication, computation and interaction via threshold fhe. In *EUROCRYPT*, pages 483–501, 2012. 1, 4, 5
- BDOZ11. Rikke Bendlin, Ivan Damgård, Claudio Orlandi, and Sarah Zakarias. Semi-homomorphic encryption and multiparty computation. In *EUROCRYPT*, pages 169–188, 2011. 5
- Bea91. Donald Beaver. Efficient multiparty protocols using circuit randomization. In *CRYPTO*, pages 420–432, 1991. 5
- BHY09. Mihir Bellare, Dennis Hofheinz, and Scott Yilek. Possibility and impossibility results for encryption and commitment secure under selective opening. In *EUROCRYPT*, pages 1–35, 2009. 9, 10, 13
- CDD⁺99. Ronald Cramer, Ivan Damgård, Stefan Dziembowski, Martin Hirt, and Tal Rabin. Efficient multiparty computations secure against an adaptive adversary. In *EUROCRYPT*, pages 311–326, 1999. 20
- CDN01. Ronald Cramer, Ivan Damgård, and Jesper Buus Nielsen. Multiparty computation from threshold homomorphic encryption. In *EUROCRYPT*, pages 280–299, 2001. 6, 7, 8, 9, 15
- CDSMW08. Seung Geol Choi, Dana Dachman-Soled, Tal Malkin, and Hoeteck Wee. Black-box construction of a non-malleable encryption scheme from any semantically secure one. In *TCC*, pages 427–444, 2008. 6
- DIK10. Ivan Damgård, Yuval Ishai, and Mikkel Krøigaard. Perfectly secure multiparty computation and the computational overhead of cryptography. In *EUROCRYPT*, pages 445–465, 2010. 2
- DPSZ12. Ivan Damgård, Valerio Pastro, Nigel P. Smart, and Sarah Zakarias. Multiparty computation from somewhat homomorphic encryption. In *CRYPTO*, pages 643–662, 2012. 3, 5
- Gen09a. Craig Gentry. *A fully homomorphic encryption scheme*. PhD thesis, Stanford University, 2009. crypto.stanford.edu/craig. 1, 2, 5, 8
- Gen09b. Craig Gentry. Fully homomorphic encryption using ideal lattices. In *STOC*, pages 169–178, 2009. 10, 18
- GLOV12. Vipul Goyal, Chen-Kuei Lee, Rafail Ostrovsky, and Ivan Visconti. Constructing non-malleable commitments: A black-box approach. In *FOCS*, pages 51–60, 2012. 6
- GO94. Oded Goldreich and Yair Oren. Definitions and properties of zero-knowledge proof systems. *J. Cryptology*, 7(1):1–32, 1994. 1, 2
- HLOV11. Brett Hemenway, Benoît Libert, Rafail Ostrovsky, and Damien Vergnaud. Lossy encryption: Constructions from general assumptions and efficient selective opening chosen ciphertext security. In *ASIACRYPT*, pages 70–88, 2011. 3, 9, 10
- IKK⁺11. Yuval Ishai, Jonathan Katz, Eyal Kushilevitz, Yehuda Lindell, and Erez Petrank. On achieving the “best of both worlds” in secure multiparty computation. *SIAM J. Comput.*, 40(1):122–141, 2011. 6, 15

- JJ00. Markus Jakobsson and Ari Juels. Mix and match: Secure function evaluation via ciphertexts. In *ASIACRYPT*, pages 162–177, 2000. [6](#)
- KLML05. Eike Kiltz, Gregor Leander, and John Malone-Lee. Secure computation of the mean and related statistics. In *TCC*, pages 283–302, 2005. [17](#)
- LATV11. Adriana López-Alt, Eran Tromer, and Vinod Vaikuntanathan. Cloud-assisted multiparty computation from fully homomorphic encryption. *IACR Cryptology ePrint Archive*, 2011:663, 2011. [1](#), [4](#), [5](#)
- MSas11. Steven Myers, Mona Sergi, and abhi shelat. Threshold fully homomorphic encryption and secure computation. *Cryptology ePrint Archive*, Report 2011/454, 2011. <http://eprint.iacr.org/>. [7](#), [11](#), [13](#), [15](#)
- NN01. Moni Naor and Kobbi Nissim. Communication preserving protocols for secure function evaluation. In *STOC*, pages 590–599, 2001. [2](#)
- vDGHV10. Marten van Dijk, Craig Gentry, Shai Halevi, and Vinod Vaikuntanathan. Fully homomorphic encryption over the integers. In *EUROCRYPT*, pages 24–43, 2010. [4](#), [8](#), [17](#), [18](#)

A Verifiable Secret-Sharing Scheme

A $\binom{n}{n/2+2}$ Verifiable Secret-Sharing scheme consists of a sharing algorithm which takes as input a secret s and produces n -shares $s_1, \dots, s_n$. These shares have the property that for any $T \subset \{1, \dots, n\}$, $|T| < n/2 + 2$ it is the case that $\{s_i\}_{i \in T}$ is information theoretically independent from s . However, for any $S \subseteq \{1, \dots, n\}$, $|S| \geq n/2 + 2$, it is the case that the reveal algorithm, when given $\{s_i\}_{i \in S}$, can reconstruct s . In a traditional interactive setting we require that all non-cheating parties agree on the reconstructed secret. We use a modification of the Cramer et al. [CDD⁺99] verifiable secret sharing scheme; we do not need to deal with interactive adversaries, nor players, so the scheme is significantly simplified. We present the sharing and revealing algorithms in our full version.

Protocol 2 . $[s]$, VSShare(s)

- 1: Choose a random degree $n/2 + 1$ bi-variate polynomial f such that $f(0, 0) = s$.
- 2: Share $s_i = (\mathbf{a}, \mathbf{b}) = (i, (f(i, 1), \dots, f(i, n)), (f(1, i), \dots, f(n, i)))$.
- 3: Output $s_1, \dots, s_n$.

Protocol 3 . $[s]$, VSReveal $_{(n, n/2+2)}(s_1, \dots, s_{n/2+2})$

- 1: For each $s_i = (i, \mathbf{a}_i, \mathbf{b}_i)$ ensure that a_i and b_i are $n/2 + 2$ -consistent
- 2: If not output $\perp$.
- 3: For each $i \neq j$ ensure s_j, s_i are pairwise-consistent
- 4: If not output $\perp$.
- 5: Interpolate f , based on shares.
- 6: Output $f(0, 0)$

Definition 5. A vector $(e_1, \dots, e_n) \in F_n$ is $n/2 + 2$ -consistent if there exists a polynomial w of degree at most $n/2 + 1$ such that $w(i) = e_i$ for $0 \leq i < n$.

Definition 6. Given two shares $s_i = (i, \mathbf{a}_i = (a_{i1}, \dots, a_{in}), \mathbf{b}_i = (b_{1i}, \dots, b_{ni}))$ and $s_j = (j, \mathbf{a}_j = (a_{j1}, \dots, a_{jn}), \mathbf{b}_j = (b_{1j}, \dots, b_{nj}))$, we say that they are pairwise consistent if $a_{ij} = b_{ij}$ and $a_{ji} = b_{ji}$.

Definition 7. For our purposes it is useful to note that given the $n \times n$ matrix

$$\begin{bmatrix} f(1,1) & f(1,2) & \dots & f(1,n) \\ f(2,1) & f(2,2) & \dots & f(2,n) \\ \vdots & \vdots & \ddots & \vdots \\ f(n,1) & f(n,2) & \dots & f(n,n) \end{bmatrix},$$

that a share s_i simply corresponds to the i^{th} row and column of the matrix. We will call this the matrix representation of the shares. Notice that when given in the matrix representation, any two shares are necessarily pairwise consistent. Given a set of n pairwise consistent shares $\mathbf{s} = (s_1, \dots, s_n)$, we define $M_{\mathbf{s}}$ as the $n \times n$ matrix representation of the shares.

Black-box Constructions of Composable Protocols without Set-Up

Huijia Lin*

Rafael Pass†

Abstract

We present the first *black-box* construction of a secure multi-party computation protocol that satisfies a meaningful notion of *concurrent security* in the plain model (without any set-up, and without assuming an honest majority). Moreover, our protocol relies on the minimal assumption of the existence of a semi-honest OT protocol, and our security notion “UC with super-polynomial helpers” (Canetti et al, STOC’10) is closed under universal composition, and implies “super-polynomial-time simulation”.

*MIT and Boston University, E-Mail: huijia@csail.mit.edu.

†Cornell University, E-Mail: rafael@cs.cornell.edu. Pass is supported in part by a Alfred P. Sloan Fellowship, Microsoft New Faculty Fellowship, NSF CAREER Award CCF-0746990, AFOSR YIP Award FA9550-10-1-0093, and DARPA and AFRL under contract FA8750-11-2-0211. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the Defense Advanced Research Projects Agency or the US government.

1 Introduction

The notion of *secure multi-party computation* allows m mutually distrustful parties to securely compute (or, *realize*) a functionality $f(\bar{x})$ of their corresponding private inputs $\bar{x} = x_1, \dots, x_m$, such that party P_i receives the i^{th} component of $f(\bar{x})$. Loosely speaking, the security requirements are that the output of each party is distributed according to the prescribed functionality—this is called *correctness*—and that even malicious parties learn nothing more from the protocol than their prescribed output—this is called *privacy*. These properties should hold even in case that an arbitrary subset of the parties maliciously deviates from the protocol.

Soon after the concept was proposed [Yao86], general constructions were developed that appeared to satisfy the intuitive correctness and secrecy for practically any multi-party functionality [Yao86, GMW87]. These constructions require only authenticated communication and can use any enhanced trapdoor permutation. However, definitions that capture the security properties of secure multi-party computation protocols (and, in fact, of secure cryptographic protocols in general) took more time to develop. Here, the *simulation paradigm* emerged as a natural approach: Originally developed for capturing the security of encryption and then extended to Zero-Knowledge [GM84, GMR89]. The idea is to say that a protocol π securely realizes f if running π “*emulates*” an idealized process where all parties secretly provide inputs to an imaginary trusted party that computes f and returns the outputs to the parties; more precisely, any “harm” done by a polynomial-time adversary in the real execution of π , could have been done even by a polynomial-time adversary (called a *simulator*) in the ideal process. The simulation paradigm provides strong security guarantees: It ensures that running the protocols is “as good as” having a trusted third party computing the functionality for the players, and an adversary participating in the real execution of the protocols does not gain any “computational advantage” over the simulator in the ideal process (except from polynomial time advantage). We call this definition *basic security*.

The original setting in which secure multi-party protocols were investigated, however, only allowed the execution of a single instance of the protocol at a time; this is the so called stand-alone setting. A more realistic setting, is one which allows the concurrent execution of protocols. In the concurrent setting, many protocols are executed at the same time. This setting presents a new risk of a “coordinated attack” in which an adversary interleaves many different executions of a protocol and chooses its messages in each instance based on other partial executions of the protocol. To prevent coordinated attacks, we require the following basic security guarantee:

Concurrent Security: The security properties, correctness and privacy, of the *analyzed protocol* should remain valid even when if multiple instance of the protocol are concurrently executed in a potentially unknown environment.

Another natural desideratum is the capability of supporting modular design of secure protocols.

Modular analysis: The notion of security should support designing composite protocols in a modular way, while preserving security. That is, there should be a way to deduce security properties of the overall protocol from security properties of its components. This is essential for asserting security of complex protocols.

Unfortunately, these properties are not implied by the basic security. In the literature, the strongest and also the most realistic formalization of concurrent security is the notion of Universal Composability (UC) [Can01]: It considers the concurrent execution of an unbounded number of instances of the analyzed protocol, in an arbitrary, and adversarially controlled, network environment. It also supports modular analysis of protocols. But, these strong properties come at a price: Many natural functionalities cannot be realized with UC security in the *plain model*, where players only have access to authenticated communication channels; some additional trusted

set-up is necessary [CF01, CKL03]; furthermore, the need for additional trusted set up extends to any protocol that only guarantees a concurrent extension of basic security [Lin04]. A large body of works (e.g. [CLOS02, BCNP04, KLP05, CPS07, GO07, Kat07, CDPW07]) have shown that indeed, with the appropriate trusted set-ups, UC-security becomes feasible. However, in many situations, trusted set-up is hard to come by (or at least expensive). It is thus important to have a notion of concurrent security that can be achieved in the plain model.

Concurrent Security in the Plain model. *Security with super-polynomial simulators (SPS)* [Pas03a] is a relaxation of UC security that allows the adversary in the ideal execution to run in super-polynomial time. Informally, this corresponds to guaranteeing that “any polytime attack that can be mounted against the protocol can also be mounted in the ideal execution—albeit with super-polynomial resources.” Although SPS security is sometimes weaker than basic security, it often provides an adequate level of security. In contrast to basic security, however, SPS directly considers security in the concurrent setting. Protocols that realize practically any functionality with SPS security in the plain model were shown based on sub-exponential hardness assumptions [Pas03a, BS05, LPV09]. Very recently, improved constructions are presented [CLP10, GGJS12, LPV12] that are based on only standard polynomial-time hardness assumptions.

One drawback of SPS security is that it is not closed under composition; thus it is not a convenient basis for modular analysis of protocols. Angel-based UC security [PS04] is a framework for notions of security that provides similar security guarantees as SPS and at the same time supports modular analysis. Specifically, angel-based security considers a model where both the adversary and the simulator have access to an oracle (an “angel”) that allows some judicious use of super-polynomial resources. Since the angels can be implemented in super-polynomial time, for any angel, angel-based security implies SPS security. Furthermore, akin to UC security, angel-based UC security, with any angel, can be used as a basis for modular analysis. Prabhakaran and Sahai [PS04] exhibited an angel with respect to which practically all functionalities can be securely realized; later another angel is given by [MMY06]; both constructions, however, rely on some non-standard hardness assumptions.

Recently, Canetti, Lin and Pass [CLP10] proposed a new notion of security, called *UC with super-polynomial time helpers*. This notion is very similar to the angel-based security where both the adversary and the simulator have access to a helper that provides some super-polynomial time help through a limited interface. Like angel-based security, UC security with super-polynomial time helpers implies SPS security. But, unlike angel-based security where angels are basically non-interactive and stateless, the helpers are *highly interactive and stateful*. Canetti, Lin and Pass [CLP10] then constructed protocols that realize practically all functionalities with respect to a particular super-polynomial-time interactive helper, based on the existence of trapdoor permutations.

Summarizing the state-of-the-art, there are constructions [CLP10, GGJS12, LPV12] of protocols satisfying a meaningful notion of concurrent security—SPS security—in the plain model based on standard polynomial time hardness assumptions. Furthermore, the construction of [CLP10] also supports modular analysis (the constructions of [GGJS12, LPV12] are better in terms of round-complexity—they only require a constant number of communication rounds—but they only achieve “non-composable” SPS security).

However, all these constructions are *non-black-box*, that is, the constructed protocols make non-black-box use of the underlying primitives. In fact, these constructions all follow the “Feige-Shamir” paradigm [FS90]: The protocols contain “trapdoors” embedded into the messages of the protocol, allowing a super-polynomial time simulator to extract the trapdoor and simulate messages in the protocol by “proving that it knows the trapdoor”. In general, protocols following this approach seem hard to turn into a “practical” protocol for secure computations; as such, their results should only be viewed as “feasibility results” regarding concurrent secure computation without set-up, but not candidates for practical purposes.

In contrast, black-box constructions that only use the underlying primitives through their in-

put/output interfaces, are often much more efficient and are more suitable for implementation. Therefore, a series of recent works [DI05, IKLP06, IPS08, LP07, Wee10, Goy11] have focused on constructing *black-box construction of secure computation protocols*, as an important step towards bringing secure multi-party computation closer to the practice. However, their constructions are all in either the stand-alone setting or rely on strong trusted set-ups (e.g., trusted hardware). This leaves open the following basic question:

Can we obtain a black-box construction of concurrently secure protocols in the plain model (preferably based only standard polynomial-time assumptions)?

Can we have such a black-box construction that also satisfies a notion of security supporting composability?

1.1 Our Results

We present a black-box construction of protocols that satisfy UC security with super-polynomial time helper for a specific helper, based on the existence of a stand-alone semi-honest oblivious transfer (OT) protocols; that is, the weakest possible assumption. The framework of UC with super-polynomial time helper of [CLP10] is formalized through the extended UC (EUC) framework of [CDPW07]; it is identical to the standard UC model [Can00] except that the corrupted parties (and the environment) have access to an additional super-polynomial time entity $\mathcal{H}$, called a helper functionality.

Main Theorem (Informally Stated): *Assume the existence of stand-alone semi-honest oblivious transfer protocols. Then there exists a sub-exponential-time computable interactive machine $\mathcal{H}$ such that for any “well-formed” polynomial-time functionality $\mathcal{F}$, there exists a protocol that realizes $\mathcal{F}$ with $\mathcal{H}$ -EUC security, in the plain model. Furthermore, the protocol makes only black-box calls to the underlying oblivious transfer protocol.*

As far as we know, this is the first black-box construction of secure multi-party computation protocols that achieve any non-trivial notion of concurrent security in the plain model (without any trusted-set up, and without assuming an honest majority).

The main technical tool used in our construction is a new notion of a commitment that is secure against adaptive chosen commitment attack (CCA security). The notion of CCA secure commitments was previously introduced in [CLP10]. Roughly speaking, a tag-based commitment scheme (i.e., commitment scheme that take an identifier—called the tag—as an additional input) is said to be *CCA-secure* if the value committed to using the tag id remains hidden even if the receiver has access to a (super-polynomial time) oracle that “breaks” commitments using any tag $\text{id}' \neq \text{id}$, where by breaking, it means the oracle returns a decommitment of the commitment. Thus the oracle is called a decommitment oracle. In [CLP10], a commitment scheme that is CCA-secure w.r.t. a decommitment oracle is constructed based on the minimal assumption of one-way functions. However, their construction is non-black-box. In this work, to obtain black-box secure computation protocols, we need a new *black-box* construction of a CCA-secure commitment scheme. Towards this, we weaken the notion of CCA security w.r.t. decommitment oracle to instead consider an oracle that “breaks” commitments by returning only the unique committed value (and not the decommitment information) of the commitments (or $\perp$ if there is no unique committed value); we call this the committed-value oracle. We then provide a black-box construction of a commitment scheme that is CCA-secure w.r.t. the committed-value oracle.

Theorem (Informally Stated): *Assume the existence of one-way functions. Then, for every $\epsilon > 0$, there exists an $O(n^\epsilon)$ -round commitment scheme that is CCA-secure w.r.t. the committed-value oracle and only relies on black-box access to one-way functions (where n is the security parameter).*

We next show that the notion of CCA-secure commitments intuitively is better behaved than traditional notions of non-malleability [DDN00] in the context of black-box construction of concurrently secure protocol. On a very high-level (and significantly oversimplifying), CCA security of commitment schemes allow us to deal with “cut-and-choose” techniques (typically used in black-box constructions) in concurrent executions, while ensuring hiding of commitments in other executions.

1.2 Outline

In Section 2, we define the notion of CCA-security w.r.t. the committed-value oracle. (Notations and definitions of basic primitives appear in Appendix A.) In Section 4, we present our black-box robust CCA-secure commitment scheme; and provide the proof of security in Appendix B. We recall the notion of UC security with super-polynomial time helper in Appendix C, and show how to achieve this security notion using CCA-secure commitments in a black-box way in Section 3.

2 Definition of CCA-Secure Commitments

We assume familiarity with the definition of commitment schemes and the statistically/computational binding and statistically/computational hiding properties. Unless specified otherwise, by a commitment scheme, we mean one that is statistically binding and computationally hiding. A *tag-based commitment schemes* with $l(n)$ -bit identities [PR05, DDN00] is a commitment scheme where, in addition to the security parameter 1^n , the committer and the receiver also receive a “tag”—a.k.a. the identity—id of length $l(n)$ as common input.

2.1 CCA-Security w.r.t. Committed Value Oracle

Let $\langle C, R \rangle$ be a tag-based commitment scheme with $l(n)$ -bit identities. A committed-value oracle $\mathcal{O}$ of $\langle C, R \rangle$ acts as follows in interaction with an adversary A : it participates with A in many sessions of the commit phase of $\langle C, R \rangle$ as an honest receiver, using identities of length $l(n)$, chosen adaptively by A . At the end of each session, if the session is *valid*, it reveals the unique committed value of that session to A ; otherwise, it sends $\perp$. (If a session has multiple committed values, the decommitment oracle also returns $\perp$. The statistically binding property guarantees that this happens with only negligible probability.) Loosely speaking, a tag-based commitment scheme $\langle C, R \rangle$ is said to be CCA-secure w.r.t. the committed-value oracle, if the hiding property of the commitment holds even with respect to adversaries with access to the committed-value oracle $\mathcal{O}$. More precisely, denote by $A^{\mathcal{O}}$ the adversary A with access to the committed-value oracle $\mathcal{O}$. Let $\text{IND}_b(\langle C, R \rangle, A, n, z)$, where $b \in \{0, 1\}$, denote the output of the following probabilistic experiment: on common input 1^n and auxiliary input z , $A^{\mathcal{O}}$ (adaptively) chooses a pair of challenge values $(v_0, v_1) \in \{0, 1\}^n$ —the values to be committed to—and an identity $\text{id} \in \{0, 1\}^{l(n)}$, and receives a commitment to v_b using identity id . Finally, the experiment outputs the output y of $A^{\mathcal{O}}$; the output y is replaced by $\perp$ if during the execution A sends $\mathcal{O}$ any commitment using identity id (that is, any execution where the adversary queries the decommitment oracle on a commitment using the same identity as the commitment it receives, is considered invalid). Let

Definition 1 (CCA-secure Commitments.). *Let $\langle C, R \rangle$ be a tag-based commitment scheme with $l(n)$ -bit identities. We say that $\langle C, R \rangle$ is CCA-secure w.r.t. the committed-value oracle, if for every PPT ITM A , the following ensembles are computationally indistinguishable:*

- $\{\text{IND}_0(\langle C, R \rangle, A, n, z)\}_{n \in N, z \in \{0, 1\}^*}$
- $\{\text{IND}_1(\langle C, R \rangle, A, n, z)\}_{n \in N, z \in \{0, 1\}^*}$

2.1.1 k -Robustness w.r.t. Committed-Value Oracle

Consider a man-in-the-middle adversary that participates in an *arbitrary* left interaction with a *limited number of rounds*, while having access to a committed oracle. Roughly speaking, $\langle C, R \rangle$ is k -robust if the (joint) output of every k -round interaction, with an adversary having access to the oracle $\mathcal{O}$, can be simulated without the oracle. In other words, having access to the oracle does not help the adversary in participating in any k -round protocols much.

Definition 2. Let $\langle C, R \rangle$ be a tag-based commitment scheme with $l(n)$ -bit identities. We say that $\langle C, R \rangle$ is k -robust w.r.t. the committed-value oracle, if there exists a simulator S , such that, for every PPT adversary A , the following two conditions hold.

Simulation: For every PPT k -round ITM B , the following two ensembles are computationally indistinguishable.

- $\{\text{output}_{B,A^\mathcal{O}}[\langle B(y), A^\mathcal{O}(z) \rangle(1^n, x)]\}_{n \in N, x, y, z \in (\{0,1\}^*)^3}$
- $\{\text{output}_{B,S^A}[\langle B(y), S^A(z) \rangle(1^n, x)]\}_{n \in N, x, y, z \in (\{0,1\}^*)^3}$

where $\text{output}_{A,B}[\langle B(y), A(z) \rangle(x)]$ denote the joint output of A and B in an interaction between them, on common input x and private inputs z to A and y to B respectively, with uniformly and independently chosen random inputs to each machine.

Efficiency: There exists a polynomial t and a negligible function μ , such that, for every $n \in N$, $z \in \{0,1\}^*$ and $x \in \{0,1\}^*$, and every polynomial T , the probability that S with oracle access to $A(z)$ and on input $1^n, x$, runs for more than $T(n)$ steps is smaller than $\frac{t(n)}{T(n)} + \mu(n)$.

The following proposition shows that to construct a robust CCA-secure commitment scheme for identities of length n , it suffices to construct one for identities of length $\ell(n) = n^\varepsilon$. The same proposition is established in [CLP10] for robust CCA-security w.r.t. decommitment oracles, and the proof there also applies to CCA-security w.r.t. committed-value oracles; we omit the proof here.

Proposition 1. Let ε be any constant such that $0 < \varepsilon < 1$, ℓ a polynomial such that $\ell(n) = n^\varepsilon$, and $\langle C, R \rangle$ a k -round robust CCA-secure commitment scheme (w.r.t. the committed-value oracle) with ℓ -bit identities. Then assuming the existence of one-way functions, there exists a robust $k+1$ -round CCA-secure commitment scheme $\langle \hat{C}, \hat{R} \rangle$ (w.r.t. the committed-value oracle) with n -bit identities.

3 Black-Box UC-Secure Protocols with Super-Polynomial Helpers

We consider the model of UC with super-polynomial helper introduced in [CLP10]. At a very high-level, this model is essentially the same as the UC-model introduced by [Can00], except that both the adversary and the environment in the real and ideal worlds have access to a super-polynomial time functionality that acts as a helper. A formal definition of the model is presented in Appendix C. In this section, we show the following theorem:

Theorem 1. Let δ be any positive constant. Assume the existence of a T'_{OT} -round stand-alone semi-honest oblivious transfer protocol. Then there exists a super-polynomial time helper functionality $\mathcal{H}$, such that, for every well-formed functionality¹ $\mathcal{F}$, there exists a $O(\max(n^\delta, T'_{OT}))$ -round protocol Π that $\mathcal{H}$ -EUC-emulates $\mathcal{F}$. Furthermore, the protocol Π only uses the underlying oblivious transfer protocol in a black-box way.

¹See [CLOS02] for a definition of well-formed functionalities.

Towards this theorem, we need to first exhibit a super-polynomial time helper functionality $\mathcal{H}$. Roughly speaking, $\mathcal{H}$ simply acts as the committed-value oracle of a CCA secure commitment scheme. More precisely, consider the following two building blocks: First, given any $T'_{OT}(n)$ -round stand-alone semi-honest OT protocol, it follows from previous works [IKLP06, Hai08] that there exists an $T_{OT}(n)$ -round OT protocol $\langle S, R \rangle$ that is secure against a malicious sender and a semi-honest receiver—called mS-OT protocol for short—that only relies on black-box access to the semi-honest OT protocol; furthermore $T_{OT} = O(T'_{OT}(n))$. Second, we need a $T_{OT}(n)$ -robust CCA-secure commitment scheme $\langle C, R \rangle$, whose committed-value oracle $\mathcal{O}$ can be computed in sub-exponential time.² As we will show in the next section (see Theorem 2), such a protocol exists with $O(\max(T_{OT}, n^\delta)) = O(\max(T'_{OT}, n^\delta))$ rounds, relying on the underlying OWF in a black-box way. Since OWFs can be constructed from a semi-honest OT protocol in a black-box way. Therefore, we have that the second building block can also be based on the semi-honest OT protocols in a black-box way.

Consider a helper functionality $\mathcal{H}$ that “breaks” commitments of $\langle C, R \rangle$ in the same way as its committed-value oracle $\mathcal{O}$ does, subject to the condition that player P_i in a protocol instance sid can only query the functionality on commitments that uses identity (P_i, sid) . More precisely, every party P_i in a secure computation can simultaneously engage with $\mathcal{H}$ in multiple sessions of the commit phase of $\langle C, R \rangle$ as a committer using identity P_i , where the functionality simply forwards all the messages internally to the committed-value oracle $\mathcal{O}$, and forwards P_i the committed value returned from $\mathcal{O}$ at the end of each session. Since the committed-value oracle $\mathcal{O}$ can be computed in sub-exponential time, this functionality can also be implemented in sub-exponential time. A formal description of the functionality appears in Figure 6 in Appendix D.

We show that Theorem 1 holds w.r.t. the helper functionality defined above in two steps. First, note that to realize any well-formed functionality in a black-box way, it suffices to realize the *ideal oblivious transfer functionality* $\mathcal{F}_{OT}$ [Rab05, EGL85]. This is because it follows from previous works [Kil92, BOGW88, GMW91, IPS08] that every functionality can be UC securely implemented in the $\mathcal{F}_{OT}$ -hybrid model, even w.r.t. super-polynomial time environments. Based on previous works, [CLP10] further shows that by considering only dummy adversaries and treating environments with access to a super-polynomial functionality $\mathcal{H}$ as sub-exponential time machines, we have that every functionality can be $\mathcal{H}$ -EUC securely implemented in the $\mathcal{F}_{OT}$ model. Formally, we have the following lemma from [CLP10].

Lemma 1. *Fix any super-polynomial time functionality $\mathcal{H}$. For every well-formed functionality $\mathcal{F}$, there exists a constant-round $\mathcal{F}_{OT}$ -hybrid protocol that $\mathcal{H}$ -EUC-emulates $\mathcal{F}$.*

Next we show how to implement the $\mathcal{F}_{OT}$ functionality in the $\mathcal{H}$ -EUC model. Then combining with Lemma 1, we conclude Theorem 1.

Lemma 2. *Let δ be any positive constant. Assume the existence of a T'_{OT} -round semi-honest oblivious transfer protocol. Then there exists a $O(\max(n^\delta, T'_{OT}))$ -round protocol Π_{OT} that $\mathcal{H}$ -EUC-emulates $\mathcal{F}_{OT}$. Furthermore, the protocol Π_{OT} only uses the underlying oblivious transfer protocol in a black-box way.*

3.1 Overview of the OT Protocol Π_{OT}

In this section we first provide an overview of our black-box construction of $\mathcal{H}$ -EUC secure OT protocol Π_{OT} , in comparison with the black-box construction of an OT protocol secure against both malicious players from a mS-OT protocol of [IKLP06, Hai08]. Roughly speaking, the protocol

²This can be instantiated by simply using a normal T_{OT} -robust CCA secure commitments with an exponential time committed value $\mathcal{O}$, with a “scaled-down” security parameter.

of [IKLP06, Hai08], relying on a stand-alone mS-OT protocol $\langle S, R \rangle$, proceeds in the following four stages:

Stage 1 (Receiver’s Random Tape Generation) The sender and the receiver jointly decide the receiver’s inputs and random tapes in Stage 2 using $2n$ parallel “coin tossing in the well” executions.

Stage 2 (OT with Random Inputs) The sender and the receiver perform $2n$ parallel OT executions of $\langle S, R \rangle$ using *random* inputs (s_j^0, s_j^1) and r_j respectively, where the receiver’s inputs r_j ’s (and its random tapes) are decided in Stage 1.

Stage 3 (Cut-and-Choose) A random subset $Q \subset [2n]$ of n locations is chosen using a 3-round coin-tossing protocol where the sender commits to a random value first. (Thus the receiver knowing that random value can bias the coin-tossing output.) The receiver is then required to reveal its randomness in Stage 1 and 2 at these locations, which allows the sender to check whether the receiver behaved honestly in the corresponding OT executions. The randomness of the receiver at the rest of locations remains hidden.

Stage 4 (OT Combiner) Finally, for these locations $j \notin Q$ that are not open, the receiver sends $\alpha_j = u \oplus c_j$ where u is the receiver’s true input. The sender replies with $\beta_0 = v_0 \oplus (\bigoplus_{j \notin Q} s_j^{\alpha_j})$ and $\beta_1 = v_1 \oplus (\bigoplus_{j \notin Q} s_j^{1-\alpha_j})$. The honest receiver obtains $s_j^{c_j}$ ’s through the OT execution, and thus can always recover v_u .

At a very high-level, the protocol of [IKLP06, Hai08] augments security of the mS-OT protocol $\langle S, R \rangle$ to handle malicious receivers, by adding the cut-and-choose (as well as the random tape generation) stage to enforce the adversary behaving honestly in *most* (Stage 2) OT executions. (This is in a similar spirit as the non-black-box approach of requiring the receiver to prove that it has behaved honestly.) Then the security against malicious receivers can be based on that against semi-honest receivers of $\langle S, R \rangle$.

Wee [Wee10] further augmented the stand-alone security of the protocol of [IKLP06, Hai08] to achieve parallel security, that is, obtaining a protocol that is secure against man-in-the-middle adversaries that simultaneously acts as sender and receiver in many parallel executions. Towards this, Wee instantiates the commitments in the coin-tossing sub-protocols of the protocol of [IKLP06, Hai08], with ones that satisfy a notion of “non-malleable w.r.t. extraction”. Roughly speaking, non-malleability w.r.t. extraction [Wee10] is a weaker notion than non-malleability of [DDN00, LPV08]; it guarantees that no matter what values the adversary is receiving commitments to, the committed values extracted out of the commitments from the adversary (with over-extraction) are indistinguishable. This guarantees that a simulator can bias the coin-tossing output by extracting the committed values from the adversary while the adversary cannot, as otherwise, by non-malleability w.r.t. extraction, it could do so even if the honest player sends a commitment to 0 instead of its true random challenge q . However, this is impossible as in this case no information of q is revealed. In other words, the coin-tossing protocol when instantiated with a non-malleable w.r.t. extraction commitment becomes parallel secure, relying which Wee shows that the OT protocol also becomes parallel secure.

Towards $\mathcal{H}$ -EUC-Secure OT protocols, we need to further overcome two problems.

First, we need to go from parallel security to concurrent security. In other words, we need coin-tossing protocols that are concurrently secure. Informally speaking, non-malleability w.r.t. extraction guarantees that the simulator can extract the committed values of commitments from the adversary (to bias the output of the coin-tossing) while keeping the commitment to the adversary hiding amid rewindings (to ensure that the adversary cannot bias the output). However, this only holds in the parallel setting, as non-malleability only guarantees hiding of a commitment when

values of the commitments from the adversary are extracted in parallel at the end of the execution. But, in the concurrent setting, the simulator needs to extract the committed values from the adversary in an on-line manner, that is, whenever the adversary successfully completes a commitment the committed value is extracted. To resolve this problem, we resort to CCA-secure commitments, which guarantees hiding of a commitment even when the committed values are extracted (via the committed-value oracle) concurrently and immediately after each commitment. Now, instantiating the commitment scheme in the coin-tossing protocols with a CCA-secure commitment yields a coin-tossing protocol that is concurrently secure.

The second problem is that to achieve $\mathcal{H}$ -EUC-security (similar to UC-security), we need to design a protocol that admits straight-line simulation. The simulator of a OT protocol has three tasks: It needs to simulate the messages of the honest sender and receiver, extract a choice from the adversary when it is acting as a receiver, and extract two inputs when it is acting as a sender. To achieve the first two tasks, the original simulation strategy in [IKLP06, Hai08, Wee10] uses rewindings to breaking the non-malleable commitments from the adversary to bias coin-tossing. When using CCA-secure commitments, the simulator can extract the committed values in a straight-line, by forwarding the commitment from the adversary to the helper functionality $\mathcal{H}$ that breaks the CCA commitments using brute force. For the last task, the original simulation strategy uses the simulator of the mS-OT protocol $\langle S, R \rangle$ against malicious senders to extract the adversary's inputs s_j^b 's in the Stage 3 OT executions, which then allows extraction of the real inputs v_0 and v_1 from the last message. However, the simulator of the mS-OT protocol may use rewindings. To solve this, one way is to simply assume a mS-OT protocol that has a straight-line simulator. We here however, present a different solution.

In our protocol, the sender and the receiver participate in parallel “coin tossing in the well” executions to decide the sender's random inputs s_j^b (and random tapes) in the Stage 3 OT executions (besides the receiver's inputs and random tapes). Since the simulator can bias the coin-tossing in a straight line, it can determine the sender's inputs s_j^b 's, which allows extraction of the sender's true inputs. For this to work, we need to make sure that a malicious sender would indeed uses the outputs of coin-tossing as inputs in the OT executions. Towards this, we again use the cut-and-choose technique: After the OT execution, the sender is required to reveal its randomness in the coin-tossing and OT execution at a randomly chosen subset of locations. The cut-and-choose technique guarantees that a malicious sender will behave consistently in most OT executions. Therefore the simulator extracts (through the coin-tossing executions) the inputs s_j^b 's correctly at *most* locations. However, in the protocol of [IKLP06, Hai08, Wee10], to recover the real inputs v_0 and v_1 , the simulator needs to obtain *all* s_j^b 's correctly. To bridge the gap, we modify the protocol to have the sender compute a random secret-sharing $\{a_j^b\}$ of each input v_b and hide each share using the appropriate s_j^b , that is, it sends $a_j^b \oplus s_j^{b \oplus \alpha}$ for every j (that is not open in the cut-and-choose procedures). Then, the simulator, able to extract most s_j^b 's correctly, can recover enough shares to decode to the real inputs correctly. In contrast, a malicious receiver that is enforced to behave honestly in most OT executions by the cut-and-choose procedure, cannot obtain enough shares for both inputs and thus can only recover one of them. Finally, we remark that as in [Wee10], to avoid over-extraction from the secret shares, we use the technique used in [CDSMW08, CDSMW09], which adds a another cut-and-choose procedure. A formal description of our OT protocol Π_{OT} that implements $\mathcal{F}_{OT}$ is provided in Figure 1; a formal proof of security of Π_{OT} (i.e., Lemma 2) appears in Appendix D.1.

OT protocol Π_{OT}

Inputs: The sender and receiver receive common input a security parameter 1^n and private inputs (v_0, v_1) and $u \in \{0, 1\}$ respectively.

Stage 1: The sender chooses a random subset $\Gamma_R \subseteq [20n]$ of size n and commits to Γ_R using $\langle C, R \rangle$.

The receiver chooses a random subset $\Gamma_S \subseteq [20n]$ of size n and another random subset $\Gamma \subseteq [18n]$ of size n ; it then commits to both Γ_S and Γ using $\langle C, R \rangle$.

Stage 2 (Coin-Tossing):

Receiver Random-Tape Generation: The receiver chooses $20n$ random strings $(a_1^R, \dots, a_{20n}^R)$ and commits to them using $\langle C, R \rangle$. The sender sends $20n$ random strings $(b_1^R, \dots, b_{20n}^R)$. The receiver calculates $r_i^R = a_i^R \oplus b_i^R$ for every $i \in [20n]$, and interprets r_i^R as $c_i \parallel \tau_i^R$, where c_i will be used as the receiver's input bit, and τ_i^R the random tape in the OT executions below.

Sender Random-Tape Generation: The sender chooses $20n$ random strings $(a_1^S, \dots, a_{20n}^S)$ and commits to them using $\langle C, R \rangle$. The receiver sends $20n$ random strings $(b_1^S, \dots, b_{20n}^S)$. The sender calculates $r_i^S = a_i^S \oplus b_i^S$ for every $i \in [20n]$, and interprets r_i^S as $s_i^0 \parallel s_i^1 \parallel \tau_i^S$, where s_i^0 and s_i^1 will be used as the sender's two input bits, and τ_i^S the random tape in the OT executions below.

Stage 3 (OT with Random Inputs): The sender and the receiver participates in $20n$ executions of the OT protocol $\langle S, R \rangle$ in parallel, where the sender acts as S and the receiver acts as R . In the i^{th} execution of $\langle S, R \rangle$, the sender uses inputs s_i^0, s_i^1 and random tape r_i^S and the receiver uses input c_i and random tape r_i^R . At the end of the execution, the receiver obtains outputs $\tilde{s}_1 \dots \tilde{s}_{20n}$.

Stage 4 (Cut-and-Choose—Honesty Checking):

Sender Honesty Checking: The receiver opens Γ_S and sender responds as follows: for every $j \in \Gamma_S$, the sender opens the j^{th} commitments of $\langle C, R \rangle$ in Stage 2 to $\tilde{a}_j^S$. The receiver checks if the openings are valid, and if for every $j \in \Gamma_S$, the sender acted honestly in the j^{th} OT execution according to $\tilde{a}_j^S \oplus b_j^S$. The receiver aborts if not.

Receiver Honesty Checking: The sender opens Γ_R and receiver responds as follows: for every $j \in \Gamma_R$, the receiver opens the j^{th} commitments of $\langle C, R \rangle$ in Stage 2 to $\tilde{a}_j^R$. The sender checks if the openings are valid and if for every $j \in \Gamma_R$, the receiver acted honestly in the j^{th} OT execution according to $\tilde{a}_j^R \oplus b_j^R$. The sender aborts if not.

Stage 5 (Combiner): Set $\Delta = [20n] - \Gamma_R - \Gamma_S$ (i.e., Δ is the set of unopened locations). For every $i \in \Delta$ The receiver computes $\alpha_i = u \oplus c_i$ and sends α_i . The sender responds as follows: It computes a $10n$ -out-of- $18n$ secret-sharing of v_0 ; without loss of generality, we index shares in that secret-sharing with elements in Δ ; let the secret-sharing be $\rho^0 = \{\rho_i^0\}_{i \in \Delta}$. Similarly, it also computes a $10n$ -out-of- $18n$ secret-sharing $\rho^1 = \{\rho_i^1\}_{i \in \Delta}$ for v_1 . Then the sender computes $\beta_i^b = \rho_i^b \oplus s_i^{b \oplus \alpha_i}$ for every $i \in \Delta$ and sends back all the β_i^b 's.

The receiver after receiving all the β_i^b 's, computes shares corresponding to the u^{th} input as $\tilde{\rho}_i = \beta_i^u \oplus \tilde{s}_i$ for every $i \in \Delta$, and sets $\tilde{\rho} = \{\tilde{\rho}_i\}_{i \in \Delta}$.

Stage 6 (Cut-and-Choose—Consistency Checking): The receiver opens to Γ . Then for every $j \in \Gamma \cap \Delta$, the sender reveals the two inputs $\hat{s}_j^0$ and $\hat{s}_j^1$ and random tape $\hat{\tau}_j^S$ that it uses in the j^{th} OT execution in Stage 3. The receiver checks if the sender acts honestly according to input $(\hat{s}_j^0, \hat{s}_j^1)$ and random tape $\hat{\tau}_j^S$ and aborts if not.

Finally the receiver checks whether $\tilde{\rho}$ computed in Stage 5 is $17n$ -close to a valid codeword w (that is, it agrees with w at $17n$ locations), and if for every $j \in \Gamma \cap \Delta$, w_j is equal to $\beta_j^u \oplus \hat{s}_j^{u \oplus \alpha_j}$. If so it outputs the value v encoded in w ; otherwise, it aborts.

Figure 1: Our OT protocol Π_{OT} that implements $\mathcal{F}_{\text{OT}}$ in the $\mathcal{H}$ -EUC model

4 Black-Box Robust CCA-Secure Commitments

In this section, we present a *black-box* construction of a robust CCA-secure commitment scheme w.r.t. committed-value oracle based on one-way functions. For simplicity of exposition, the presentation below relies on a non-interactive statistically binding commitment scheme `com`; this can be replaced with a standard 2-round statistically binding commitment scheme using standard techniques³. Our construction makes use of previous black-box constructions of extractable commitments and trapdoor commitment scheme. So let's start by reviewing them.

Extractable Commitments Intuitively, an extractable commitment is one such that for any machine C^* sending a commitment, a committed value can be extracted from C^* if the commitment it sends is valid; otherwise, if the commitment is invalid, then no guarantee is provided, that is, an arbitrary garbage value may be extracted. This is known as the “over-extraction” problem. (See Appendix A.7 for a formal definition.) As shown in [PW09], the following protocol used in the works of [DDN00, PRS02, Ros04] (also [Kil88]) yields a black-box extractable commitment scheme `ExtCom`: To commit to a value $v \in \{0, 1\}^m$, the committer and receiver on common input a security parameter 1^n , proceed as follows:

Commit: The committer finds n pairs of random shares $\{v_0^i, v_1^i\}_{i \in [n]}$ that sum up to v , (i.e., $v_0^i \oplus v_1^i = v$ for all $i \in [n]$) and commits to them in parallel using the non-interactive statistically binding commitment scheme `com`. Let c_b^i be the commitment to v_b^i .

Challenge: The receiver sends a n -bit string $ch \in \{0, 1\}^n$ sampled at random.

Reply: The committer opens commitments $c_{ch_i}^i$ for every $i \in [n]$.

To decommit, the sender sends v and opens the commitments to all n pairs of strings. The receiver checks whether all the openings are valid and also $v = v_0^i \oplus v_1^i$ for all i .

It is proved in [PW09] that `ExtCom` is extractable. Furthermore, the commitment scheme has the property that from any two accepting transcripts of the commit stage that has the same commit message but different challenge messages, the committed value can be extracted. This property is similar to the notion of special-soundness for interactive proof/argument systems; here we overload this notion, and refer to this special extractability property of `ExtCom` as special-soundness.

In our construction, we will actually need an extractable commitment scheme to a string $\sigma \in \{0, 1\}^m$ for which we can open any subset of the bits in σ without compromising the security (i.e. hiding) of the remaining bits. As shown in [PW09], we may obtain such a scheme `PExtCom` by running `ExtCom` to commit to each bit of σ in parallel. It is easy to see that `PExtCom` is also special-sound in the sense that, given two accepting transcripts of `PExtCom` that have the same commit message and two challenge messages that contain a pair of different challenges for every `ExtCom` commitment, the committed string σ can be extracted. We call such two transcripts a pair of admissible transcripts for `PExtCom`.

Trapdoor Commitments Roughly speaking, a trapdoor commitment scheme is a computationally binding and computationally hiding commitment scheme, such that, there exists a simulator that can generate a simulated commitment, and later open it to any value. (See Appendix A.8 for a formal definition.) Pass and Wee [PW09] presented a black-box trapdoor bit commitment scheme `TrapCom`. To commit to a bit σ , the committer and the receiver on common input 1^n do:

Stage 1: The receiver picks a random string challenge $e = (e_1, \dots, e_n)$ and commits to e using the non-interactive statistically binding commitment scheme `com`.

³This can be done by sending a first message of a 2-round commitment scheme at the beginning of the protocol, and using the second message of the 2-round commitment scheme w.r.t. that first message as a non-interactive commitment in the rest of the protocol.

Stage 2: The committer prepares $v_1, \dots, v_n$. Each v_i is a 2×2 0,1-matrix given by

$$\begin{pmatrix} v_i^{00} & v_i^{01} \\ v_i^{10} & v_i^{11} \end{pmatrix} = \begin{pmatrix} \eta_i & \eta_i \\ \sigma \oplus \eta_i & \sigma \oplus \eta_i \end{pmatrix}$$

where η_i is a random bit. The sender commits to $v_1, \dots, v_n$ using PExtCom (i.e., committing using ExtCom bit by bit in parallel). In addition, the sender prepares $(a_1^0, a_1^1), \dots, (a_n^0, a_n^1)$ where a_i^β is the opening to $v_i^{\beta 0}, v_i^{\beta 1}$ (i.e., either the top or bottom row of v_i).

Stage 3: The receiver opens to the challenge $e = (e_1, \dots, e_n)$; the sender responds with $a_1^{e_1}, \dots, a_n^{e_n}$.

To decommit, the sender sends σ . In addition, it chooses a random $\gamma \in \{0, 1\}$, sends γ , sends the openings to values $v_i^{0\gamma}, v_i^{1\gamma}$ for $i = 1, 2, \dots, n$ (i.e., either the left columns or the right columns of all the matrices). The receiver checks that all the openings are valid, and also that $\sigma = v_1^{0\gamma} \oplus v_1^{1\gamma} = \dots = v_n^{0\gamma} \oplus v_n^{1\gamma}$.

As shown in [PW09], the protocol TrapCom is trapdoor, following a Goldreich-Kahan [GK96] style proof; moreover, by running TrapCom in parallel, we obtain a trapdoor commitment scheme PTrapCom for multiple bits. Furthermore, since Stage 2 of the protocol TrapCom is simply an execution of PExtCom, given any two *admissible transcripts* of Stage 2, the matrices $v_1, \dots, v_n$ prepared in Stage 2 can be extracted; we show that from these matrices, the actual bit committed in the TrapCom commitment can be extracted, provided that the commitment is valid and has a unique committed value. We call this, again, the special-soundness of TrapCom. It is easy to see that the notion of special soundness (and admissible transcripts) can be easily extended for PTrapCom. The formal definition and proof of special-soundness of TrapCom (and PTrapCom) appear in Appendix B.1.

4.1 Overview of Our Construction

The CLP Construction: At a very high level, the CLP construction proceeds by having the committer first commit to the value v using a normal statistically binding commitment com, followed by a sequence of $\text{poly}(n)$ *WLSSP* proofs of the committed value. The *WLSSP* proofs are the non-black-box component of the CLP construction, but are crucial for achieving CCA-security. Recall that proving CCA-security w.r.t. $\mathcal{O}$ amounts to showing that the views of A in experiments IND₀ and IND₁ are indistinguishable (when A has oracle access to $\mathcal{O}$). Let us refer to the adversary's interaction with C as the *left interaction*, and its interactions with $\mathcal{O}$ as the *right interactions*. The main hurdle in showing the indistinguishability of IND₀ and IND₁ is that the oracle $\mathcal{O}$ is not efficiently computable; if it were, indistinguishability would directly follow from the hiding property of the left interaction. The main idea of the security proof of [CLP10] is then to implement the oracle $\mathcal{O}$ by extracting the committed values from the adversary, via “rewinding” the special-sound proofs in the right interactions. The two main technical challenges arises in simulating oracle $\mathcal{O}$.

First, once the simulation starts rewinding the right interactions, A might send new messages also in the left interaction. So, if done naively, this would rewind the left interaction, which could violate its hiding property. To solve this problem, the CLP protocol schedules messages in the special-sound proofs using a special message scheduling (according to the identity of the commitment), called the CLP scheduling, which is a variant of the message scheduling technique of [DDN00, LPV08]. The special message scheduling ensures that for every accepting right interaction with an identity that is different from the left interaction, there exists many points—called **safe-points**—in the interaction, from which one can rewind the right interaction without requesting any *new* message in the left interaction.

Second, in the experiment IND_b, the adversary A expects to receive the committed value at the very moment it completes a commitment to its oracle. If the adversary “nests” its oracle

calls, these rewindings become recursive and the running-time of the extraction quickly becomes exponential. To avoid the extraction time from exploding, the simulation strategy in CLP rewinds from **safe-points** using a concurrent extraction strategy that is similar to that used in the context of concurrent $\mathcal{ZK}$ by Richardson and Killian [RK99].

New Approach: To obtain a black-box construction, our main goal is to replace the *WISSP* proofs with an “equivalent” black-box component. The key property that the CLP proof relies on is that the protocol contains many *3-round* constructs satisfying that rewinding the last two messages reveals the committed value, but rewinding three messages reveals nothing. It seems that the 3-round commitment scheme **PExtCom** is a good replacement of *WISSP* proofs as one such 3-round construct: The special-soundness property of **PExtCom** ensures that rewinding the last two messages reveals the committed value, and the hiding property ensures that rewinding three messages reveals nothings. It is thus tempting to consider a commitment scheme in which the committer commits to value v using $\text{poly}(n)$ invocations of **PExtCom**, arranged according to the CLP scheduling; the CLP extraction strategy guarantees that for every accepting right interaction, (the last two messages of) one **PExtCom** commitment is rewound and a committed value is extracted. Indeed, if a commitment of this scheme is valid, meaning that all the **PExtCom** commitments contained in it are valid commitments to the same value, the CLP extraction strategy returns the unique committed value. However, if the commitment is invalid, there arises the over-extraction problem: The CLP extraction strategy may extract a garbage value from an invalid **PExtCom** commitment or from a valid commitment that is inconsistent with the other commitments.

To solve the over-extraction problem, we use the cut-and-choose technique to enforce the committer to give valid and consistent **PExtCom** commitments. Instead of having the committer commit to v directly, let it commit to a $(n + 1)$ -out-of- $10n$ Shamir’s secret sharing $s_1, \dots, s_{10n}$ of v using many **PExtCom** invocations, still arranged according to the CLP scheduling; we refer to all the commitments to the j^{th} share s_j the j^{th} column. After all the **PExtCom** commitments, the receiver requests the committer to open all the commitments in n randomly chosen columns; the receiver accepts only if each column contains valid commitments to the same value. It follows from the cut-and-choose technique that except with negligible probability, at most n columns may contain invalid or inconsistent commitments. Therefore, when applying the CLP extraction strategy on a commitment of this scheme, it guarantees to extract out a secret-sharing that is .9-close to all the secret-sharing committed to in this commitment. Then by relying on the error-correcting property of the secret sharing, a valid committed value can be reconstructed. The formal analysis is actually more subtle; to avoid over-extraction, we employ the technique used in [CDSMW08, CDSMW09, Wee10], which involves setting the validity condition of the commitment scheme carefully so that invalid commitment can be identified. We refer the reader to Appendix B.3 for more details.

Unfortunately, our use of the cut-and-choose technique brings another problem: The above commitment scheme may not be hiding. This is because, in the last stage, the receiver may request the committer to open an adaptively chosen subset of commitments of **PExtCom**, the remaining unopened commitments may not be hiding, unless **PExtCom** were secure against selective opening attack. To resolve this problem, we use the trapdoor commitment scheme **PTrapCom** to replace **PExtCom**. Since **PTrapCom** is trapdoor, it is secure against selective opening attack, and thus the hiding property holds. Furthermore, since Stage 2 of **PTrapCom** is simply a commitment of **PExtCom**, we can use Stage 2 of **PTrapCom** as an implementation of the 3-round construct needed for the CLP scheduling and extraction strategy. More precisely, the commitment scheme proceeds as follow: The committer commits to a $(n + 1)$ -out-of- $10n$ secret sharing of the value v using many invocations of **PTrapCom**, where all the invocations share the same Stage 1 message sent at the beginning, followed by all the 3-round Stage 2 executions arranged according to the CLP scheduling, and then all the Stage 3 executions performed in parallel; finally, the committer and the receiver conducts a cut-and-choose consistency check as described above.

It seems that the security proof of our CCA-secure commitment should follow from that of the non-black-box construction of [CLP10]. Unfortunately, due to the fact that the “rewinding slots” of our protocol, that is the commitment of **ExtCom**, may have over-extraction, whereas the *WTSSP* proofs in the CLP protocol never has this problem, the technical proof of [CLP10] does not go through; and in Appendix B we rely on a different analysis to show the security of our protocol. A formal description of our CCA secure protocol $\langle C, R \rangle$ in Figure 2.

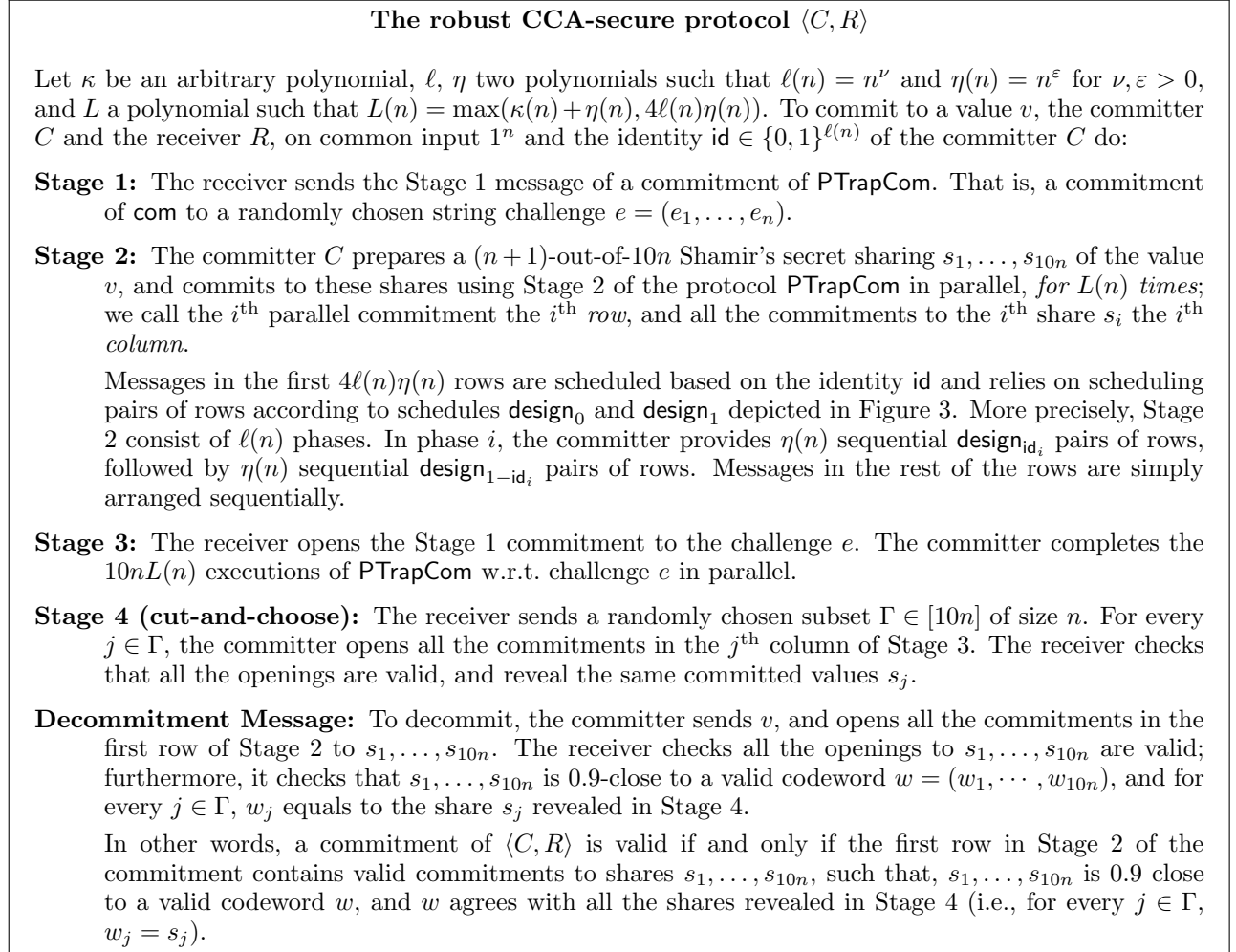


Figure 2: The formal description of the $\kappa(n)$ -robust CCA-secure protocol $\langle C, R \rangle$

References

- [BCC88] Gilles Brassard, David Chaum, and Claude Crépeau. Minimum disclosure proofs of knowledge. *J. Comput. Syst. Sci.*, 37(2):156–189, 1988.
- [BCNP04] Boaz Barak, Ran Canetti, Jesper Buus Nielsen, and Rafael Pass. Universally composable protocols with relaxed set-up assumptions. In *FOCS*, pages 186–195, 2004.
- [BOGW88] Michael Ben-Or, Shafi Goldwasser, and Avi Wigderson. Completeness theorems for non-cryptographic fault-tolerant distributed computation (extended abstract). In *STOC*, pages 1–10, 1988.

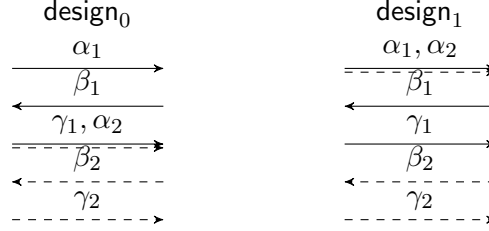


Figure 3: Description of the schedules used in Stage 2 of the protocol. $(\alpha_1, \beta_1, \gamma_1)$ and $(\alpha_2, \beta_2, \gamma_2)$ are respectively the transcripts of a pair of rows in Stage 2 of the protocol $\langle C, R \rangle$.

- [BS05] Boaz Barak and Amit Sahai. How to play almost any mental game over the net - concurrent composition via super-polynomial simulation. In *FOCS*, pages 543–552, 2005.
- [Can00] Ran Canetti. Security and composition of multiparty cryptographic protocols. *Journal of Cryptology*, pages 143–202, 2000.
- [Can01] Ran Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *FOCS*, pages 136–145, 2001.
- [Can04] Ran Canetti. Universally composable signature, certification, and authentication. In *CSFW*, pages 219–, 2004.
- [CDPW07] Ran Canetti, Yevgeniy Dodis, Rafael Pass, and Shabsi Walfish. Universally composable security with global setup. In *TCC*, pages 61–85, 2007.
- [CDSMW08] Seung Geol Choi, Dana Dachman-Soled, Tal Malkin, and Hoeteck Wee. Black-box construction of a non-malleable encryption scheme from any semantically secure one. In *TCC*, pages 427–444, 2008.
- [CDSMW09] Seung Geol Choi, Dana Dachman-Soled, Tal Malkin, and Hoeteck Wee. Simple, black-box constructions of adaptively secure protocols. In *TCC*, pages 387–402, 2009.
- [CF01] Ran Canetti and Marc Fischlin. Universally composable commitments. In *CRYPTO*, pages 19–40, 2001.
- [CKL03] Ran Canetti, Eyal Kushilevitz, and Yehuda Lindell. On the limitations of universally composable two-party computation without set-up assumptions. In *EUROCRYPT*, pages 68–86, 2003.
- [CLOS02] Ran Canetti, Yehuda Lindell, Rafail Ostrovsky, and Amit Sahai. Universally composable two-party and multi-party secure computation. In *STOC*, pages 494–503, 2002.
- [CLP10] Ran Canetti, Huijia Lin, and Rafael Pass. Adaptive hardness and composable security in the plain model from standard assumptions. In *FOCS*, pages 541–550, 2010.
- [CPS07] Ran Canetti, Rafael Pass, and Abhi Shelat. Cryptography from sunspots: How to use an imperfect reference string. In *FOCS*, pages 249–259, 2007.
- [DDN00] Danny Dolev, Cynthia Dwork, and Moni Naor. Nonmalleable cryptography. *SIAM Journal on Computing*, 30(2):391–437, 2000.

- [DI05] Ivan Damgård and Yuval Ishai. Constant-round multiparty computation using a black-box pseudorandom generator. In *CRYPTO*, pages 378–394, 2005.
- [DNS04] Cynthia Dwork, Moni Naor, and Amit Sahai. Concurrent zero-knowledge. *J. ACM*, 51(6):851–898, 2004.
- [EGL85] Shimon Even, Oded Goldreich, and Abraham Lempel. A randomized protocol for signing contracts. *Commun. ACM*, 28(6):637–647, 1985.
- [FS90] Uriel Feige and Adi Shamir. Witness indistinguishable and witness hiding protocols. In *STOC*, pages 416–426, 1990.
- [GGJS12] Sanjam Garg, Vipul Goyal, Abhishek Jain, and Amit Sahai. Concurrently secure computation in constant rounds. To appear in *EUROCRYPT* 2012, 2012.
- [GK96] Oded Goldreich and Ariel Kahan. How to construct constant-round zero-knowledge proof systems for NP. *Journal of Cryptology*, 9(3):167–190, 1996.
- [GM84] Shafi Goldwasser and Silvio Micali. Probabilistic encryption. *J. Comput. Syst. Sci.*, 28(2):270–299, 1984.
- [GMR89] Shafi Goldwasser, Silvio Micali, and Charles Rackoff. The knowledge complexity of interactive proof systems. *SIAM Journal on Computing*, 18(1):186–208, 1989.
- [GMW87] Oded Goldreich, Silvio Micali, and Avi Wigderson. How to play any mental game or a completeness theorem for protocols with honest majority. In *STOC*, pages 218–229, 1987.
- [GMW91] Oded Goldreich, Silvio Micali, and Avi Wigderson. Proofs that yield nothing but their validity or all languages in NP have zero-knowledge proof systems. *J. ACM*, 38(3):690–728, 1991.
- [GO07] Jens Groth and Rafail Ostrovsky. Cryptography in the multi-string model. In *CRYPTO*, pages 323–341, 2007.
- [Gol01] Oded Goldreich. *Foundations of Cryptography — Basic Tools*. Cambridge University Press, 2001.
- [Goy11] Vipul Goyal. Constant round non-malleable protocols using one way functions. In *STOC*, pages 695–704, 2011.
- [Hai08] Iftach Haitner. Semi-honest to malicious oblivious transfer - the black-box way. In *TCC*, pages 412–426, 2008.
- [HILL99] Johan Håstad, Russell Impagliazzo, Leonid Levin, and Michael Luby. A pseudorandom generator from any one-way function. *SIAM Journal on Computing*, 28:12–24, 1999.
- [IKLP06] Yuval Ishai, Eyal Kushilevitz, Yehuda Lindell, and Erez Petrank. Black-box constructions for secure computation. In *STOC*, pages 99–108, 2006.
- [IPS08] Yuval Ishai, Manoj Prabhakaran, and Amit Sahai. Founding cryptography on oblivious transfer - efficiently. In *CRYPTO*, pages 572–591, 2008.
- [Kat07] Jonathan Katz. Universally composable multi-party computation using tamper-proof hardware. In *EUROCRYPT*, pages 115–128, 2007.

- [Kil88] Joe Kilian. Founding cryptography on oblivious transfer. In *STOC*, pages 20–31, 1988.
- [Kil92] Joe Kilian. A note on efficient zero-knowledge proofs and arguments (extended abstract). In *STOC*, pages 723–732, 1992.
- [KLP05] Yael Tauman Kalai, Yehuda Lindell, and Manoj Prabhakaran. Concurrent general composition of secure protocols in the timing model. In *STOC*, pages 644–653, 2005.
- [Lin04] Yehuda Lindell. Lower bounds for concurrent self composition. In *TCC*, pages 203–222, 2004.
- [LP07] Yehuda Lindell and Benny Pinkas. An efficient protocol for secure two-party computation in the presence of malicious adversaries. In *EUROCRYPT*, pages 52–78, 2007.
- [LPV08] Huijia Lin, Rafael Pass, and Muthuramakrishnan Venkitasubramaniam. Concurrent non-malleable commitments from any one-way function. In *TCC*, pages 571–588, 2008.
- [LPV09] Huijia Lin, Rafael Pass, and Muthuramakrishnan Venkitasubramaniam. A unified framework for concurrent security: universal composability from stand-alone non-malleability. In *STOC*, pages 179–188, 2009.
- [LPV12] Huijia Lin, Rafael Pass, and Muthuramakrishnan Venkitasubramaniam. Uc from semi-honest ot. Manuscript, 2012.
- [MMY06] Tal Malkin, Ryan Moriarty, and Nikolai Yakovenko. Generalized environmental security from number theoretic assumptions. In *TCC*, pages 343–359, 2006.
- [Nao91] Moni Naor. Bit commitment using pseudorandomness. *Journal of Cryptology*, 4:151–158, 1991.
- [Pas03a] Rafael Pass. Simulation in quasi-polynomial time, and its application to protocol composition. In *EUROCRYPT*, pages 160–176, 2003.
- [Pas03b] Rafael Pass. Simulation in quasi-polynomial time, and its application to protocol composition. In *EUROCRYPT*, pages 160–176, 2003.
- [PR05] Rafael Pass and Alon Rosen. Concurrent non-malleable commitments. In *FOCS*, pages 563–572, 2005.
- [PRS02] Manoj Prabhakaran, Alon Rosen, and Amit Sahai. Concurrent zero knowledge with logarithmic round-complexity. In *FOCS*, pages 366–375, 2002.
- [PS04] Manoj Prabhakaran and Amit Sahai. New notions of security: achieving universal composability without trusted setup. In *STOC*, pages 242–251, 2004.
- [PW09] Rafael Pass and Hoeteck Wee. Black-box constructions of two-party protocols from one-way functions. In *TCC*, pages 403–418, 2009.
- [Rab05] Michael O. Rabin. How to exchange secrets with oblivious transfer. Cryptology ePrint Archive, Report 2005/187, 2005.
- [RK99] Ransom Richardson and Joe Kilian. On the concurrent composition of zero-knowledge proofs. In *Eurocrypt*, pages 415–432, 1999.

- [Ros04] Alon Rosen. A note on constant-round zero-knowledge proofs for np. In *TCC*, pages 191–202, 2004.
- [Wee10] Hoeteck Wee. Black-box, round-efficient secure computation via non-malleability amplification. To appear in FOCS 2010, 2010.
- [Xia11] David Xiao. (nearly) round-optimal black-box constructions of commitments secure against selective opening attacks. In *TCC*, pages 541–558, 2011.
- [Yao86] Andrew Chi-Chih Yao. How to generate and exchange secrets (extended abstract). In *FOCS*, pages 162–167, 1986.

A General Definitions

A.1 Witness Relations

We recall the definition of a witness relation for a $\mathcal{NP}$ language [Gol01].

Definition 3 (Witness relation). *A witness relation for a language $L \in \mathcal{NP}$ is a binary relation R_L that is polynomially bounded, polynomial time recognizable and characterizes L by $L = \{x : \exists y \text{ s.t. } (x, y) \in R_L\}$*

We say that y is a witness for the membership $x \in L$ if $(x, y) \in R_L$. We will also let $R_L(x)$ denote the set of witnesses for the membership $x \in L$, i.e., $R_L(x) = \{y : (x, y) \in R_L\}$. In the following, we assume a fixed witness relation R_L for each language $L \in \mathcal{NP}$.

A.2 Indistinguishability

Definition 4 (Computational Indistinguishability). *Let Y be a countable set. Two ensembles $\{A_{n,y}\}_{n \in \mathbb{N}, y \in Y}$ and $\{B_{n,y}\}_{n \in \mathbb{N}, y \in Y}$ are said to be **computationally indistinguishable** (denoted by $\{A_{n,y}\}_{n \in \mathbb{N}, y \in Y} \approx \{B_{n,y}\}_{n \in \mathbb{N}, y \in Y}$), if for every PPT “distinguishing” machine D , there exists a negligible function $\nu(\cdot)$ so that for every $n \in \mathbb{N}, y \in Y$:*

$$|\Pr[a \leftarrow A_{n,y} : D(1^n, y, a) = 1] - \Pr[b \leftarrow B_{n,y} : D(1^n, y, b) = 1]| < \nu(n)$$

A.3 Interactive Proofs

We use the standard definitions of interactive proofs (and interactive Turing machines) [GMR89] and arguments (a.k.a computationally-sound proofs) [BCC88]. Given a pair of interactive Turing machines, P and V , we denote by $\langle P(w), V \rangle(x)$ the random variable representing the (local) output of V , on common input x , when interacting with machine P with private input w , when the random input to each machine is uniformly and independently chosen.

Definition 5 (Interactive Proof System). *A pair of interactive machines $\langle P, V \rangle$ is called an **interactive proof system** for a language L if there is a negligible function $\nu(\cdot)$ such that the following two conditions hold :*

- **Completeness:** *For every $x \in L$, and every $w \in R_L(x)$, $\Pr[\langle P(w), V \rangle(x) = 1] = 1$*
- **Soundness:** *For every $x \notin L$, and every interactive machine B , $\Pr[\langle B, V \rangle(x) = 1] \leq \nu(|x|)$*

*In case that the soundness condition is required to hold only with respect to a computationally bounded prover, the pair $\langle P, V \rangle$ is called an **interactive argument system**.*

A.4 Witness Indistinguishable Proofs

The notion of *witness indistinguishability* ($\mathcal{WI}$) was introduced by Feige and Shamir in [FS90]. Roughly speaking, an interactive proof is said to be $\mathcal{WI}$ if the verifier's output is "computationally independent" of the witness used by the prover for proving the statement. In this context, we focus on languages $L \in \mathcal{NP}$ with a corresponding witness relation R_L . Namely, we consider interactions in which, on common input x , the prover is given a witness in $R_L(x)$. By saying that the output is computationally independent of the witness, we mean that for any two possible $\mathcal{NP}$ -witnesses that could be used by the prover to prove the statement $x \in L$, the corresponding outputs are computationally indistinguishable.

Definition 6 (Witness-indistinguishability). *Let $\langle P, V \rangle$ be an interactive proof system for a language $L \in \mathcal{NP}$. We say that $\langle P, V \rangle$ is witness-indistinguishable for R_L , if for every $\mathcal{PPT}$ ITM V^* and for every two sequences $\{w_{n,x}^1\}_{n \in N, x \in L \cap \{0,1\}^n}$ and $\{w_{n,x}^2\}_{n \in N, x \in L \cap \{0,1\}^n}$, such that $w_{n,x}^1, w_{n,x}^2 \in R_L(x)$ for every x , the following probability ensembles are computationally indistinguishable.*

- $\{\langle P(w_{n,x}^1), V^*(z) \rangle(x)\}_{n \in N, x \in L \cap \{0,1\}^n, z \in \{0,1\}^*}$
- $\{\langle P(w_{n,x}^2), V^*(z) \rangle(x)\}_{n \in N, x \in L \cap \{0,1\}^n, z \in \{0,1\}^*}$

A.5 Special-sound $\mathcal{WI}$ proofs

A 4-round public-coin interactive proof for the language $L \in \mathcal{NP}$ with witness relation R_L is *special-sound* with respect to R_L , if for any two transcripts $(\delta, \alpha, \beta, \gamma)$ and $(\delta', \alpha', \beta', \gamma')$ such that the initial two messages, δ, δ' and α, α' , are the same but the challenges β, β' are different, there is a deterministic procedure to extract the witness from the two transcripts and runs in polynomial time. In this paper, we rely on special sound proofs that are also witness indistinguishable ($\mathcal{WI}$) Special-sound $\mathcal{WI}$ proofs for languages in $\mathcal{NP}$ can be based on the existence of 2-round commitment schemes, which in turn can be based on one-way functions [GMW91, FS90, HILL99, Nao91].

A.6 Concurrent $\mathcal{ZK}$ Protocols

Let $\langle P, V \rangle$ be an interactive proof for a language L . Consider a concurrent adversarial verifier V^* that on common input a security parameter 1^n , a statement $x \in \{0,1\}^n$ and auxiliary input z , interacts with $m(n)$ independent copies of P concurrently, without any restrictions over the scheduling of the messages in the different interactions with P .

Definition 7. *Let $\langle P, V \rangle$ be an interactive proof system for a language L . We say that $\langle P, V \rangle$ is black-box concurrent zero-knowledge if for every polynomials q and m , there exists a probabilistic polynomial time algorithm $S_{q,m}$, such that for every concurrent adversary V^* that on common input 1^n , x and auxiliary input z opens up $m(n)$ executions and has a running-time bounded by $q(n)$, $S_{q,m}(1^n, x, z)$ runs in time polynomial in n . Furthermore, it holds that the following ensembles are computationally indistinguishable*

- $\{\text{view}_{V^*}[\langle P(w), V^*(z) \rangle(1^n, x)]\}_{n \in N, x \in L \cap \{0,1\}^n, w \in R_L(x), z \in \{0,1\}^*}$
- $\{S_{q,m}(1^n, x, z)\}_{n \in N, x \in L \cap \{0,1\}^n, w \in R_L(x), z \in \{0,1\}^*}$

A.7 Extractable Commitments

We recall the definition of extractable commitments defined in [PW09]. Let $\langle C_s, R_s \rangle$ be a statistically binding commitment scheme. We say that $\langle C_s, R_s \rangle$ is an extractable commitment scheme if there exists an expected polynomial-time probabilistic oracle machine (the extractor) E that given oracle access to any $\mathcal{PPT}$ cheating sender C^* outputs a pair (τ, σ^*) such that:

- (simulation) τ is identically distributed to the view of C^* at the end of interacting with an honest receiver R_s in commit phase.
- (extraction) the probability that τ is *accepting and valid* $\sigma^* = \perp$ is negligible.
- (binding) if $\sigma^* \neq \perp$, then it is statistically impossible to open τ to any value other than σ^* .

We will also consider extractable commitment schemes that are computationally binding; the definition is as above, except if $\sigma^* \neq \perp$, we only require that it is computationally infeasible to open τ to any value other than σ^* .

A.8 Trapdoor Commitments

We say that $\langle C_t, R_t \rangle$ is a trapdoor commitment scheme if there exists an expected polynomial-time probabilistic oracle machine $\mathcal{S} = (\mathcal{S}_1, \mathcal{S}_2)$ such that for any PPT R^* and all $v \in \{0, 1\}^n$, the output (τ, w) of the following experiments are computationally indistinguishable:

- an honest sender C_t interacts with R^* to commit to v , and then opens the commitment: τ is the view of R^* in the commit phase, and w is the message C_t sends in the open phase.
- the simulator $\mathcal{S}$ generates a simulated view τ for the commit phase, and then opens the commitment to v in the open phase: formally, $\mathcal{S}_1^{R^*}(1^n, 1^k) \rightarrow (\tau, \text{STATE})$, $\mathcal{S}_2(\text{STATE}, v) \rightarrow w$.

B Proofs for Our Black-Box Robust CCA-Secure Commitments

B.1 Properties of the Trapdoor Commitments TrapCom

Before proving the security properties of our robust CCA-secure commitment scheme, we first prove a few properties of the trapdoor commitment scheme TrapCom of [PW09], which will be very instrumental the proof of robust CCA-security.

Special-soundness of TrapCom: Since Stage 2 of the protocol TrapCom is simply a PExtCom commitment, given any two *admissible transcripts* of Stage 2, a committed value can be extracted. Consider the following deterministic polynomial time procedure **reconst** that on input two *admissible transcripts* $\mathcal{T}_1, \mathcal{T}_2$ of Stage 2 extracts a committed value as follows: It first reconstructs all the matrices $v_1, \dots, v_n$ from $\mathcal{T}_1, \mathcal{T}_2$ by relying on the extractability of PExtCom; then it checks whether all the left columns of the matrices sum up to the same bit b , and sets σ_0 to b if this is the case and $\perp$ otherwise; it computes σ_1 similarly with respect to the right columns; next,

- If σ_1 and σ_1 equal to two different bits, **reconst** outputs **err**.
- Else if σ_0 and σ_1 both equal to $\perp$, it outputs $\perp$ as well.
- Finally, if σ_0 and σ_1 equal to the same bit b , or only one of them equals to b and the other equals to $\perp$, **reconst** outputs b .

We show below in Lemma 3 that as long as the **reconst** procedure does not output **err**, then the extracted value must be the committed value—we call this the **special-soundness** property of TrapCom—and in Lemma 4 that when the **reconst** procedure outputs **err**, then the receiver's challenge in the TrapCom commitment can be computed efficiently. It follows essentially from Lemma 3 and 4 that TrapCom is computationally hiding. Formally, let $\mathcal{T}$ be a commitment of TrapCom, we say that a pair of admissible transcripts $\mathcal{T}_1, \mathcal{T}_2$ is consistent with TrapCom if the first message in $\mathcal{T}_1$ and $\mathcal{T}_2$ equals to the first message of Stage 2 in $\mathcal{T}$. Then,

Lemma 3 (Special-soundness of TrapCom). *Let $\mathcal{T}$ be a commitment of TrapCom, and $\mathcal{T}_1, \mathcal{T}_2$ a pair of admissible transcripts of TrapCom that is consistent with $\mathcal{T}$. Then, if $\text{reconst}(\mathcal{T}_1, \mathcal{T}_2) = \sigma \neq \text{err}$, it is statistically impossible to open $\mathcal{T}$ to any value other than σ .*

Proof. Assume for contradiction that $\text{reconst}(\mathcal{T}_1, \mathcal{T}_2)$ outputs $\sigma \neq \text{err}$ but there exists a decommitment that opens $\mathcal{T}$ to a bit $\sigma' \in \{0, 1\}$ different from σ .

It follows from the validity condition of TrapCom that if a commitment can be opened to σ' there must exist a $\gamma \in \{0, 1\}$, such that all the commitments of ExtCom to $v_i^{0\gamma}, v_i^{1\gamma}$ for $i = 1, 2, \dots, k$ are valid and $\sigma' = v_1^{0\gamma} \oplus v_1^{1\gamma} = \dots = v_k^{0\gamma} \oplus v_k^{1\gamma}$, where σ is the committed value. That is, either all the left columns or the right columns are valid commitments to values that sum up to σ' . Since two admissible transcripts of TrapCom contains a pair of admissible transcripts of ExtCom for each commitment to a bit $v_j^{b_1 b_2}$. It follows from the extractability of ExtCom that from $\mathcal{T}_1$ and $\mathcal{T}_2$, values $v_i^{0\gamma}, v_i^{1\gamma}$ can be extracted correctly, as by the validity condition commitments to $v_i^{b\gamma}$'s are all valid. Therefore the procedure reconst will set bit σ_γ to σ' . Then, conditioned on reconst does not output err , it must output σ' , which gives a contradiction. $\square$

Lemma 4. *Let $\mathcal{T}$ be an accepting commitment of TrapCom, and $\mathcal{T}_1, \mathcal{T}_2$ a pair of admissible transcripts of TrapCom that is consistent with $\mathcal{T}$. Then, if $\text{reconst}(\mathcal{T}_1, \mathcal{T}_2) = \text{err}$, the receiver's challenge committed to in Stage 1 of $\mathcal{T}$ can be computed efficiently and deterministically from $\mathcal{T}_1, \mathcal{T}_2$.*

Proof. The reconst procedure, on input $\mathcal{T}_1, \mathcal{T}_2$, reconstructs a tuple of n matrices $\tilde{v}_1, \dots, \tilde{v}_n$ by relying on the special soundness of ExtCom, and outputs err if and only if all the values $\tilde{v}_j^{00}, \tilde{v}_j^{10}$ in the left columns sum up to a bit b whereas all the values $\tilde{v}_j^{01}, \tilde{v}_j^{11}$ in the right columns sum up to $1 - b$. Furthermore, since $\mathcal{T}$ is accepting, in Stage 3 of $\mathcal{T}$, the receiver must successfully open the stage 1 com commitment to a challenge e and the committer must successfully open the two commitments in the e_j^{th} row of the j^{th} matrices to the same value η_j for every $j \in [k]$. Thus, by the special soundness of ExtCom, values extracted from the e_j^{th} row $v_j^{e_j 0}, v_j^{e_j 1}$ must equal to η_j , which means the two bits $v_j^{(1-e_j)0}, v_j^{(1-e_j)1}$ extracted from the $(1 - e_j)^{\text{th}}$ row must differ. Thus from $\mathcal{T}_1$ and $\mathcal{T}_2$, the receiver's challenge e in $\mathcal{T}$ can be computed efficiently. $\square$

Strong Computational Binding: We show that TrapCom enjoys a strong computational binding property as described in Lemma 5.

Lemma 5 (Strong computational binding). *For every PPT machine C^* , the probability that C^* in interaction with the honest receiver of TrapCom, generates a commitment that has more than one valid committed values is negligible.*⁴

Proof. Assume for contradiction that there is a committer C^* that can generate a commitment that has two valid committed values with polynomial probability. Then we can construct a machine A that violates the hiding property of com.

The machine A on input a com commitment c to a random n -bit string e , internally incorporates C^* and emulates the messages from the receiver for C^* honestly, except that: it forwards c to A at the Stage 1 message; after C^* completes Stage 2, it repeatedly rewinds C^* from the challenge message in Stage 2 by sending C^* freshly sampled challenges, until another accepting transcript of Stage 2 is obtained; then it checks whether the pair of transcripts of Stage 2 is admissible and if so whether reconst outputs err on input these two transcripts; if this is the case, it computes the challenge e' from the two admissible transcripts by Lemma 4 and outputs e' ; otherwise, it aborts. Finally, A cuts its execution off after $2^{n/3}$ steps.

⁴Note that the computational binding property only guarantees that it is impossible for an efficient committer to generate a commitment of TrapCom and opens it to two different values.

It follows from standard technique that the expected running time of C^* is bounded by a polynomial. Furthermore, each challenge in Stage 2 of a **TrapCom** commitment is a n -tuple of n -bit strings. Then Since A runs for at most $2^{n/3}$ steps, the probability that any n -bit string is picked for a second time in A is $2^{n/3}/2^n$; since A picks at most $2^{n/3}$ strings, by union bound, the probability that any n -bit string is picked twice is $\frac{1}{2^{n/3}} = 2^{n/3} \frac{2^{n/3}}{2^n}$. Therefore, except with negligible probability, the pair of accepting transcripts collected by A is also admissible.

Next, we show that conditioned on that the pair of transcripts collected by A is admissible, A outputs the value committed to in c with polynomial probability. Since A emulates the execution of C^* perfectly, with polynomial probability C^* in A generates a commitment that can be opened to both 0 and 1. When this happens, by the validity condition of **TrapCom**, the commitment generated by C^* must have the property that all the commitments of **ExtCom** in Stage 2 are valid, and the committed values in the left columns sum up to a bit b whereas the committed values in the right columns sum up to another bit $1 - b$. In this case, the procedure **reconst** fails to extract a value from the pair of admissible transcripts collected by A and outputs **err**. The by Lemma 4 the challenge committed to in Stage 1 can be computed. Thus A outputs the committed value of c with polynomial probability. $\square$

Extension to multiple bits. As shown in [PW09], by running the trapdoor bit commitment scheme **TrapCom** in parallel, we obtain a trapdoor commitment scheme **PTrapCom** for multiple bits, with the additional property that we can open the commitment to any subset of the bits without compromising the security of the remaining bits. The hiding, binding and trapdoor property of the commitment remains. Furthermore, Stage 2 of the protocol **PTrapCom** consists of many parallel executions of **PExtCom**. We say that two transcripts of Stage 2 of **PTrapCom** are *admissible* if they contain a pair of admissible transcripts of **PTrapCom** for each parallel execution in it. Then given a pair of admissible transcripts of **PTrapCom**, the committed string can be extracted by running the following procedure **reconst**: For each parallel execution, it extracts a value σ_i using the **reconst** procedure; then it outputs **err** if any σ_i equals to **err**, otherwise, it outputs all the extracted bits σ_i 's concatenated. Again, we call this property the special-soundness of **PTrapCom**.

B.2 $\langle C, R \rangle$ is a Statistically-Binding Commitment Scheme

In this section, we provide formally that that $\langle C, R \rangle$ is a statistically binding commitment scheme.

Proposition 2. *$\langle C, R \rangle$ is a statistically binding commitment scheme.*

Proof. The statistically binding property of $\langle C, R \rangle$ follows directly from that of **com**. We then, focus on showing the hiding property.

Recall that the commitment scheme **TrapCom** is trapdoor. In particular, as shown in [PW09], if the receiver's challenge is fixed, then there is a straight-line simulator that can generate a simulated transcript of the commit phase that can be equivocated to both 0 and 1 later. We recall the simulation strategy. Let R^* be a malicious receiver using a fixed challenge e , then to simulate Stage 2 and 3 of **TrapCom** for R^* , the simulator samples a random bit γ and prepares $v_1 \cdots v_n$ where each v_i is a 2×2 0,1-matrix such that, the e_i^{th} row of v_i has the form (η_i, η_i) and the $(1 - e_i)^{\text{th}}$ row has the form $(\gamma \oplus \eta_i, (1 - \gamma) \oplus \eta_i)$ with a randomly and independently sampled bit η_i . It then commits to $v_1, \dots, v_n$ using **PExtCom** in Stage 2; later, in Stage 3, upon receiving challenge e , for every i , it opens commitments to the e_i^{th} row in the i^{th} matrix to (η_i, η_i) , yielding an accepting commitment. To equivocate the simulated commitment to 0, the simulator sends γ and opens all the commitments in the γ^{th} column of the matrices; to equivocate the commitment to 1, it sends $1 - \gamma$ and opens the $(1 - \gamma)^{\text{th}}$ column of the matrices. It follows from the hiding property of **ExtCom** (recall that a commitment of **PExtCom** to $v_1 \cdots v_n$ is simply many commitments of **ExtCom** to bits

$v_j^{b_1, b_2}$ in parallel) that, for every $b \in \{0, 1\}$, the simulated commitment of **TrapCom** together with the equivocated opening to b is indistinguishable from an honest commitment and opening to b . Furthermore, since the simulation is straight-line and thus is composable under concurrent composition, we have that **TrapCom** is secure under selective opening attack with concurrent composition (See [Xia11] for a formal definition) against malicious receivers that always use a fixed challenge.

Next, by relying on the security against selective opening attack of **TrapCom**, we show that for every malicious receiver R^* of $\langle C, R \rangle$, there is a simulator S that can generate a simulated commitment that is indistinguishable from an honest commitment to R^* to any value v ; then the hiding property follows. More precisely, given a malicious receiver R^* of $\langle C, R \rangle$ (with loss of generality, deterministic), let c_1 be the Stage 1 commitment from R^* and e the challenge committed to in c_1 . The simulator S , on input e , simulates a commitment of $\langle C, R \rangle$ to v as follows: In Stage 2 and 3, it simulates the $10nL(n)$ commitments of **PTrapCom** w.r.t. challenge e by using the simulator of **PTrapCom** described above does; finally in Stage 4, upon receiving Γ , for every column $j \in \Gamma$, it samples a random string $\tilde{s}_j$ and equivocate all the simulated commitments of **PTrapCom** in the j^{th} column to $\tilde{s}_j$. Since R^* uses a fixed challenge e in all the **PTrapCom** commitments, it follows from the security against selective opening attack of **TrapCom** that in H the simulated commitments of **PTrapCom** in Stage 2 and 3, together with the equivocated openings to n random values in Stage 4 is indistinguishable from, the honest commitments and openings to n shares of v in the real execution (since by the property of the $(n+1)$ -out-of- $10n$ secret-sharing, n shares of v is identically distributed to n random values). Thus, the simulated commitment by S is indistinguishable to an honest commitment to v . Since S does not depend on the committed value v , we conclude that honest commitments to any two values v_1 and v_2 are indistinguishable. $\square$

B.3 Proof of Robust CCA-Security of $\langle C, R \rangle$

In this section, we prove the following theorem.

Theorem 2. $\langle C, R \rangle$ is $\kappa(n)$ -robust CCA-secure w.r.t. committed value oracles.

The formal proof of Theorem 2 consists of two parts: in Section B.3.2, we show that $\langle C, R \rangle$ is CCA-secure. and in section B.3.3, we show that it is also robust. Towards this, below we first adapt the definition of **safe-points** in [CLP10] to work with our protocol $\langle C, R \rangle$.

B.3.1 Safe-Points

Our notion of **safe-points** is almost the same as that in [CLP10] (which in turn is based on the notion of safe-points of [LPV08] and safe rewinding block of [DDN00]), with the only exception that our definition considers the 3-round rows in our black-box construction of CCA commitment $\langle C, R \rangle$, instead of the 3-round *WISSP* proofs in the non-black-box construction in [CLP10]. Recall that, like a *WISSP* proof, each row in Stage 3 of the protocol $\langle C, R \rangle$ has the 3-round challenge-reply structure—we call the three messages respectively, the commit, challenge and reply messages—and has the property that rewinding a complete row reveals nothing about the committed value.

Let Δ be a transcript of one left and many right interactions of $\langle C, R \rangle$. Intuitively, a **safe-point** ρ of a right interaction k is a prefix of a transcript Δ that lies in between the first two messages α_r and β_r of a row $(\alpha_r, \beta_r, \gamma_r)$ in interaction k , such that, when rewinding from ρ to γ_r , if A uses the *same* “scheduling of messages” as in Δ , then the left interaction can be emulated without affecting the hiding property. This holds, if in Δ from ρ to where γ_r is sent, A expects either no message or only complete rows in the left interaction, as shown in Figure 4 (i) and (ii) respectively, (Additionally, in both cases, A may request the reply message of some row in the left, as shown in Figure 4 (iii). This is because, given the first two messages of a row, the reply message is deterministic, and hence can be emulated in the rewinding by replaying the reply in Δ .)

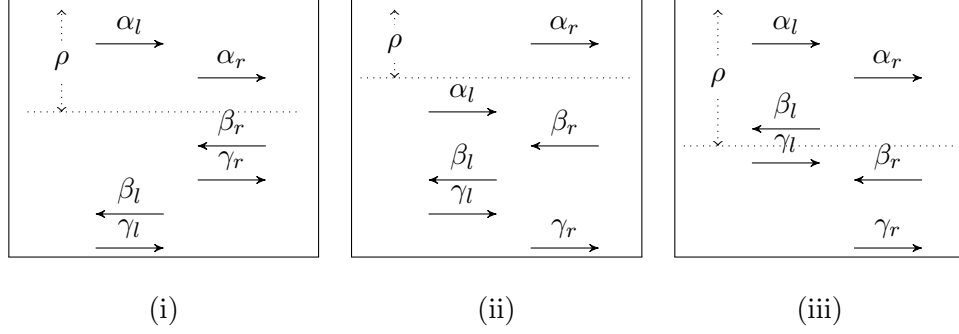


Figure 4: Three characteristic **safe-points**.

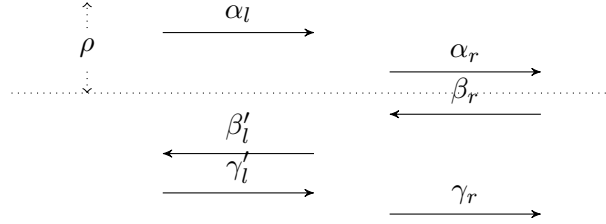


Figure 5: Prefix ρ that is not a safe point.

Definition 8. Let Δ be any transcript of one left interaction, and many right interactions, of $\langle C, R \rangle$. A prefix ρ of a transcript Δ is called a **safe-point** for right interaction k , if there exists an accepting row $(\alpha_r, \beta_r, \gamma_r)$ in the right interaction k , such that:

1. α_r occurs in ρ , but not β_r (and γ_r).
2. for any row $(\alpha_l, \beta_l, \gamma_l)$ in the left interaction, if α_l occurs in ρ , then β_l occurs after γ_r .
3. messages in Stage 1, 3, and 4 of the left interaction occur either before ρ or after γ_r .

If ρ is a **safe-point**, let $(\alpha_\rho, \beta_\rho, \gamma_\rho)$ denote the canonical “safe” right row associated with ρ . Note that the only case a right-interaction row is not associated with any **safe-point** is if it is “aligned” with a left-execution row, as shown in Figure 5. In contrast, in all other cases, a right-interaction row has a **safe-point**, as shown in Figure 4.

It follows from exactly the same proof in [CLP10] that in any transcript of one left and many right interactions of $\langle C, R \rangle$, every accepting right interaction that has a different identity from the left interaction, has at least $\eta(n)$ **safe-points**. This technical lemma will be very instrumental in the proof of CCA-security in the next section.

Lemma 6 (Safe-point Lemma). Let Δ be any transcript of one left interaction, and many right interactions, of $\langle C, R \rangle$. Then, in Δ , for every successful right interaction that has a different identity from the left interaction, there exist at least a number of $\Omega(\eta(n))$ non-overlapping rows that are associated with a **safe-point**.

B.3.2 Proof of CCA Security

We show that for every PPT adversary A , the following ensembles are computationally indistinguishable.

- $\{\text{IND}_0(\langle C, R \rangle, A, n, z)\}_{n \in N, z \in \{0,1\}^*}$

- $\{\text{IND}_1(\langle C, R \rangle, A, n, z)\}_{n \in N, z \in \{0,1\}^*}$

Towards this, we consider new commitment scheme $\langle \hat{C}, \hat{R} \rangle$ (similar to the “adaptor” schemes of [DDN00, LPV08, CLP10]), which is a variant of $\langle C, R \rangle$ where the receiver can ask for an arbitrary number of designs in Stage 2. Furthermore, $\langle \hat{C}, \hat{R} \rangle$ does not have a fixed scheduling in Stage 2; the receiver instead gets to choose which design to execute in each iteration (by sending bit b to select design_b). Note that, clearly, any execution of $\langle C, R \rangle$ can be emulated by an execution of $\langle \hat{C}, \hat{R} \rangle$ by simply requesting the appropriate designs. It follows using essentially the same proof for the hiding property of $\langle C, R \rangle$ in Proposition 2 that $\langle \hat{C}, \hat{R} \rangle$ is computationally hiding; we omit the proof here.

Now assume for contradiction that there exists an adversary A , a distinguisher D , and a polynomial p , such that for infinitely many $n \in N$, there exists $z \in \{0,1\}^*$, such that,

$$\left| \Pr[D(\text{IND}_0(\langle C, R \rangle, A, n, z)) = 1] - \Pr[D(\text{IND}_1(\langle C, R \rangle, A, n, z)) = 1] \right| \geq \frac{1}{p(n)}$$

We reach a contradiction by exhibiting a (stand-alone) adversary B^* that distinguishes commitments using $\langle \hat{C}, \hat{R} \rangle$. Let $\text{STA}_b(\langle \hat{C}, \hat{R} \rangle, B^*, n, z)$ denote the output of $B^*(1^n, z)$ after receiving a commitment of $\langle \hat{C}, \hat{R} \rangle$ to value v_b , where as in experiment IND_b the challenges v_0 and v_1 are chosen adaptively by B^* . We show that the following two claims hold w.r.t B^* .

Lemma 7. *There exists a polynomial function t and a negligible function μ , such that for every $b \in \{0,1\}$, $n \in N$ and $z \in \{0,1\}^*$, and every function p , the probability that B^* in experiment $\text{STA}_b(\langle \hat{C}, \hat{R} \rangle, B^*, n, z)$ takes more than $p(n)$ steps is less than or equal to $\frac{t(n)}{p(n)} + \mu(n)$.*

Lemma 8. *Let $b \in \{0,1\}$. The following ensembles are computationally indistinguishable.*

- $\left\{ \text{STA}_b(\langle \hat{C}, \hat{R} \rangle, B^*, n, z) \right\}_{n \in N, z \in \{0,1\}^*}$
- $\left\{ \text{IND}_b(\langle C, R \rangle, A, n, z) \right\}_{n \in N, z \in \{0,1\}^*}$

By Lemma 8, it thus follows that for infinitely many $n \in N$, there exists $z \in \{0,1\}^*$, such that,

$$\left| \Pr \left[D(\text{STA}_0(\langle \hat{C}, \hat{R} \rangle, B^*, n, z)) = 1 \right] - \Pr \left[D(\text{STA}_1(\langle \hat{C}, \hat{R} \rangle, B^*, n, z)) = 1 \right] \right| \geq \frac{3}{4p(n)}$$

Furthermore, by Lemma 7, the probability that B^* runs for more than $T(n) = 4t(n)p(n)$ steps is smaller than $1/4p(n)$. Therefore, the execution of B^* can be truncated after $T(n)$ steps, while only affecting the distinguishing probability by at most $\frac{1}{4p(n)}$, which means there exists a $\mathcal{PPT}$ machine that distinguishes commitments with probability $\frac{1}{2p(n)}$; this contradicts the hiding property of $\langle \hat{C}, \hat{R} \rangle$.

Construction of B^* . On a high-level, B^* in interaction with an honest committer $\hat{C}$ on the left emulates the committed-value oracle $\mathcal{O}$ for A by extracting the committed values of the right interactions from the rows in Stage 2 of $\langle C, R \rangle$. Recall that Stage 2 of $\langle C, R \rangle$ contains multiple rows; each in turn contains commitments to secret shares of the committed value (more precisely, shares of a decommitment of Stage 2 of $\langle C, R \rangle$), using Stage 2 of PTrapCom . It follows from the special soundness of PTrapCom that, given a pair of transcripts of a row that are admissible for each commitment of TrapCom contained in that row, the secret shares committed in that row can be reconstructed using the `reconst` procedure (provided that it does not output `err`)—we say that such a pair of transcripts of a row is admissible. Then the committed value can be recovered.

The construction of B^* contains two parts, a rewinding procedure and a reconstruction procedure. The rewinding procedure recursively rewinds the rows of the right integrations and guarantees that at the end of every accepting right interaction with different identity from the left interaction, a pair of admissible transcripts (of one row) of that right interaction is collected. The reconstruction procedure, on the other hand, on input a pair of admissible transcripts of one right interaction, reconstructs the committed value of that interaction. The rewinding procedure that we use here is essentially the same as that used in the CLP extraction procedure, except from the superficial difference that in [CLP10], pairs of accepting transcripts of *WLSSP* proofs are collected. However, in [CLP10] given two different accepting transcripts of a *WLSSP* proof in one right interaction, a committed value can be extracted directly using the special-soundness property *without over-extraction*. In this work, in order to avoid the over-extraction problem, the procedure for reconstructing the committed value from two admissible transcripts is much more involved; we employ the technique used in [CDSMW08, CDSMW09, Wee10] and formalize it in the reconstruction procedure REC.

Next, we formally describe the rewinding procedure, which invokes the reconstruction procedure REC whenever it obtains a pair of admissible transcripts; readers who are familiar with the CLP extraction procedure can skip this part and jump to the description reconstruction procedure REC.

The Rewinding Procedure At a high-level, the rewinding procedure rewinds A only from **safe-points**. This ensures that we do not have to rewind the external left execution; rather, it suffices to request an additional design on the left to handle these rewindings. But, as the simulator needs to provide the committed values in a “on-line” fashion (i.e., as soon as a right-interaction completes, the simulator needs to provide A with decommitment information for this interaction), these rewindings might become recursive (if the right interactions are nested). And, if we were to perform these rewindings naively, the running-time quickly grows exponentially (just as in the context of *concurrent zero knowledge* [DNS04]). To make sure that the recursion depth is constant, we instead only rewind from **safe-points** ρ such that the number of new right-rows that start between ρ and the last message γ_ρ of the right-row associated with ρ , is “small”; here, “small” is defined appropriately based on the recursion level. More precisely, we say that a **safe-point** ρ is $d + 1$ -good for a transcript Δ if less than $k_d = M/\eta'^d$ right-rows start between ρ and γ_ρ , where M is an upper-bound on the total number of messages that A sends or receives, and $\eta' = n^{\varepsilon'}$ for some constant ε' such that $0 < \varepsilon' < \varepsilon$. On recursion level d , B^* then only rewinds A from $d + 1$ -good **safe-points**.

Formally, we describe the rewinding procedure using a recursive helper procedure EXT. EXT, on input an integer d (the recursion level), a partial joint view $\mathcal{V}$ of A and the (emulated) right receivers, the index s of a right-row, a “repository” $\mathcal{R}$ of pairs of admissible transcripts of the right interactions that have been previously collected, proceeds as follows.

PROCEDURE EXT($d, \mathcal{V}, s, \mathcal{R}$): Let ρ be the (partial) transcript contained in $\mathcal{V}$. If $d = 0$, EXT will emulate a complete execution of IND with the adversary A . If $d > 0$, it will instead extends the partial view $\mathcal{V}$ to the completion of the right-row s ; if at any point in the emulation, ρ is not a $d + 1$ -good **safe-point** for s , EXT aborts and returns $\perp$. Finally, EXT returns the the view $\mathcal{V}_A$ of A in the emulation (generated so far). We now turn to describe how EXT emulates the left and the right interactions.

The left interaction is emulated by simply requesting the appropriate messages from the external committer. At the top level (i.e., $d = 0$), A participates in a *complete* $\langle C, R \rangle$ commitment on the left, which can be easily emulated by simply requesting the appropriate designs from $\hat{C}$. At lower levels (i.e., $d > 0$), EXT cancels every execution in which ρ is not a **safe-point**. Hence it only needs to emulate the left interaction when ρ is a **safe-point**. In this case, as previously discussed, A either does not request any new messages on the left, or only asks for complete new rows; the former case can be trivially emulated (by simply doing nothing or replaying old messages if A asks for the

third message of a left row again), in the latter case, EXT emulates the left interaction by asking for more designs from $\hat{C}$.

On the other hand, in the right interactions EXT follows the honest receiver strategy of $\langle C, R \rangle$. Furthermore, whenever A completes a row $(\alpha_r, \beta_r, \gamma_r)$ in a right interaction j , EXT attempts to extract a decommitment for this interaction, if the row $(\alpha_r, \beta_r, \gamma_r)$ is associated with a $d+1$ -good safe-point ρ' . To extract, EXT invokes itself recursively on input $(d+1, \mathcal{V}', s', \mathcal{R})$, where $\mathcal{V}'$ is the (partial) joint view of A and the right receivers corresponding to the transcript ρ' , and s' is the index of the right-row $(\alpha_r, \beta_r, \gamma_r)$. It continues invoking itself recursively until one of the recursive invocations returns a view containing another accepting transcript $(\alpha_r, \beta'_r, \gamma'_r)$ of the s'^{th} row. When this happens, if $(\alpha_r, \beta_r, \gamma_r)$ and $(\alpha_r, \beta'_r, \gamma'_r)$ are a pair of admissible transcripts, EXT records them in the repository $\mathcal{R}$. Later, whenever A expects the committed value for a right interaction j , it simply checks the repository $\mathcal{R}$ for a matching pair of admissible transcripts $\mathcal{T}_1, \mathcal{T}_2$, and invokes REC with $\mathcal{T}_1, \mathcal{T}_2$ and the transcript $\mathcal{T}$ of the right integration j to obtain the committed value u ; it aborts and outputs fail if no pair of admissible transcripts is available or the REC procedure returns err—we say that EXT “gets stuck” on interaction j in this case. (If A expects the committed value of a right interaction that fails or has the same identity as the left, it simply sends $\perp$ to A .)

The Reconstruction Procedure Let $\mathcal{T}_1 = (\alpha_r, \beta_r, \gamma_r)$ and $\mathcal{T}_2 = (\alpha_r, \beta'_r, \gamma'_r)$ be a pair of admissible transcripts of one row of a right commitment $\mathcal{T}$. Recall that Stage 2 in $\mathcal{T}$ contains a **com** commitment to the committed value v , and each row in $\mathcal{T}$ commits to the $10n$ secret shares of a decommitment (v, d) of the **com** commitment, using Stage 2 of PTrapCom. Since $\mathcal{T}_1$ and $\mathcal{T}_2$ are admissible, they contain a pair of admissible transcripts of PTrapCom for each commitment to one of the $10n$ share. Therefore, by the special-soundness property of PTrapCom, a value can be reconstructed from each pair of admissible transcripts of PTrapCom using the $\overline{\text{reconst}}$ procedure; if the extracted value is not err, then either the corresponding commitment is invalid, or it is and the reconstructed value is the committed share. At a high-level, the reconstruction procedure REC uses this property to try to extract shares of the decommitment (v, d) and then decode the shares to obtain the committed value; it utilizes the final cut-and-choose Stage in the right commitment $\mathcal{T}$ to avoid over-extraction. Formally,

PROCEDURE REC($\mathcal{T}_1, \mathcal{T}_2, \mathcal{T}$): For every $j \in [10n]$, REC sets T_1^j, T_2^j to the pair of admissible transcripts of PTrapCom for the commitment to the j^{th} secret share contained in $\mathcal{T}_1, \mathcal{T}_2$, and y_j to the output of $\overline{\text{reconst}}(T_1^j, T_2^j)$; it aborts and outputs err if y_j equals to err; otherwise, it sets the j^{th} share $\tilde{s}_j$ to y_j . After extracting all the shares $\tilde{\pi} = (\tilde{s}_1, \dots, \tilde{s}_{10n})$, it recovers a valid codeword w that is 0.8-close to $\tilde{\pi}$; then, it checks whether w agrees with all the shares revealed in the cut-and-choose stage in the right commitment $\mathcal{T}$ (that is, for every $i \in [\Gamma]$, it checks whether w_i equals to the share $\hat{s}_i$ revealed for the i^{th} column in the cut-and-choose stage in $\mathcal{T}$); if this holds, it decodes w to a tuple (v', d') , and outputs v' if (v', d') is a valid decommitment of the Stage 2 **com** commitment in $\mathcal{T}$. It aborts and outputs $\perp$ whenever any of the following events happens: (1) the extracted shares $\tilde{\pi}$ is not 0.8-close to any valid codeword, or (2) the codeword w does not agree with any of the shares revealed in the cut-and-choose stage, or (3) the tuple (v', d') decoded from w is not a valid decommitment of the Stage 2 **com** commitment.

We now return to the description of B^* . B^* in interaction with $\hat{C}$, simply invokes EXT on inputs $(0, \mathcal{V}, \text{null}, \emptyset)$, where $\mathcal{V}$ is the initial joint states of A and honest right receivers. Once EXT returns a view $\mathcal{V}_A$ of A , B^* return the output of A in this view if A never used the identity of the left interaction in any of the right interactions, and returns $\perp$ otherwise. Furthermore, to simplify our analysis, B^* cuts-off EXT whenever it runs for more than 2^n steps. If this happens, B^* halts and outputs fail.

Proof of Lemma 7 and 8: Next we proceed to show that B^* runs in expected polynomial time (Lemma 7) and the output of B^* is correctly distributed (Lemma 8).⁵ Towards this, we consider another machine $\tilde{B}$ that proceeds identically to B^* except that it has access to an oracle $\tilde{\mathcal{O}}$ that on input an *accepting transcript* $\mathcal{T}$ of a commitment of $\langle C, R \rangle$, returns the unique committed value of that commitment if it is valid, and returns $\perp$ if it is invalid or has more than one valid committed value; furthermore, $\tilde{B}$ runs a variant $\widetilde{\text{EXT}}$ of the recursive helper procedure EXT that B^* runs. $\widetilde{\text{EXT}}$ proceeds identically to EXT except that whenever A during the rewindings in $\widetilde{\text{EXT}}$ expects the committed value of a right commitment $\mathcal{T}$ (that is accepting and has an identity different from that of the left commitment), it queries the oracle $\tilde{\mathcal{O}}$ on $\mathcal{T}$ and feeds A the value v returned by $\tilde{\mathcal{O}}$. $\widetilde{\text{EXT}}$ still reconstructs a value v' for that right interaction as EXT does, but, it does abort and outputs $\widetilde{\text{fail}}$ as EXT does when reconstruction fails; instead it continues the execution and simply outputs $\widetilde{\text{fail}}$ on a *special output tape*; additionally, it outputs $\widetilde{\text{fail}}$ on a *special output tape* if the reconstructed value v' is different from the value v returned by $\mathcal{O}^*$. Finally, as B^* , $\tilde{B}$ cuts the execution of $\widetilde{\text{EXT}}$ after 2^n steps and outputs $\widetilde{\text{fail}}$. Below we show that the expected running time of $\tilde{B}$ is bounded by a polynomial and the output of $\tilde{B}$ in experiment STA_b is statistically close to the view of A in the experiment IND_b .

Claim 1. *There exists a polynomial function t , such that for every $b \in \{0, 1\}$, $n \in N$ and $z \in \{0, 1\}^*$, $\tilde{B}$ in experiment $\text{STA}_b(\langle \hat{C}, \hat{R} \rangle, \tilde{B}^{\tilde{\mathcal{O}}}, n, z)$ takes $t(n)$ steps in expectation.*

Claim 2. *Let $b \in \{0, 1\}$. The following ensembles are statistically close.*

- $\left\{ \text{STA}_b(\langle \hat{C}, \hat{R} \rangle, \tilde{B}^{\tilde{\mathcal{O}}}, n, z) \right\}_{n \in N, z \in \{0, 1\}^*}$
- $\left\{ \text{IND}_b(\langle C, R \rangle, A, n, z) \right\}_{n \in N, z \in \{0, 1\}^*}$

Furthermore, we show that except with negligible probability the, during the execution of $\tilde{B}$, the probability that $\tilde{B}$ outputs $\widetilde{\text{fail}}$ on the special output tape is negligible.

Claim 3. *For every $b \in \{0, 1\}$, $n \in N$ and $z \in \{0, 1\}^*$, the probability that $\tilde{B}$ during the execution of $\text{STA}_b(\langle \hat{C}, \hat{R} \rangle, \tilde{B}^{\tilde{\mathcal{O}}}, n, z)$ outputs $\widetilde{\text{fail}}$ on its special output tape is negligible.*

By construction, $\tilde{B}$ outputs $\widetilde{\text{fail}}$ only when during the rewindings in $\widetilde{\text{EXT}}$, it fails to reconstruct a committed value for a right interaction (that is accepting and has an identity different from that of the left commitment), or a value is reconstructed but is different from that returned by $\tilde{\mathcal{O}}$. By Claim 3, the above event happens with negligible probability. In other words, during the execution of $\tilde{B}$, except with negligible probability, the value $\tilde{B}$ reconstructs is always identical to that extracted by $\mathcal{O}^*$. Thus, except with negligible probability, it is equivalent to replace the values returned by the oracle $\tilde{\mathcal{O}}$ with the values $\tilde{B}$ reconstructs. Then since the only difference between $\tilde{B}$ and B^* lies in which values they use to feed the adversary A when it expects a committed value, we have that except with negligible probability, the running time and output distributions of $\tilde{B}$ are identical to that of B^* . Therefore, combining Claim 2, we directly have that for every $b \in \{0, 1\}$, the output of B^* in experiment $\text{STA}_b(\langle \hat{C}, \hat{R} \rangle, \tilde{B}^{\tilde{\mathcal{O}}}, n, z)$ is statistically close to $\text{IND}_b(\langle C, R \rangle, A, n, z)$, which concludes Lemma 8. Furthermore, By Claim 1, the expected running time of $\tilde{B}$ is bounded by a polynomial t ; therefore, for every function T , the probability that $\tilde{B}$ runs for more than $T(n)$ steps is no more than $p(n) = t(n)/T(n)$; then, the probability that B^* runs for more than $T(n)$

⁵Though the rewinding strategy of B^* is very similar to that in [CLP10], due to the difference in the reconstruction procedure, the analysis of the running time and output distribution of B^* is quite different from that in [CLP10].

steps must be bounded by $p(n)$ plus a negligible amount, which concludes Lemma 7. Now it remains to prove Claim 1, 2 and 3.

Proof of Claim 1—Running-time Analysis of $\widetilde{B}$.

To bound the expected running time of B^* , it suffices to bound the expected running time of the procedure $\widetilde{\text{EXT}}$. Below in Subclaim 1 we first show that the recursive depth of $\widetilde{\text{EXT}}$ is always a constant, and then bound the running time of $\widetilde{\text{EXT}}$ in Subclaim 2.

Subclaim 1. *There exists a constant D such that for every $n \in N$, and every $\mathcal{V}$, s , and $\mathcal{R}$, $\widetilde{\text{EXT}}(D, \mathcal{V}, s, \mathcal{R})$ does not perform any recursive calls.*

Proof. Recall that at recursion level d , the procedure $\widetilde{\text{EXT}}$ terminates and returns $\perp$ whenever more than $k_d = M(n)/\eta'(n)^d$ new right-rows has started in its execution, where $M(n)$ is an upper bound on the total number of messages that the adversary A may send and receive, and $\eta'(n)$ equals to $n^{\varepsilon'}$ for some constant $0 < \varepsilon' < \varepsilon < 1$. Let n^c be an upper bound on $M(n)$; set D to $\lceil \log_{\eta'(n)} n^c \rceil$, which is a constant. When $d = D$, $k_D < 1$, which means the execution terminates whenever A starts a new right-row. On the other hand, $\widetilde{\text{EXT}}$ only makes a recursive call at the completion of a new right-row. Therefore at recursion level D , $\widetilde{\text{EXT}}$ never makes any recursive calls. $\square$

Next, we show that the expected number of queries that $\widetilde{\text{EXT}}$ makes to A at every recursion level $d \leq D$ is always bounded by a polynomial.

Subclaim 2. *For every $0 \leq d \leq D$, it holds that for every $n \in N$, $\mathcal{V}$, s , and $\mathcal{R}$, the expected number of queries that $\widetilde{\text{EXT}}(d, \mathcal{V}, s, \mathcal{R})$ makes to A is bounded by $\theta(d) = M(n)^{3(D-d+1)}$.*

Proof. Consider some fixed $\mathcal{V}$, s and $\mathcal{R}$. We prove the subclaim by induction on d . When $d = D$, the claim follows, since $\widetilde{\text{EXT}}$ does not perform any recursive calls and the number of queries made by $\widetilde{\text{EXT}}$ can be at most the total number of messages, which is $M = M(n)$.

Assume the claim is true for $d = d' + 1$. We show that it holds also for $d = d'$. The procedure $\widetilde{\text{EXT}}$ simulates an execution with A in a straight-line on recursion level d' , until it encounters the completion of a right-row s that has a $d' + 1$ -good **safe-point** ρ , then it tries to obtain another transcript of s , by repeatedly invoking $\widetilde{\text{EXT}}$ on recursion level $d' + 1$ from (the partial transcript) ρ . Hence, the number of queries made by $\widetilde{\text{EXT}}$ is bounded by the sum of the number of queries made on level d' , and the queries made by the recursive calls: the former is at most the total number of messages, that is, M , while the latter is bounded by the sum of the queries made by those recursive calls invoked for every right-row s . Furthermore we compute the expected number of queries made by the recursive calls for a right-row s by taking expectation over all partial transcript that is potentially a d' -good **safe-point** for s . let Γ_i denote the set of all partial transcripts of length i that are consistent with $\mathcal{V}$; for every $\rho \in \Gamma_i$, we denote by $\Pr[\rho \text{ occurs on level } d']$ the probability that ρ occurs (in the simulation) on level d' , and $E[Q_{d'}^s(\rho)|\rho]$ the expected number of queries made by the recursive calls started from ρ for the right-row s , conditioned on ρ occurring on level d' . Then

$$E[\text{number of queries by } \widetilde{\text{EXT}}] = M + \sum_s \sum_i \sum_{\rho \in \Gamma_i} \Pr[\rho \text{ occurs on level } d'] E[Q_{d'}^s(\rho)|\rho]$$

Next we bound $E[Q_{d'}^s(\rho)|\rho]$ in two steps: the first step bounds the expected number of recursive calls started from ρ for proof s , and the second step uses the induction hypothesis to derive a bound on $E[Q_{d'}^s(\rho)|\rho]$.

Step 1: Given a partial transcript ρ from Γ_i , let $p_{d'}^s(\rho)$ denote the probability that conditioned on ρ occurring on level d' , $\widetilde{\text{EXT}}$ starts recursive calls from ρ for the right-row s , which happens if and only if ρ is a $d' + 1$ -good **safe-point** for proof s , that is,

$$p_{d'}^s(\rho) = \Pr[\rho \text{ is } d' + 1\text{-good at level } d' \mid \rho]$$

When this happens, $\widetilde{\text{EXT}}$ repeatedly calls itself on recursion level $d' + 1$, until an invocation succeeds without cancelling. Let $q_{d'}^s(\rho)$ denote the probability that conditioned on ρ occurring on level d' , a recursive call to $\widetilde{\text{EXT}}$ on level $d' + 1$ succeeds without cancelling. Since an invocation is cancelled if and only if ρ fails to be a $d' + 1$ -good **safe-point** for s in the invocation on level $d' + 1$, we have

$$q_{d'}^s(\rho) = \Pr [\rho \text{ is } d' + 1\text{-good at level } d' + 1 \mid \rho]$$

We claim that $q_{d'}^s(\rho) \geq p_{d'}^s(\rho)$. This is because, conditioned on ρ occurring, the view of A on levels d' and $d' + 1$ are simulated identically: on both levels d' and $d' + 1$, $\widetilde{\text{EXT}}$ emulates messages in the commitments of $\langle C, R \rangle$ for A perfectly; and furthermore, whenever A expects a committed value of a right interaction, $\widetilde{\text{EXT}}$ sends it the value returned by the oracle $\mathcal{O}^*$, which is deterministic; thus A always receives the same value on both level d' and $d' + 1$.

Then conditioned on ρ occurring on level d' , the expected number of recursive invocations to level $d' + 1$ before encountering a successful one is $1/q_{d'}^s(\rho)$. Since $\widetilde{\text{EXT}}$ only starts recursive invocations from ρ with probability $p_{d'}^s(\rho)$, we have that the expected number of recursive calls from ρ for proof s , conditioned on ρ occurring on level d' , is at most $p_{d'}^s(\rho)/q_{d'}^s(\rho) \leq 1$.

Step 2: From the induction hypothesis, we know that the expected number of queries made by an invocation of $\widetilde{\text{EXT}}$ on level $d' + 1$ is at most $\theta(d' + 1)$. Therefore, if u recursive invocations are made from ρ for a right row s , the expected number of queries made is bounded by $u\theta(d' + 1)$. Then we bound $E[Q_{d'}^s(\rho) \mid \rho]$ as follow:

$$\begin{aligned} E[Q_{d'}^s(\rho) \mid \rho] &\leq \sum_{u \in N} \Pr[u \text{ recursive calls are made from } \rho \text{ for } s] \cdot u \theta(d' + 1) \\ &= \theta(d' + 1) \sum_{u \in N} \Pr[u \text{ recursive calls are made from } \rho \text{ for } s] \cdot u \\ &\leq \theta(d' + 1) \end{aligned}$$

Therefore,

$$\begin{aligned} E[\text{number of queries by } \widetilde{\text{EXT}}] &\leq M + \sum_s \sum_i \sum_{\rho \in \Gamma_i} \Pr[\rho \text{ occurs on level } d'] \cdot \theta(d' + 1) \\ &= M + \theta(d' + 1) \sum_s \sum_i \sum_{\rho \in \Gamma_i} \Pr[\rho \text{ occurs on level } d'] \\ &= M + \theta(d' + 1) M^2 \\ &\leq M^{3(D-d'+1)} = \theta(d') \end{aligned}$$

□

Combining Subclaim 1 and 2, we conclude that the expected running time of machine $\tilde{B}$ is bounded by a polynomial $t'(n)$. This concludes Claim 1.

Proof of Claim 2—Correctness of the Output distribution of $\tilde{B}$. We show that for every $b \in \{0, 1\}$, the output of $\tilde{B}$ in $\text{STA}_b(\langle \hat{C}, \hat{R} \rangle, \tilde{B}^{\hat{\mathcal{O}}}, n, z)$ is statistically close to $\text{IND}_b(\langle C, R \rangle, A, n, z)$. Towards this, we show that conditioned on that $\tilde{B}$ does not output fail in $\text{STA}_b(\langle \hat{C}, \hat{R} \rangle, \tilde{B}^{\hat{\mathcal{O}}}, n, z)$, the output of $\tilde{B}$ is identically distributed to $\text{IND}_b(\langle C, R \rangle, A, n, z)$. By construction, $\tilde{B}$ invokes the recursive helper procedure $\widetilde{\text{EXT}}$ (at the top recursion level $d = 0$) and outputs fail if $\widetilde{\text{EXT}}$ runs for more than 2^n steps. Conditioned on $\tilde{B}$ not outputting fail, $\tilde{B}$ returns the output of A contained in the simulated view $\mathcal{V}_A$ returned by $\widetilde{\text{EXT}}$. As in $\text{IND}_b(\langle C, R \rangle, A, n, z)$, the output is replaced with $\perp$

if A copies the identity of the left interaction in any right interaction. Hence it suffices to show that in the case where A does not copy the identity of the left interaction, $\widetilde{\text{EXT}}$ simulates the messages in the left and right interactions for A perfectly. By construction of $\widetilde{\text{EXT}}$, all the messages belonging to the commitments of $\langle C, R \rangle$ (both on the left and right) are simulated perfectly; furthermore, $\widetilde{\text{EXT}}$ emulates the committed values of the right commitments using the values returned by the oracle $\tilde{\mathcal{O}}$, which are identical to the values extracted by the committed-value oracle $\mathcal{O}$ of $\langle C, R \rangle$. Therefore, the simulated view of A output by $\widetilde{\text{EXT}}$ is identically distributed to the real view of A in IND_b . Finally, by Claim 1, the probability that $\tilde{B}$ runs for more than 2^n steps is exponentially small. Therefore we conclude that $\text{STA}_b(\langle \hat{C}, \hat{R} \rangle, D^{\tilde{\mathcal{O}}}, n, z)$ is statistically close to $\text{IND}_b(\langle C, R \rangle, A, n, z)$.

Proof of Claim 3— $\tilde{B}$ almost never outputs fail. Assume for contradiction that there exist a polynomial p , $b \in \{0, 1\}$ and an infinitely number of $n \in N$ and $z \in \{0, 1\}^*$ such that $\tilde{B}$ in experiment $\text{STA}_b(\langle \hat{C}, \hat{R} \rangle, \tilde{B}^{\tilde{\mathcal{O}}}, n, z)$ outputs fail with probability $1/p(n)$. Fix a b , n , and z for which this holds. Then by Claim 1, the probability that $\tilde{B}$ runs for more than $T(n) = 2t(n)p(n)$ steps is no more than $1/2p(n)$, where $t(n)$ is the expected running time of $\tilde{B}$. Then consider another machine $\tilde{B}_T$ that proceeds identically to $\tilde{B}$ except that it cuts-off the execution after $T(n)$ steps. We have that $\tilde{B}_T$ takes a strict polynomial number $T(n)$ of steps and the probability that $\tilde{B}_T$ outputs fail is at least $1/2p(n)$.

Now consider another machine B_T^* that proceeds identically to B^* except that it also cuts-off the execution after $T(n)$ steps. We claim that in the experiment STA_b , B_T^* outputs fail or reconstructs a value for a right interaction that is not the valid committed value with polynomial probability. Assume for contradiction that this is false, that is, except with negligible probability, B_T^* always succeeds in reconstructing a valid committed value whenever the adversary expects a committed value during the rewindings. Then except with negligible probability, the committed values emulated by B_T^* are identical to that emulated by $\tilde{B}_T$. Therefore, the simulated view of A in B_T^* is statistically close to that in $\tilde{B}_T$. This implies that except with negligible probability, $\tilde{B}_T$ also always succeeds in reconstructing a valid committed value whenever the adversary expects one. Thus $\tilde{B}_T$ outputs fail only with negligible probability, which contradicts with our hypothesis. Therefore with polynomial probability, B_T^* fails to extract a valid committed value for some right interaction during its execution.

Below we reach a contradiction by showing that the probability that B_T^* outputs fail (Subclaim 3) and the probability that B_T^* reconstructs a value that is not the value committed value are negligible.

Subclaim 3. *For every $b \in \{0, 1\}$, $n \in N$ and $z \in \{0, 1\}^*$, the probability that B_T^* outputs fail during the execution of $\text{STA}_b(\langle \hat{C}, \hat{R} \rangle, B_T^*, n, z)$ is negligible.*

Subclaim 4. *For every $b \in \{0, 1\}$, $n \in N$ and $z \in \{0, 1\}^*$, the probability that there exists a right interaction that is accepting and has an identity different from that of the left interaction, for which B_T^* reconstruct a value that is not the valid committed value in $\text{STA}_b(\langle \hat{C}, \hat{R} \rangle, B_T^*, n, z)$ is negligible.*

Proof of Subclaim 3. Consider a fixed $b \in \{0, 1\}$. By construction, B_T^* outputs fail if and only if one of the following cases occurs (in which B^* outputs fail).

Case 1: None of the rows in the right interaction is rewound.

Case 2: Some row is rewound and a pair of accepting transcripts of that row is collected, but that pair of transcripts is not admissible.

Case 3: A pair of admissible transcripts is collected, but the REC construction invoked with them outputs err.

Below we analyze the probabilities that each of the above cases occurs. We show that all these events occur with only negligible probability. Therefore, overall the probability that B_T^* outputs $\widetilde{\text{fail}}$ is negligible.

Analysis of Case 1: We show that Case 1 never happens. More precisely, for every accepting right interaction j with a different identity from the left interaction, one of its rows must be rewound. By Lemma 6, there exist a number of $\Omega(\eta(n))$ non-overlapping rows in the right interaction j that has a **safe-point**. Recall that in B_T^* (more precisely, in $\widetilde{\text{EXT}}$), a right interaction may be carried out at multiple different recursion levels (through recursive calls); and at level d , B_T^* rewinds every row in this interaction that has a $d + 1$ -good **safe-point**. By Subclaim 1, the recursion depth is only a constant; hence there must be a level d , on which a number of $\Omega(\eta(n))$ non-overlapping rows with a **safe-point** start in interaction j . Since the total number of right-rows that start on level d is bounded by $k_d = M/\eta'(n)^d$ (otherwise, the simulation is cancelled) and $\eta'(n) = o(\eta(n))$, there must exist one right-row that has a **safe-point** ρ , such that there are less than $M/\eta'(n)^{d+1}$ right-rows starting in between ρ and the last message of the row. Therefore ρ is a $d + 1$ -good **safe-point** for this right-row, and will be rewound.

Analysis of Case 2: Recall that a transcript of one row consists of a polynomial number of parallel commitments using Stage 2 of PTrapCom, each of which consists of a polynomial number of parallel commitments using ExtCom. For a pair of transcripts of one row to be admissible, it must hold that all the pairs of transcripts it contains for each commitment of ExtCom are admissible w.r.t. ExtCom, that is, the two n -bit challenges in that pair of transcripts are different. Therefore, a pair of transcripts of a row is admissible if and only if the two challenge messages it contains, which are two tuples of a polynomial number q of n -bit strings $\alpha = (\alpha_1, \dots, \alpha_q)$, $\beta = (\beta_1, \dots, \beta_q)$, are different at every location, that is, $\alpha_i \neq \beta_i$ for every $i \in [q]$.

We bound the probability that any n -bit challenge message is picked twice in whole execution of B_T^* to be negligible. Then since conditioned on this not happening, every two accepting transcripts of a row are admissible, we conclude that this case happens with negligible probability. Since B_T^* runs for at most $T(n)$ steps, it picks at most $T(n)$ n -bit challenges during the whole execution. By applying the union bound, we obtain that, the probability that a challenge β is picked again is at most $\frac{T(n)}{2^n}$, and hence, using the union bound again, the probability that *any* challenge in the execution is picked twice is at most $T(n)\frac{T(n)}{2^n}$. Hence, overall, the probability that this case occurs is negligible.

Analysis of Case 3: Let $\mathcal{T}_1, \mathcal{T}_2$ be a pair of admissible transcripts of one row of an accepting commitment $\mathcal{T}$ of $\langle C, R \rangle$. The REC procedure, on input $\mathcal{T}_1, \mathcal{T}_2, \mathcal{T}$, outputs **err** if and only if one of the invocations to the **reconst** procedure returns **err**. Then by Lemma 4, the receiver's challenge in $\mathcal{T}$, which is shared among all the TrapCom commitments in $\mathcal{T}$, can be computed efficiently and deterministically from $\mathcal{T}_1$ and $\mathcal{T}_2$.

Then, suppose for contradiction that, there exists a polynomial g , such that for infinitely many $n \in N$ and z , case 3 occurs with probability at least $1/g(n)$ during the execution of B_T^* . Then the probability that case 3 occurs in a randomly chosen right interaction in B_T^* is at least $1/g(n)T(n)$. In other words, with polynomial probability, for a randomly chosen right interaction in B_T^* , REC is invoked and outputs **err**; then by the argument above, the receiver's challenge in this randomly chosen right interaction can be computed efficiently from the pair of admissible transcripts collected for this interaction (as input to REC). Furthermore, we note that in B_T^* , a pair of admissible transcripts is collected (if at all) for a right interaction,

before Stage 3 of that right interaction starts, and thus before the `com` commitment to the receiver's challenge is opened. Therefore we can use B^* to violate the hiding property of `com`.

More precisely, we construct a machine A^* that violates the hiding property of `com`. A^* on input a `com` commitment c to a random n -bit string e , internally emulates an interaction between $\hat{C}$ and B_T^* , except that it picks a random right interaction (in simulation by B_T^*) and feeds c as the Stage 1 message of that interaction; furthermore, after a pair of admissible transcripts $\mathcal{T}_1, \mathcal{T}_2$ is collected for this right interaction, A^* computes a challenge e' as described above; then, it aborts and outputs e' . Since A^* emulates an interaction between $\hat{C}$ and B_T^* perfectly before it aborts, the probability case 3 happens in a randomly chosen right interaction in A^* is identical to that in a randomly chosen right interaction in B_T^* , which is $1/g(n)T(n)$. Thus with polynomial probability, the challenge computed by A^* is the real challenge committed to in c . Thus A^* violates the hiding property of `com`. □

Proof of Subclaim 4. Consider a fixed $b \in \{0, 1\}$, n, z and a fixed right interaction $j \in [T(n)]$, we show that the probability that B_T^* reconstructs a value for the j^{th} right interaction that is not the valid committed value is negligible. Then it follows from a union bound that the probability that B_T^* reconstructs a value that is not the valid committed value in any right interaction is also negligible. If a value is reconstructed successfully for the j^{th} right interaction $\mathcal{T}$, it must be the case that interaction j is accepting, and a pair of admissible transcripts $\mathcal{T}_1, \mathcal{T}_2$ is collected and $\text{REC}(\mathcal{T}_1, \mathcal{T}_2, \mathcal{T}) = x \neq \text{err}$ in $\mathcal{B}_T^*$. Then we show that except with negligible probability, x must be the committed value in $\mathcal{T}$.

Each row of a commitment of $\langle C, R \rangle$ contains $10n$ commitments of `TrapCom`. It follows from the strong computational binding property (see Lemma 5) of `TrapCom` that the probability that any of the `TrapCom` commitments generated by machine B_T^* has two valid committed value is negligible.⁶ Therefore, the shares committed to using `TrapCom` in the j^{th} right interaction $\mathcal{T}$ are uniquely defined; let $s_1^i, \dots, s_{10n}^i$ be the committed shares in the i^{th} row of $\mathcal{T}$ (s_k^i is set to $\perp$ if the k^{th} commitment in the i^{th} row is invalid). We say a column k is inconsistent if it contains a s_i^k equals to $\perp$ or two $s_{i_1}^k, s_{i_2}^k$ that are different; we claim that the probability that there are more than n inconsistent columns in the j^{th} right interaction is negligible. Recall that in the cut-and-choose stage of the right interaction j , B_T^* emulates the honest receiver's message by sending a randomly chosen subset Γ of size n ; since the j^{th} right interaction is accepting, the adversary A must successfully reveal all the commitments in each column in Γ to the same value; therefore all the columns in Γ are consistent. Since Γ is chosen at random, the probability that the j^{th} right interaction contains more than n inconsistent columns, but none of them is chosen in Γ is exponentially small. Therefore, except with negligible probability, at least 0.9 fraction of the columns are consistent.

Now we are ready to show that the value x returned by the procedure `REC` on input a pair of admissible transcripts $\mathcal{T}_1, \mathcal{T}_2$ of the k^{th} row of the right commitment $\mathcal{T}$ is the valid committed value. From $\mathcal{T}_1, \mathcal{T}_2$, the procedure `RECEXTRACTS` $10n$ values $\tilde{s}_1^k, \dots, \tilde{s}_{10n}^k$ corresponding to the $10n$ shares committed to in the k^{th} row of $\mathcal{T}$. Then consider the following two possible cases:

In the first case, the shares $s_1^1, \dots, s_{10n}^1$ committed to in the first row of $\mathcal{T}$ is 0.9-close to a valid code w . Since except with negligible probability, at least 0.9 fraction of the columns are consistent, by the special soundness of `TrapCom`, the extracted shares $\tilde{s}_1^k, \dots, \tilde{s}_{10n}^k$ and with $s_1^1, \dots, s_{10n}^1$ agree with each other at no less than 0.9 fraction of positions. Therefore

⁶This also relies on the fact that `TrapCom` is a generalized public-coin protocol, in the sense, that given a partial transcript of a commitment of `TrapCom`, messages from the honest receiver continuing from that partial transcript can be emulated efficiently, by simply sending random strings.

$\tilde{s}_1^k, \dots, \tilde{s}_{10n}^k$ is 0.8-close to w . Therefore the procedure REC will recover w uniquely. Then if w agrees with all the shares opened in the cut-and-choose stage in $\mathcal{T}$, the valid committed value is the value v encoded in w ; in this case, REC also performs the same check and will output v as the committed value correctly. On the other hand, if w disagrees with one of the shares opened in the cut-and-choose stage in $\mathcal{T}$, the commitment is invalid and the committed value is set to $\perp$; in this case REC performing the same check, will also return $\perp$ correctly.

In the second case, the shares $s_1^1, \dots, s_{10n}^1$ committed to in the first row of $\mathcal{T}$ is 0.1-far away from every valid code w . In this case, the commitment $\mathcal{T}$ is invalid and the committed value is set to $\perp$. We show that in this case, the probability that the procedure REC does not output $\perp$ is negligible. If REC outputs a value $v' \neq \perp$, it must be the case that $\tilde{s}_1^k, \dots, \tilde{s}_{10n}^k$ is 0.8-close to a valid codeword w' that encodes v' . By our hypothesis, $s_1^1, \dots, s_{10n}^1$ is 0.1-far away from w' . Since $\mathcal{T}$ is accepting, all the columns in Γ are consistent, and thus the shares revealed in the cut-and-choose stage equals to $\{s_i^1\}_{i \in \Gamma} \neq \perp$. By construction of REC, it outputs v' only if w' agrees with all the shares revealed in the cut-and-choose stage, that is, $s_i^1 = w'_i$ for every $i \in \Gamma$. However, since the set Γ is chosen at random by the honest receiver (emulated by B_T^*), the probability that w' disagrees with $s_1^1, \dots, s_{10n}^1$ at more than n locations but none of them is selected in Γ is exponentially small. Therefore, except with negligible probability, REC outputs $\perp$ correctly.

Combining the above two cases, we conclude that, except with negligible probability, the values reconstructed by REC must be the valid committed value. $\square$

B.3.3 Proof of Robustness

In this section, we extend the proof in the last section to show that $\langle C, R \rangle$ is also *robust* w.r.t. the committed-value oracle $\mathcal{O}$. Towards this, we need to show that for every $k \leq \kappa(n)$, and every $\mathcal{PPT}$ adversary A , there exists a simulator S , such that, for every $\mathcal{PPT}$ k -round ITM B , the interaction between B and A with access to $\mathcal{O}$ is indistinguishable from that between B and S . The construction of the simulator is similar to that in [CLP10], but the correctness follows from a proof similar to the proof of CCA in the last section. For completeness, we provide the construction of S below; but omit the proof.

Given an adversary A , and a constant k , the construction of the simulator S is very similar to that of B^* in the last section. On a high-level, S externally interacts with an arbitrary k -round ITM B , and internally simulates an execution between B and $A^{\mathcal{O}}$, by forwarding messages from B internally to A , while concurrently extracting the committed values of the right interactions from A to simulate $\mathcal{O}$. The extraction strategy of S is essentially the same as that used by B^* : it recursively rewinds A over the rows in Stage 2 of the protocol to extract the committed values, except that, here the goal is to make sure that the left interaction with B is *never* rewind, (instead of the goal of ensuring that the left interaction remains hiding (in B^*)). This is achieved by rewinding only those right-rows that do not interleave with any messages in the left interaction, and cancelling every rewinding in which the right-row interleaves with a left-message. More precisely, consider the notion of **R-safe-point** (which is in analogous to the notion of **safe-point**)—a prefix ρ of a transcript Δ is a **R-safe-point** for a right-row (α, β, γ) if it includes all the messages in Δ up to α (inclusive), and that no left-message is exchanged in between ρ and γ . Then S simply runs the procedure EXT defined in the last section internally, except that it replaces the notion of **safe-point** with **R-safe-point**, and that it simulates the left interaction with A by forwarding the messages between A and B ; everything else remains the same. Then it follows from the fact that S always rewinds A from a **R-safe-point** ρ , and cancels every rewinding in which ρ is not a **R-safe-point**, the left interaction is never rewind. Furthermore, since the left interaction with B consists of only $k \leq \kappa(n)$ rounds, and the protocol $\langle C, R \rangle$ contains at least $\kappa(n) + \eta(n)$ rows, there exist at least $\eta = n^\epsilon$ **R-safe-point** in every successful

right interaction. Then, it follows from the same proof as in Claim 7 and Claim 8 that there is a polynomial t , such that the probability that S takes more than $T(n)$ steps is smaller than $t(n)/T(n)$ plus a negligible amount, and that the joint output of S and B is indistinguishable from that of $A^\mathcal{O}$ and B .

C Model of Security

C.1 UC and Global UC security

We briefly review UC and externalized UC (EUC) security. For full details see [Can00, CDPW07]. The original motivation to define EUC security was to capture settings where all ITMs in the system have access to some global, potentially trusted information (such as a globally available public key infrastructure or a bulletin board) [CDPW07]. Here however we use the EUC formalism to capture the notion of global helper functionalities that are available only to the corrupted parties.

We first review the model of computation, ideal protocols, and the general definition of securely realizing an ideal functionality. Next we present hybrid protocols and the composition theorem.

The basic model of execution. Following [GMR89, Gol01], a protocol is represented as an interactive Turing machine (ITM), which represents the program to be run within each participant. Specifically, an ITM has three tapes that can be written to by other ITMs: the **input** and **subroutine output** tapes model the inputs from and the outputs to other programs running within the same “entity” (say, the same physical computer), and the **incoming communication** tapes and **outgoing communication** tapes model messages received from and to be sent to the network. It also has an **identity tape** that cannot be written to by the ITM itself. The identity tape contains the program of the ITM (in some standard encoding) plus additional identifying information specified below. Adversarial entities are also modeled as ITMs.

We distinguish between ITMs (which represent static objects, or programs) and *instances of ITMs*, or *ITIs*, that represent interacting processes in a running system. Specifically, an ITI is an ITM along with an identifier that distinguishes it from other ITIs in the same system. The identifier consists of two parts: A **session-identifier** (SID) which identifies which protocol instance the ITI belongs to, and a **party identifier** (PID) that distinguishes among the parties in a protocol instance. Typically the PID is also used to associate ITIs with “parties”, or clusters, that represent some administrative domains or physical computers.

The model of computation consists of a number of ITIs that can write on each other’s tapes in certain ways (specified in the model). The pair (SID,PID) is a unique identifier of the ITI in the system. With one exception (discussed within) we assume that all ITMs are probabilistic polynomial time.⁷

Security of protocols. Protocols that securely carry out a given task (or, protocol problem) are defined in three steps, as follows. First, the process of executing a protocol in an adversarial environment is formalized. Next, an “ideal process” for carrying out the task at hand is formalized. In the ideal process the parties do not communicate with each other. Instead they have access to an “ideal functionality,” which is essentially an incorruptible “trusted party” that is programmed to capture the desired functionality of the task at hand. A protocol is said to securely realize an ideal functionality if the process of running the protocol amounts to “emulating” the ideal process

⁷An ITM is *PPT* if there exists a constant $c > 0$ such that, at any point during its run, the overall number of steps taken by M is at most n^c , where n is the overall number of bits written on the *input tape* of M in this run. In fact, in order to guarantee that the overall protocol execution process is bounded by a polynomial, we define n as the total number of bits written to the input tape of M , *minus the overall number of bits written by M to input tapes of other ITMs*; see [Can01].

for that ideal functionality. Below we overview the model of protocol execution (called the *real-life model*), the ideal process, and the notion of protocol emulation.

The model for protocol execution. The model of computation consists of the parties running an instance of a protocol π , an **adversary** A that controls the communication among the parties, and an **environment** Z that controls the inputs to the parties and sees their outputs. We assume that all parties have a security parameter $k \in \mathbf{N}$. (We remark that this is done merely for convenience and is not essential for the model to make sense). The execution consists of a sequence of *activations*, where in each activation a single participant (either Z , A , or some other ITM) is activated, and may write on a tape of at most *one* other participant, subject to the rules below. Once the activation of a participant is complete (i.e., once it enters a special waiting state), the participant whose tape was written on is activated next. (If no such party exists then the environment is activated next.)

The environment is given an external input z and is the first to be activated. In its first activation, the environment invokes the adversary A , providing it with some arbitrary input. In the context of UC security, the environment can from now on invoke (namely, provide input to) only ITMs that consist of a single instance of protocol π . That is, all the ITMs invoked by the environment must have the same SID and the code of π . In the context of EUC security the environment can in addition invoke an additional ITI that interacts with all parties. We call this ITI the **helper functionality**, denoted $\mathcal{H}$.

Once the adversary is activated, it may read its own tapes and the outgoing communication tapes of all parties. It may either **deliver** a message to some party by writing this message on the party's incoming communication tape or **report** information to Z by writing this information on the subroutine output tape of Z . For simplicity of exposition, in the rest of this paper we assume authenticated communication; that is, the adversary may deliver only messages that were actually sent. (This is however not essential since authentication can be realized via a protocol, given standard authentication infrastructure [Can04].)

Once a protocol party (i.e., an ITI running π) is activated, either due to an input given by the environment or due to a message delivered by the adversary, it follows its code and possibly writes a local output on the subroutine output tape of the environment, or an outgoing message on the adversary's incoming communication tape.

The protocol execution ends when the environment halts. The output of the protocol execution is the output of the environment. Without loss of generality we assume that this output consists of only a single bit.

Let $\text{EXEC}_{\pi,A,Z}(k, z, r)$ denote the output of the environment Z when interacting with parties running protocol π on security parameter k , input z and random input $r = r_Z, r_A, r_1, r_2, \dots$ as described above (z and r_Z for Z ; r_A for A , r_i for party P_i). Let $\text{EXEC}_{\pi,A,Z}(k, z)$ denote the random variable describing $\text{EXEC}_{\pi,A,Z}(k, z, r)$ when r is uniformly chosen. Let $\text{EXEC}_{\pi,A,Z}$ denote the ensemble $\{\text{EXEC}_{\pi,A,Z}(k, z)\}_{k \in \mathbf{N}, z \in \{0,1\}^*}$.

Ideal functionalities and ideal protocols. Security of protocols is defined via comparing the protocol execution to an *ideal protocol* for carrying out the task at hand. A key ingredient in the ideal protocol is the *ideal functionality* that captures the desired functionality, or the specification, of that task. The ideal functionality is modeled as another ITM (representing a “trusted party”) that interacts with the parties and the adversary. More specifically, in the ideal protocol for functionality $\mathcal{F}$ all parties simply hand their inputs to an ITI running $\mathcal{F}$. (We will simply call this ITI $\mathcal{F}$. The SID of $\mathcal{F}$ is the same as the SID of the ITIs running the ideal protocol. (the PID of $\mathcal{F}$ is null.)) In addition, $\mathcal{F}$ can interact with the adversary according to its code. Whenever $\mathcal{F}$ outputs a value to a party, the party immediately copies this value to its own output tape. We call the parties in the ideal protocol **dummy parties**. Let $\pi(\mathcal{F})$ denote the ideal protocol for functionality $\mathcal{F}$.

Securely realizing an ideal functionality. We say that a protocol π *emulates* protocol ϕ if for any adversary A there exists an adversary S such that no environment Z , on any input, can tell with non-negligible probability whether it is interacting with A and parties running π , or it is interacting with S and parties running ϕ . This means that, from the point of view of the environment, running protocol π is ‘just as good’ as interacting with ϕ . We say that π *securely realizes* an ideal functionality $\mathcal{F}$ if it emulates the ideal protocol $\pi(\mathcal{F})$. More precise definitions follow. A distribution ensemble is called *binary* if it consists of distributions over $\{0, 1\}$.

Definition 9. Let π and ϕ be protocols. We say that π UC-emulates (resp., EUC-emulates) ϕ if for any adversary A there exists an adversary S such that for any environment Z that obeys the rules of interaction for UC (resp., EUC) security we have $\text{EXEC}_{\phi, S, Z} \approx \text{EXEC}_{\pi, A, Z}$.

Definition 10. Let $\mathcal{F}$ be an ideal functionality and let π be a protocol. We say that π UC-realizes (resp., EUC-realizes) $\mathcal{F}$ if π UC-emulates (resp., EUC-emulates) the ideal protocol $\pi(\mathcal{F})$.

Security with dummy adversaries. Consider the adversary $\mathcal{D}$ that simply follows the instructions of the environment. That is, any message coming from one of the ITIs running the protocol is forwarded to the environment, and any input coming from the environment is interpreted as a message to be delivered to the ITI specified in the input. We call this adversary the **dummy adversary**. A convenient lemma is that UC security with respect to the dummy adversary is equivalent to standard UC security. That is:

Definition 11. Let π and ϕ be protocols. We say that π UC-emulates (resp., EUC-emulates) ϕ w.r.t the dummy adversary $\mathcal{D}$ if there exists an adversary S such that for any environment Z that obeys the rules of interaction for UC (resp., EUC) security we have $\text{EXEC}_{\phi, S, Z} \approx \text{EXEC}_{\pi, \mathcal{D}, Z}$.

Theorem 3. Let π and ϕ be protocols. Then π UC-emulates (resp., EUC-emulates) ϕ if and only if π UC-emulates (resp., EUC-emulates) ϕ with respect to the dummy adversary.

Hybrid protocols. Hybrid protocols are protocols where, in addition to communicating as usual as in the standard model of execution, the parties also have access to (multiple copies of) an ideal functionality. Hybrid protocols represent protocols that use idealizations of underlying primitives, or alternatively make *trust assumptions* on the underlying network. They are also instrumental in stating the universal composition theorem. Specifically, in an $\mathcal{F}$ -hybrid protocol (i.e., in a hybrid protocol with access to an ideal functionality $\mathcal{F}$), the parties may give inputs to and receive outputs from an unbounded number of copies of $\mathcal{F}$.

The communication between the parties and each one of the copies of $\mathcal{F}$ mimics the ideal process. That is, giving input to a copy of $\mathcal{F}$ is done by writing the input value on the input tape of that copy. Similarly, each copy of $\mathcal{F}$ writes the output values to the subroutine output tape of the corresponding party. It is stressed that the adversary does not see the interaction between the copies of $\mathcal{F}$ and the honest parties.

The copies of $\mathcal{F}$ are differentiated using their SIDs. All inputs to each copy and all outputs from each copy carry the corresponding SID. The model does not specify how the SIDs are generated, nor does it specify how parties “agree” on the SID of a certain protocol copy that is to be run by them. These tasks are left to the protocol. This convention seems to simplify formulating ideal functionalities, and designing protocols that securely realize them, by freeing the functionality from the need to choose the SIDs and guarantee their uniqueness. In addition, it seems to reflect common practice of protocol design in existing networks.

The definition of a protocol securely realizing an ideal functionality is extended to hybrid protocols in the natural way.

The universal composition operation. We define the universal composition operation and state the universal composition theorem. Let ρ be an $\mathcal{F}$ -hybrid protocol, and let π be a protocol that securely realizes $\mathcal{F}$. The composed protocol ρ^π is constructed by modifying the code of each ITM in ρ so that the first message sent to each copy of $\mathcal{F}$ is replaced with an invocation of a new copy of π with fresh random input, with the same SID, and with the contents of that message as input. Each subsequent message to that copy of $\mathcal{F}$ is replaced with an activation of the corresponding copy of π , with the contents of that message given to π as new input. Each output value generated by a copy of π is treated as a message received from the corresponding copy of $\mathcal{F}$. The copy of π will start sending and receiving messages as specified in its code. Notice that if π is a $\mathcal{G}$ -hybrid protocol (i.e., ρ uses ideal evaluation calls to some functionality $\mathcal{G}$) then so is ρ^π .

The universal composition theorem. Let $\mathcal{F}$ be an ideal functionality. In its general form, the composition theorem basically says that if π is a protocol that UC-realizes $\mathcal{F}$ (resp., EUC-realizes $\mathcal{F}$) then, for any $\mathcal{F}$ -hybrid protocol ρ , we have that an execution of the composed protocol ρ^π “emulates” an execution of protocol ρ . That is, for any adversary A there exists a simulator $\mathcal{S}$ such that no environment machine Z can tell with non-negligible probability whether it is interacting with A and protocol ρ^π or with $\mathcal{S}$ and protocol ρ , in a UC (resp., EUC) interaction. As a corollary, we get that if protocol ρ UC-realizes $\mathcal{F}$ (resp., EUC-realizes $\mathcal{F}$), then so does protocol ρ^π .⁸

Theorem 4 (Universal Composition [Can01, CDPW07]). *Let $\mathcal{F}$ be an ideal functionality. Let ρ be a $\mathcal{F}$ -hybrid protocol, and let π be a protocol that UC-realizes $\mathcal{F}$ (resp., EUC-realizes $\mathcal{F}$). Then protocol ρ^π UC-emulates ρ (resp., EUC-emulates ρ).*

An immediate corollary of this theorem is that if the protocol ρ UC-realizes (resp., EUC-realizes) some functionality $\mathcal{G}$, then so does ρ^π .

C.2 UC Security with Super-polynomial Helpers

We modify the definitions of UC security by giving the corrupted parties access to an external “helper” entity, in a conceptually similar way to [PS04]. This entity, denoted $\mathcal{H}$, is computationally unbounded, and can be thought of as providing the corrupted parties with some judicious help. (As we’ll see, this help will be used to assist the simulator to “reverse engineering” the adversary in order to extract relevant information hidden in its communication.)

The definition uses the formalism of EUC security [CDPW07]. Specifically, we model the helper entity as an ITM that is invoked directly by the environment, and that interacts with the environment and the corrupted parties. More formally, let $\mathcal{H}$ be an ITM. An environment Z is called *aided by $\mathcal{H}$* if: (a) Z invokes a single instance $\mathcal{H}$ immediately after invoking the adversary; (b) As soon as a party (i.e., an ITI) P is corrupted (i.e., P receives a **corrupted** message), Z lets $\mathcal{H}$ know of this fact; (c) $\mathcal{H}$ interacts only with the corrupted parties. Then:

Definition 12. *Let π and ϕ be protocols, and let $\mathcal{H}$ be a helper functionality (i.e., an ITM). We say that π $\mathcal{H}$ -EUC-emulates ϕ if for any adversary A there exists an adversary $\mathcal{S}$ such that for any environment Z that’s aided by $\mathcal{H}$ we have $\text{EXEC}_{\phi, \mathcal{S}, Z} \approx \text{EXEC}_{\pi, A, Z}$.*

The meaningfulness of relativized UC security of course depends on the particular helper ITM in use. Still, it is easy to see that if protocol π $\mathcal{H}$ -EUC-emulates protocol ϕ where $\mathcal{H}$ obeys the above rules and runs in time $T(n)$, then π UC-emulates ϕ according to a relaxed notion where the adversary $\mathcal{S}$ can run in time $\text{poly}(T(n))$. As noted in the past, for many protocols and ideal

⁸The universal composition theorem in [Can01] applies only to “subroutine respecting protocols”, namely protocols that do not share subroutines with any other protocol in the system. In [CDPW07] the theorem is extended to protocols that share subroutines with arbitrary other protocols, as long as the composed protocol, ρ^π , realizes $\mathcal{F}$ with EUC security.

functionalities, this relaxed notion of security suffices even when $T(n) = \exp(n)$ [Pas03b, PS04, BS05, MMY06].

Universal Composition with super-polynomial helpers. The universal composition theorem generalizes naturally to the case of EUC, even with super-polynomial helper functionalities:

Theorem (universal composition for relativized UC). *Let $\mathcal{F}$ be an ideal functionality, let $\mathcal{H}$ be a helper functionality, let π be an $\mathcal{F}$ -hybrid protocol, and let ρ be a protocol that $\mathcal{H}$ -EUC-realizes $\mathcal{F}$. Then protocol π^ρ $\mathcal{H}$ -EUC-emulates π .*

Proof. The proof of Theorem C.2 follows the same steps as the proof of Theorem 4 (see e.g. the proof in [Can00]). The only difference is in the construction of the distinguishing environment Z_π (see there). Recall that Z_π takes an environment Z that distinguishes between an execution of π and an execution of π^ρ , and uses it to distinguish between an execution of ρ and an ideal evaluation of $\mathcal{F}$. For this purpose, Z_π emulates for Z an execution of π^ρ .

Now, in the presence of the helper $\mathcal{H}$, Z_ρ must emulate for Z also the interaction with $\mathcal{H}$. Note that Z_π cannot run $\mathcal{H}$ on its own, since $\mathcal{H}$ may well be super-polynomial in complexity. Instead, Z_π will forward to the external instance of $\mathcal{H}$ each message sent to $\mathcal{H}$ by Z . Similarly, whenever any of the corrupted parties that Z_π locally runs sends a message to $\mathcal{H}$, Z_π externally invokes a party with the same ID and code, corrupts it, and instructs it to send the query to the external instance of $\mathcal{H}$. The responses of $\mathcal{H}$ are handled analogously.

Note that the proof uses the fact that the helper functionality $\mathcal{H}$ does not take messages directly from the adversary. Indeed, Z_π cannot emulate for the external instance of $\mathcal{H}$ messages coming from the adversary. $\square$

D Black-Box UC-Secure Protocols in $\mathcal{H}$ -EUC Model

In this work, we consider UC-security with the a super-polynomial time helper that help breaks commitments of our black-box robust CCA secure commitment scheme $\langle C, R \rangle$. More precisely, it proceeds as described in Figure 6

Functionality $\mathcal{H}$
Corrupted Parties: Upon receiving an input $(\text{Corrupt}, P_i, \text{sid})$ from the environment, record $(\text{Corrupt}, P_i, \text{sid})$.
Initialization: Upon receiving an input $(\text{Init}, P_i, \text{sid}, k)$ from party P_i in the protocol instance sid, if there is no previously recorded tuple $(\text{Corrupt}, P_i, \text{sid})$ or there is a previously recorded session (P_i, sid, k), ignore this message; otherwise, initialize a session of $\langle C, R \rangle$ with $\mathcal{O}$ using identity (P_i, sid), and record session (P_i, sid, k).
Accessing $\mathcal{O}$: Upon receiving an input $(\text{Msg}, P_i, \text{sid}, k, m)$ from party P_i in the protocol instance sid, if there is no previously recorded session (P_i, sid, k), ignore the message; otherwise, forward m to $\mathcal{O}$ in the k^{th} session that uses identity (P_i, sid), obtain a reply m', and return $(\text{Msg}, P_i, \text{sid}, k, m')$ to P_i.

Figure 6: The ideal functionality $\mathcal{H}$

D.1 Proof of Lemma 2

In this section, we first recall the protocol Π_{OT} that $\mathcal{H}$ -EUC emulates $\mathcal{F}_{\text{OT}}$ and then prove its security. The construction relies on the T_{OT} -round mS-OT protocol $\langle S, R \rangle$ and the CCA-secure

commitment scheme $\langle C, R \rangle$. On common input 1^n , the sender and the receiver of the protocol Π_{OT} on private inputs (v_0, v_1) and u respectively proceed as follow:

Stage 1: The sender chooses a random subset $\Gamma_R \subseteq [20n]$ of size n and commits to Γ_R using $\langle C, R \rangle$.

The receiver chooses a random subset $\Gamma_S \subseteq [20n]$ of size n and another random subset $\Gamma \subseteq [18n]$ of size n ; it then commits to both Γ_S and Γ using $\langle C, R \rangle$.

Stage 2 (Coin-Tossing):

Receiver Random-Tape Generation: The receiver chooses $20n$ random strings $(a_1^R, \dots, a_{20n}^R)$ and commits to them using $\langle C, R \rangle$. The sender sends $20n$ random strings $(b_1^R, \dots, b_{20n}^R)$. The receiver calculates $r_i^R = a_i^R \oplus b_i^R$ for every $i \in [20n]$, and interprets r_i^R as $c_i \parallel \tau_i^R$, where c_i will be used as the receiver's input bit, and τ_i^R the random tape in the OT executions below.

Sender Random-Tape Generation: The sender chooses $20n$ random strings $(a_1^S, \dots, a_{20n}^S)$ and commits to them using $\langle C, R \rangle$. The receiver sends $20n$ random strings $(b_1^S, \dots, b_{20n}^S)$. The sender calculates $r_i^S = a_i^S \oplus b_i^S$ for every $i \in [20n]$, and interprets r_i^S as $s_i^0 \parallel s_i^1 \parallel \tau_i^S$, where s_i^0 and s_i^1 will be used as the sender's two input bits, and τ_i^S the random tape in the OT executions below.

Stage 3 (OT with Random Inputs): The sender and the receiver participates in $20n$ executions of the OT protocol $\langle S, R \rangle$ in parallel, where the sender acts as S and the receiver acts as R . In the i^{th} execution of $\langle S, R \rangle$, the sender uses inputs s_i^0, s_i^1 and random tape r_i^S and the receiver uses input c_i and random tape r_i^R . At the end of the execution, the receiver obtains outputs $\tilde{s}_1 \dots \tilde{s}_{20n}$.

Stage 4 (Cut-and-Choose—Honesty Checking):

Sender Honesty Checking: The receiver opens Γ_S and sender responds as follows: for every $j \in \Gamma_S$, the sender opens the j^{th} commitments of $\langle C, R \rangle$ in Stage 2 to $\tilde{a}_j^S$. The receiver checks if the openings are valid, and if for every $j \in \Gamma_S$, the sender acted honestly in the j^{th} OT execution according to $\tilde{a}_j^S \oplus b_j^S$. The receiver aborts if not.

Receiver Honesty Checking: The sender opens Γ_R and receiver responds as follows: for every $j \in \Gamma_R$, the receiver opens the j^{th} commitments of $\langle C, R \rangle$ in Stage 2 to $\tilde{a}_j^R$. The sender checks if the openings are valid and if for every $j \in \Gamma_R$, the receiver acted honestly in the j^{th} OT execution according to $\tilde{a}_j^R \oplus b_j^R$. The sender aborts if not.

Stage 5 (Combiner): Set $\Delta = [20n] - \Gamma_R - \Gamma_S$ (i.e., Δ is the set of unopened locations). For every $i \in \Delta$ The receiver computes $\alpha_i = u \oplus c_i$ and sends α_i . The sender responds as follows: It computes a $10n$ -out-of- $18n$ secret-sharing of v_0 ; without loss of generality, we index shares in that secret-sharing with elements in Δ ; let the secret-sharing be $\rho^0 = \{\rho_i^0\}_{i \in \Delta}$. Similarly, it also computes a $10n$ -out-of- $18n$ secret-sharing $\rho^1 = \{\rho_i^1\}_{i \in \Delta}$ for v_1 . Then the sender computes $\beta_i^b = \rho_i^b \oplus s_i^{b \oplus \alpha_i}$ for every $i \in \Delta$ and sends back all the β_i^b 's.

The receiver after receiving all the β_i^b 's, computes shares corresponding to the u^{th} input as $\tilde{\rho}_i = \beta_i^u \oplus \tilde{s}_i$ for every $i \in \Delta$, and sets $\tilde{\rho} = \{\tilde{\rho}_i\}_{i \in \Delta}$.

Stage 6 (Cut-and-Choose—Consistency Checking): The receiver opens to Γ . Then for every $j \in \Gamma \cap \Delta$, the sender reveals the two inputs $\hat{s}_j^0$ and $\hat{s}_j^1$ and random tape $\hat{\tau}_j^S$ that it uses in the j^{th} OT execution in Stage 3. The receiver checks if the sender acts honestly according to input $(\hat{s}_j^0, \hat{s}_j^1)$ and random tape $\hat{\tau}_j^S$ and aborts if not.

Finally the receiver checks whether $\tilde{\rho}$ computed in Stage 5 is $17n$ -close to a valid codeword w (that is, it agrees with w at $17n$ locations), and if for every $j \in \Gamma \cap \Delta$, w_j is equal to $\beta_j^u \oplus \hat{s}_j^{u \oplus \alpha_j}$. If so it outputs the value v encoded in w ; otherwise, it aborts.

Next we proceed to show that Π_{OT} is indeed a secure realization of $\mathcal{F}_{OT}$. Below we describe the technique for simulating the protocol execution of Π_{OT} in the ideal-world, where parties have access to the ideal commitment functionality $\mathcal{F}_{OT}$, and give a proof that the simulation in the ideal-world setting is indistinguishable from a real-world execution of Π_{OT} . Recall that we only need to prove that Π_{OT} $\mathcal{H}$ -EUC-emulates $\mathcal{F}_{OT}$; hence in both the ideal and real worlds, the environment and the adversary have access to the $\mathcal{H}$ functionality.

Let A be any $\mathcal{PPT}$ adversary and Z any $\mathcal{PPT}$ environment. The simulator Sim for A in the ideal world internally simulates a real-world execution with A on auxiliary input z : it simulates A 's interaction with the environment Z and the functionality $\mathcal{H}$, by simply forwarding the communications between A and Z or $\mathcal{H}$; furthermore, it simulates messages belonging to the OT protocol Π_{OT} for A as follows:

Strategy 1: If the Sender (P_i) is honest and the Receiver (P_j) is corrupted, the simulator needs to be able to extract the choice u of the receiver (controlled by A) so that it can send u to the ideal OT functionality $\mathcal{F}_{OT}$ to obtain an input v_u , and simulate the view of A without knowing the other input v_{1-u} .

Towards this, the simulator Sim first acts honestly in Stage 1 to 4 except the following: It forwards all the commitments of $\langle C, R \rangle$ from A in Stage 1 and 2 to the helper functionality $\mathcal{H}$. Since the receiver P_j is corrupted, $\mathcal{H}$ accepts commitments with identity P_j from Sim , and returns Sim the unique committed value if the commitment is valid and $\perp$ otherwise. These committed values include Γ_S and Γ committed to by A in Stage 1 and all the random strings a_i^R for $i \in [20n]$ committed to by A in Stage 2, which allows Sim to recover the inputs and random tapes $\{c_i, \tau_i^R\}_{i \in [20n]}$ that A is supposed to use in the Stage 3 OT executions. Then for every $j \in [20n]$, Sim checks whether A behaves honestly according to c_j, τ_j^R in the j^{th} OT execution in Stage 3, and sets Φ to be the set of locations in which A cheats. Next, if A successfully completes the first 4 stages, Sim needs to extract its input choice u and simulate the Stage 5 sender's message. To do so, it first extracts u by counting how many shares out of the $18n$ shares $\rho^0 = \{\rho_i^0\}_{i \in \Delta}$ and $\rho^1 = \{\rho_i^1\}_{i \in \Delta}$ that A will get (in Stage 5 and 6) for each input v_0 and v_1 as follows:

- For every location $j \in \Delta$ and also in Γ , since the sender's inputs s_j^0 and s_j^1 will be revealed in stage 6, Sim counts that A obtains one more share for both ρ^0 and ρ^1 .
- For every location $j \in \Delta$ and also in Φ , A has cheated in the j^{th} OT in Stage 3 and thus may obtain both of the sender's inputs s_j^0 and s_j^1 in that OT execution; recall that in Stage 5 of the protocol, the two shares ρ_j^0 and ρ_j^1 will be covered using the sender's inputs s_j^0 and s_j^1 as one-time pads. Therefore, after receiving the Stage 5 message (which contains $\beta_j^b = \rho_j^b \oplus s_j^{b \oplus \alpha_j}$), A will be able to recover both shares. Thus Sim counts that A again obtains one more share for both ρ^0 and ρ^1 .
- For the rest of locations $j \in \Delta - \Phi - \Gamma$, since A acts honestly using input c_j and the two sender's inputs s_j^0 and s_j^1 will not be revealed in Stage 6, A obtains $s_j^{c_j}$ through the OT execution while $s_j^{1-c_j}$ remains computationally hidden. Thus A later can only recover the share $\rho_j^{c_j \oplus \alpha_j}$. Therefore, Sim counts that A gets one more share of $\rho^{c_j \oplus \alpha_j}$.

Then to simulate the sender's Stage 5 message, Sim proceeds as follows: If for both inputs A gets more than $10n$ shares, the simulator Sim outputs fail and aborts. Otherwise, if for only

one input A gets more than $10n$ shares, Sim sends its index b^* to the ideal functionality $\mathcal{F}_{OT}$ and receives a value w ; it then sets $v_{b^*} = w$ and sets v_{1-b^*} to a random bit, and complete the rest of the simulation by following the honest sender's strategy using v_{b^*} and v_{1-b^*} as inputs. Finally, if for none of the inputs A gets more than $10n$ shares, then Sim simply sets both v_0 and v_1 to random bits and complete the simulation honestly according to these two values.

Strategy 2: If the Sender (P_i) is corrupted and the Receiver (P_j) is honest, the simulator needs to simulate the view of the sender (controlled by A) without knowing the choice of the receiver and extracts the two inputs from A .

Towards this, first note that during the whole execution of Π_{OT} , the only message that depends on the receiver's choice u is the Stage 5 receiver's message, consisting of $\{\alpha_j\}_{j \in \Delta}$ that are supposed to be set to $\alpha_j = u \oplus c_j$. The simulator Sim simulates the α_j 's by simply sending random bits (and emulates the rest of the receiver's messages for A honestly). Furthermore, to extract the two inputs of the sender (controlled by A), Sim proceeds as follows: It forwards all the commitments of $\langle C, R \rangle$ from A in Stage 1 and 2 to the helper functionality $\mathcal{H}$. Since the sender P_i is corrupted, $\mathcal{H}$ accepts commitments with identity P_i from Sim , and returns Sim the unique committed value if the commitment is valid and $\perp$ otherwise. These committed values include Γ_R committed to by A in Stage 1 and all the random strings a_i^S for $i \in [20n]$ committed to by A in Stage 2, which allows Sim to recover the inputs and random tapes $\{s_i^0, s_i^1, \tau_i^R\}_{i \in [20n]}$ that A is supposed to use (as a sender) in the Stage 3 OT executions. Next, if A completes the execution of Π_{OT} successfully, Sim extracts shares of the sender's inputs by computing $\hat{\rho}^b = \left\{ \hat{\rho}_j^b = \beta_j^b \oplus s_j^{b \oplus \alpha_j} \right\}_{j \in \Delta}$ for $b \in \{0, 1\}$ (The rationale behind this extraction strategy is that, for every input b , the sender of the protocol Π_{OT} is supposed to send "encryption" $\{\beta_j^b\}$ of the shares $\{\rho_j^b\}$ of that input in Stage 5, hidden using the appropriate inputs $\{s_j^{b \oplus \alpha_j}\}$ of the OT executions). Given the shares $\hat{\rho}^0$ and $\hat{\rho}^1$, Sim reconstructs inputs $\hat{v}^0$ and $\hat{v}^1$ as follows: For every b , it checks whether $\hat{\rho}^b$ is $16n$ -close to a valid codeword $\hat{w}^b$, and whether $\hat{w}^b$ passes the consistency check in the last stage, that is, if $\hat{w}^b$ agrees with $\beta_j^b \oplus s_j^{b \oplus \alpha_j}$ for all $j \in \Gamma$; If so, then it sets $\hat{v}^b$ to the value encoded in $\hat{w}^b$; otherwise it sets $\hat{v}^b = \perp$. Finally Sim sends the two values $\hat{v}^0$ and $\hat{v}^1$ externally to the OT functionality.

Strategy 3: If both the Sender (P_i) and the Receiver (P_j) are honest, the simulator needs to simulate the transcript of an honest execution of Π_{OT} for A without knowing the inputs of the honest players. To do so, it simply generates the transcript of an honest execution of Π_{OT} using inputs all 0.

Below we analyze each of the simulation strategies above, and show that the environment Z 's interactions with S in the ideal-world is indistinguishable from that with A in the real-world in each of the cases.

Analysis of the first case: Consider the following five hybrids:

Hybrid H_1 : Hybrid H_1 proceeds identically to the ideal execution, except that: In Stage 2, for every location j that the sender would not need to reveal the randomness, that is, $j \notin \Gamma_S \cup \Gamma$ (recall that the simulator obtains Γ_S and Γ by forwarding A 's commitments to the helper functionality $\mathcal{H}$ at the end of Stage 1), the simulator simulates the j^{th} commitment of $\langle C, R \rangle$ to A by committing to 0 instead of a_j^S . Since these commitments all have identities belonging to an honest player, and the helper functionality $\mathcal{H}$ only breaks

commitments with identities of corrupted parties, it follows from the CCA-security of $\langle C, R \rangle$ that the ideal-execution is indistinguishable from H_1 .

Hybrid H_2 : Hybrid H_2 proceeds identically to H_1 except that in Stage 5, instead of always using the sender's inputs s_j^0 and s_j^1 for $j \in \Delta$ to hide the shares $\rho^0 = \{\rho_i^0\}_{i \in \Delta}$ and $\rho^1 = \{\rho_i^1\}_{i \in \Delta}$, the simulator replace those inputs s_j^b 's that are computationally hidden from A with random strings to hide the shares. More precisely, recall that in every location $j \in \Delta - \Phi - \Gamma$, A acts honestly in the OT execution (using input c_j) and the sender's inputs are not revealed in Stage 6; thus the input $s_j^{1-c_j}$ is computationally hidden. Then, instead of using $s_j^{1-c_j}$ as a one-time pad to hide one of the j^{th} shares ρ_j^0 or ρ_j^1 in Stage 5, Sim uses a truly random string.

We claim that H_2 is indistinguishable from H_1 . Towards showing this, we first show that it follows from the security of the OT protocol $\langle S, R \rangle$ against semi-honest receiver that the views of a *malicious* receiver R^* in the following two experiments are indistinguishable.

In both experiments, the receiver R^* on input a choice u , random input τ and auxiliary input z , first engages in an execution with the honest sender S with inputs s_0 and s_1 chosen at random. After the execution with S completes, R^* receives the two inputs s_0 and s_1 in the first experiment. On the other hand, what R^* receives in the second experiment depends on whether it has acted honestly according to inputs u and τ : If it is dishonest, then it still receives s_0 and s_1 ; otherwise, if it is honest, it receives s_u and a random bit s' .

It follows from the security against semi-honest receivers that when R^* acts honestly according to a choice u , the sender's input s_{1-u} that is not chosen is computationally hidden, and thus R^* cannot tell apart later whether it receives s_{1-u} or a random bit. Therefore its views in the above two experiments are indistinguishable. It further follows from a simple hybrid argument that, no malicious receiver can tell apart the two experiment even if they are executed in parallel.

Then, since the only difference between hybrid H_1 and H_2 lies in whether the sender's message in Stage 5 is generated using honest inputs $s_j^{1-c_j}$ or random strings, for those locations $j \in \Delta - \Phi - \Gamma$ where the adversary A acts honestly in the OT execution according to the inputs and random tapes decided in Stage 2. It then follows from the indistinguishability of the above two experiments (executed in parallel) that the sender's messages in Stage 5 in H_1 and H_2 are indistinguishable. Then, by the 1-robustness of the CCA-secure commitment $\langle C, R \rangle$, we have that the view of A in the two hybrids are indistinguishable. Thus H_1 and H_2 are indistinguishable.

Hybrid H_3 : This hybrid proceeds identically to H_2 except the following: The ideal functionality $\mathcal{F}_{\text{OT}}$ does not expect to receive a choice from the simulator and instead directly discloses both of the sender's inputs v_0 and v_1 to the simulator; then after the simulator extracts the adversary's choice u , instead of using inputs v_u and a random bit to simulate the Stage 5 message as in H_2 , the simulator uses v_0 and v_1 .

To show that H_3 is indistinguishable from H_2 , we first prove that due to the cut-and-choose procedure in Stage 4, the probability that A cheats in a large number of—more than n —OT executions in Stage 3 without being caught is negligible.

Claim 4. *In hybrid H_4 , the probability that A cheats in more than n OT executions in Stage 3, and successfully passes the receiver's honesty check in Stage 4 is negligible.*

We remark that this claim holds, even if A receives a commitment to the set of locations Γ_R to be opened in the cut-and-choose procedure in Stage 1. This is because the CCA-

security of the commitment guarantees that Γ_R remains hidden even if all the values that A commits to in Stage 2 are extracted via a committed-value oracle (since these commitments use identities of corrupted parties, different from the identity of the commitment to Γ_R , which is the identity of the honest sender.) Then since we can identify the locations where A cheats using these committed values efficiently, these locations must be computationally independent of Γ_R . Thus if A cheats in a large number of OT executions, one of them will be selected by Γ_R to check with overwhelming probability, causing A to fail in Stage 4. A formal proof of this claim is presented at the end of this section.

It follows directly from Claim 4 that except with negligible probability, if A completes Stage 4 successfully, then the total number of shares that it gets is at most $20n$; then, by the pigeon hold principle, it gets more than $10n$ shares for at most one input. This implies that the probability that the simulator outputs fail is negligible. Assume that S_4 does not output fail. Then the only difference between the simulation in hybrid H_2 and H_3 lies in how the Stage 5 sender's message is simulated. In H_2 , for the input that A gets more than $10n$ shares, the simulator obtains the true value through the OT functionality and thus simulate the corresponding part of the sender's message perfectly. On the other hand, for the inputs that A gets no more than $10n$ shares, it simulates shares of these inputs using shares of random bits. However, since A gets at most $10n$ shares of these random bits and the rest of shares are all covered by truly random strings in H_2 , switching the random bits to true inputs does not change the distribution of the simulated message (of Stage 5). Thus we conclude that the views of A in H_3 and H_4 are statistically close, and so are the executions of H_3 and H_4 .

Hybrid H_4 : Hybrid H_4 proceeds identically to H_3 except that in Stage 5, the simulator switches back to using the sender's inputs s_j^0 and s_j^1 in the OT executions to hide the shares $\rho^0 = \{\rho_j^0\}_{j \in \Delta}$ and $\rho^1 = \{\rho_j^1\}_{j \in \Delta}$ (instead of using random strings to hide part of the shares). In other words, H_4 reverses the changes done in H_2 . Then it follows from the same argument as in H_2 that, by the 1-robustness of the commitment $\langle C, R \rangle$, the hybrid H_4 proceeds indistinguishably from H_3 .

Hybrid H_5 : Hybrid H_5 proceeds identically to H_4 except that, in Stage 2, for every location j that the sender do not need to reveal the randomness, that is, $j \notin \Gamma_S \cup \Gamma$, the simulator switches the j^{th} commitment of $\langle C, R \rangle$ to A from committing to 0 back to committing to a_j^S . That is, H_4 reverses the changes done in H_1 . Then it follows from the same argument as in H_1 that by the CCA-security of $\langle C, R \rangle$, the hybrid H_5 proceeds indistinguishably from H_4 .

Finally note that in H_5 , the simulator receives from $\mathcal{F}_{\text{OT}}$ both inputs v_0 and v_1 , and internally emulates the real-execution with A perfectly. Therefore, by a hybrid argument, we have that the real execution is indistinguishable from an execution of H_1 , which in turn is indistinguishable from the ideal execution. Thus we conclude that the simulator Sim is constructed correctly.

Analysis of the second case: Consider the following sequence of hybrids.

Hybrid $\tilde{H}_1$: This hybrid proceeds identically to the ideal execution, except the following: In Stage 2, for every location j that the receiver do not need to reveal the randomness, that is, $j \notin \Gamma_R$ (recall that the simulator obtains Γ_R by forwarding A 's commitments to the helper functionality $\mathcal{H}$ at the end of Stage 1), the simulator simulates the j^{th} commitment of $\langle C, R \rangle$ from the receiver to A , by committing to 0 instead of a_j^R . Since

these commitments all have the identity of the honest receiver, and the helper functionality $\mathcal{H}$ only breaks commitments with identities of corrupted parties, it follows from the CCA-security of $\langle C, R \rangle$ that the ideal-execution is indistinguishable from $\tilde{H}_1$.

Hybrid $\tilde{H}_2$: This hybrid proceeds identically to $\tilde{H}_1$ except the following: the ideal functionality $\mathcal{F}_{OT}$ discloses the external receiver's choice u to the simulator; furthermore, in Stage 5, instead of simulating all the α_j 's using random bits, the simulator computes them honestly as $\alpha_j = u \oplus c_j$ (where c_j 's are the inputs that the receiver uses in the Stage 3 OT executions).

We claim that $\tilde{H}_2$ is indistinguishable from $\tilde{H}_1$. Towards showing this, we first show that it follows from the security of the OT protocol $\langle S, R \rangle$ against malicious senders that the views of a malicious sender S^* in the following two experiments are indistinguishable.

In both experiments, the sender S^* first participates in an interaction with an honest receiver R using a random inputs u . After the execution with R , in the first experiment, S^* receives the input u , whereas in the second experiment, it receives another independently sampled random bit u' .

It follows from the security against malicious senders of the OT protocol that the views of the malicious sender S^* in the above two experiments are indistinguishable. Furthermore, it follows from a simple hybrid argument that, no malicious receiver can tell apart the above two experiments even if they are executed in parallel.

Then, note that the only difference between hybrid $\tilde{H}_1$ and $\tilde{H}_2$ lies in whether in Stage 5, the α_j 's (for $j \in \Delta$) from the receiver are simulated using random bits or generated honestly as $u \oplus c_j$; in other words, the difference is whether the α_j 's are computed as the sum of u and a random bit (yielding a random bit) or c_j (yielding $u \oplus c_j$). It then follows from the indistinguishability of the above two experiments (executed in parallel) that the receiver's messages in Stage 5 in $\tilde{H}_1$ and $\tilde{H}_2$ are indistinguishable. Thus, by the 1-robustness of the CCA-secure commitment $\langle C, R \rangle$, we have that the view of A and the output of the external OT receiver (which will be $\hat{v}_u$) in the two hybrids are indistinguishable. Hence $\tilde{H}_1$ and $\tilde{H}_2$ are indistinguishable.

Hybrid $\tilde{H}_3$: This hybrid proceeds identically to $\tilde{H}_2$ except that, in Stage 2, for every location j that the receiver do not need to reveal the randomness, that is, $j \notin \Gamma_R$, the simulator switches the j^{th} commitment of $\langle C, R \rangle$ from the receiver to A from a commitment to 0 back to a commitment to a_j^S . That is, $\tilde{H}_3$ reverses the changes done in $\tilde{H}_1$. Then it follows from the same argument as in $\tilde{H}_1$ that by the CCA-security of $\langle C, R \rangle$, the hybrid $\tilde{H}_3$ proceeds indistinguishably from $\tilde{H}_2$. We remark that in $\tilde{H}_3$ the simulator essentially emulates the execution of Π_{OT} for A perfectly as an honest receiver (using the external receiver's true input u that it gets from $\mathcal{F}_{OT}$), except that it tries to extract the two sender's inputs from A at the end of the execution.

Hybrid $\tilde{H}_4$: This hybrid proceeds identically to $\tilde{H}_3$ except the following: The external receiver no longer interacts with $\mathcal{F}_{OT}$, and instead, simply outputs a value that the simulator feeds it. On the other hand, the simulator internally emulates the execution of Π_{OT} for A by following the honest receiver's strategy (as in $\tilde{H}_3$), obtaining an output v ; it then skips extracting the sender's inputs $\hat{v}_0$ and $\hat{v}_1$ and simply feeds the external receiver the value v as the its output.

We show that the output of the environment in $\tilde{H}_4$ is statistically close to that in $\tilde{H}_3$. Since the only difference between the two hybrids lies in how the outputs of the external receiver are derived, it suffices to show that except with negligible probability, the outputs of the external receiver in the two hybrids are the same. In $\tilde{H}_4$, the external receiver directly outputs the value v the simulator feeds it, which is just the output of

an honest receiver of Π_{OT} with input u (emulated by the simulator for A). In $\tilde{H}_3$, the external receiver obtains the u^{th} input from $\mathcal{F}_{\text{OT}}$, which is $\hat{v}_u$ extracted by the simulator from A . Recall that both v and $\hat{v}_u$ are derived in two steps:

- In the first step, shares of the two values are recovered. The honest receiver obtains shares of v by computing $\tilde{\rho} = \{\tilde{\rho}_j = \beta_j^u \oplus \tilde{s}_j\}_{j \in \Delta}$, where $\tilde{s}_j$'s are the outputs it obtains in the Stage 3 OT executions with inputs c_j 's. On the other hand, the simulator extracts shares of $\hat{v}_u$ as $\hat{\rho}^u = \{\hat{\rho}_j^u = \beta_j^u \oplus s_j^{u \oplus \alpha_j}\}_{j \in \Delta}$, where the s_j^b 's are the inputs that A is supposed to use in the OT executions.
- In the second step, a codeword is recovered from the shares: The honest receiver recovers w that is $17n$ -close to $\tilde{\rho}$, whereas the simulator recovers $\hat{w}^u$ that is $16n$ -close to $\hat{\rho}^u$. Then both codewords are checked for consistency, that is whether they agrees with $\beta_j^u \oplus \hat{s}_j^{u \oplus \alpha_j}$ at locations $j \in \Gamma$, where the $\hat{s}_j^b$'s are A 's actual inputs in the OT executions revealed in the last stage. If w (or $\hat{w}^u$ respectively) passes the consistency check, then v (or $\hat{v}_u$ resp.) is set to the value encoded in w (or $\hat{w}^u$ resp.); otherwise, it is set to $\perp$.

Towards showing that v and $\hat{v}_u$ are (almost) always the same, Consider the following two possible cases: $\tilde{\rho}$ is $17n$ -close to a valid codeword w or not.

- If it is, (in $\tilde{H}_4$) the honest receiver will recover w from $\tilde{\rho}$. We show that the simulator will recover the same codeword w from $\hat{\rho}^u$ (i.e., $\hat{w}^u = w$). This relies on the following claim:

Claim 5. *Let $\tilde{\rho}$ and $\hat{\rho}^u$ be defined as above. Then, except with negligible probability, the shares $\tilde{\rho}$ and $\hat{\rho}^u$ are $17n$ -close to each other.*

This claim essentially follows from the fact that due to the cut-and-choose procedure, the probability that A cheats in more than n OT executions in Stage 3, and yet, passes the sender's honesty check in Stage 4 is negligible. (This follows from the same proof as Claim 4). Therefore, there are at least $17n$ locations where A acted honestly (in the OT executions), meaning that the output $\tilde{s}_j$ of the honest receiver in these OT executions equals to $s_j^{c_j}$, which in turn equals to $s_j^{u \oplus \alpha_j}$ since $\alpha = u \oplus c_j$ in $\tilde{H}_3$ and $\tilde{H}_4$. This implies that $\tilde{\rho}$ and $\hat{\rho}^u$ agree with each other at at least $17n$ locations.

Therefore, except with negligible probability, $\hat{\rho}^u$ is $16n$ -close to w . Then the simulator will uniquely recover w as well (i.e., $\hat{w}^u = w$). After that, both the honest receiver and the simulator conduct the same consistency check against w , leading to the same outputs $v = \hat{v}_u$.

- Otherwise, if $\tilde{\rho}$ is n -far away from any valid codeword, the honest receiver sets v to $\perp$. We need to show that the simulator will also set $\hat{v}_u$ to $\perp$ with overwhelming probability. Suppose not, then the simulator must have recovered a codeword $\hat{w}^u$ from $\hat{\rho}^u$. By our hypothesis, $\hat{w}^u$ is n -far away from $\tilde{\rho}$; let Ψ be the set of locations j at which $\hat{w}^u$ and $\tilde{\rho}$ differ. Then we show that the probability that $\hat{w}^u$ passes the consistency check is negligible. Formally,

Claim 6. *Let Ψ be defined as above. Then, the probability that $|\Psi| > n$ and A successfully passes the consistency check in Stage 6 is negligible.*

This claim essentially follows from the fact that due to the cut-and-choose procedure, the probability that Ψ is large but none of the locations j in Ψ is checked in the last stage (i.e., $\Psi \cap \Gamma = \emptyset$) is negligible. (This follows from a similar proof as Claim 4). With overwhelming probability there is a location j in Ψ being checked, forcing A to reveal the inputs $\hat{s}_j^0$ and $\hat{s}_j^1$ it uses in that OT execution, which also reveals the

difference between $\tilde{\rho}_j$ and $\hat{w}_j^u$. That is,

$$\tilde{\rho}_j = \beta_j^u \oplus \tilde{s}_j = \beta_j^u \oplus \hat{s}_j^{c_j} = \beta_j^u \oplus \hat{s}_j^{u \oplus \alpha_j} \neq \hat{w}_j^u$$

Thus, except with negligible probability, the adversary fails to pass the consistency check, and $\hat{v}^u$ is set to $\perp$ as claimed.

By construction, in the last hybrid $\tilde{H}_4$, the view of A is emulated perfectly according to Π_{OT} and the output of the external OT receiver is the same as that of the honest receiver of Π_{OT} . Therefore, the output of the environment in hybrid $\tilde{H}_4$ is identically distributed to the real-execution. Then by a hybrid argument, we have that the real execution is indistinguishable to $\tilde{H}_1$, and thus the ideal execution. Thus the simulator Sim is constructed correctly.

Analysis of the third case: Consider the following sequence of hybrids:

Hybrid $\hat{H}_1$: This hybrid proceeds identically to the ideal execution except the following: The ideal $\mathcal{F}_{\text{OT}}$ functionality discloses the inputs (v_0, v_1) and u of the external sender and receiver to A , and furthermore, the simulator switches the sender's and receiver's inputs that it uses for simulating the transcript of Π_{OT} (for A) from all 0 to $(0, v_1)$ and 0. It follows from the analysis of the first case above that when considering a semi-honest adversary acting as the receiver of Π_{OT} and using input 0, the adversary cannot tell apart if the honest sender is using inputs $(0, 0)$ or $(0, v_1)$. Thus the simulated transcripts in $\hat{H}_1$ and the ideal execution must be indistinguishable. Furthermore since no player is corrupted, the helper functionality $\mathcal{H}$ would not accept any query from the adversary, the indistinguishability of the simulated transcripts directly implies the indistinguishability of $\hat{H}_1$ and the ideal execution.

Hybrid $\hat{H}_2$: This hybrid proceeds identically to $\hat{H}_1$ except that the simulator switches the receiver's input that it uses for simulating the transcript of Π_{OT} from 0 to 1. It follows from the analysis of the second case above that when considering a semi-honest adversary acting as the sender of Π_{OT} , the adversary cannot tell apart if the honest receiver is using inputs 0 or 1. Thus the simulated transcripts in $\hat{H}_1$ and $\hat{H}_2$ must be indistinguishable. Therefore, it follow from a similar argument as in $\hat{H}_1$ that $\hat{H}_1$ and $\hat{H}_2$ are indistinguishable.

Hybrids $\hat{H}_3$ and $\hat{H}_4$: These two hybrids proceed identically to $\hat{H}_2$ except that the simulator switches the inputs that it uses for simulating the transcript of Π_{OT} from $((0, v_1), 1)$ in $\hat{H}_2$ to $((v_0, v_1), 1)$ in $\hat{H}_3$ and to $((v_0, v_1), u)$ in $\hat{H}_4$. It follows from the same argument as in $\hat{H}_1$ that hybrid $\hat{H}_3$ is indistinguishable to $\hat{H}_2$, and from the same argument as in $\hat{H}_2$ that hybrid $\hat{H}_4$ is indistinguishable to $\hat{H}_3$.

Finally, in the last hybrid, A receives the transcript of the execution of Π_{OT} using true inputs as in the real execution. Furthermore, since the outputs of the honest receiver in $\hat{H}_4$ is identical to that in the real execution (both equal to v_u), we have that the output of the environment in $\hat{H}_4$ is identically distributed to that in the real execution. Then by a hybrid argument we conclude that the real and ideal executions are indistinguishable.

Proof of Claim 4. Let Φ be the set of locations where A cheats in Stage 3. We note that Φ can be computed efficiently given the values that A commits to using $\langle C, R \rangle$ in Stage 2. Recall that the receiver's honesty check in Stage 4 requires the receiver to open all the commitments it sends in Stage 2 at locations in Γ_R . Therefore if $\Phi \cap \Gamma_R \neq \emptyset$, by the statistically binding property of $\langle C, R \rangle$, A would have been caught cheating. Thus if A successfully completes Stage 4, it must hold that $\Phi \cap \Gamma_R = \emptyset$. Now assume for contradiction that A cheats in more than n OT executions, but

successfully completes Stage 4 in H_4 (i.e., $|\Phi| > n$ and $\Phi \cap \Gamma_R = \emptyset$) with polynomial probability. Then we show that A can be used to violate the CCA-security of $\langle C, R \rangle$.

Consider a machine B that acts as a receiver of $\langle C, R \rangle$ externally with access to the committed-value oracle $\mathcal{O}$; internally, it emulates an execution of hybrid H_4 with A , by using $\mathcal{O}$ to implement the helper functionality $\mathcal{H}$, except the following: It forwards messages from the external committer C to A as the sender's commitment in Stage 1; then after Stage 3, it computes the set Φ using the values that A commits to in Stage 2 obtained through the helper functionality $\mathcal{H}$ as in H_4 ; finally, it simply outputs Φ and aborts. It follows from the construction that, when B externally receives a commitment to Γ_R —a randomly chosen set of size n —using the identity of the honest sender, it internally emulates the execution of H_4 perfectly up to Stage 3; then by our hypothesis, with polynomial probability, it outputs a set Φ of size greater than n , disjoint with Γ_R . However, on the other hand, if B externally receives a commitment to 0 (still using the identity of the honest sender), then the probability that B outputs a set of size greater than n that is disjoint with the randomly chosen Γ_R is exponentially small. Finally since all the commitments that B queries to the committed-value oracle $\mathcal{O}$ have identities belonging to a corrupted party, different from the identity of the honest sender, we have that B violates the CCA security of $\langle C, R \rangle$. $\square$

Interactive Coding, Revisited

Kai-Min Chung
Cornell University
chung@cs.cornell.edu

Rafael Pass *
Cornell University
rafael@cs.cornell.edu

Sidharth Telang
Cornell University
sidtelang@cs.cornell.edu

March 16, 2013

Abstract

How can we encode a communication protocol between two parties to become resilient to adversarial errors on the communication channel? This question dates back to the seminal works of Shannon and Hamming from the 1940's, initiating the study of error-correcting codes (ECC). But, even if we encode each message in the communication protocol with a “good” ECC, the error rate of the encoded protocol becomes poor (namely $O(1/m)$ where m is the number of communication rounds). Towards addressing this issue, Schulman (FOCS'92, STOC'93) introduced the notion of *interactive coding*.

We argue that whereas the method of separately encoding each message with an ECC ensures that the encoded protocol carries the same amount of information as the original protocol, this may no longer be the case if using interactive coding. In particular, the encoded protocol may completely leak a player's private input, even if it would remain secret in the original protocol. Towards addressing this problem, we introduce the notion of *knowledge-preserving interactive coding*, where the interactive coding protocol is required to preserve the “knowledge” transmitted in the original protocol. Our main results are as follows.

- The method of separately applying ECCs to each message is essentially optimal: No knowledge-preserving interactive coding scheme can have an error rate of $1/m$, where m is the number of rounds in the original protocol.
- If restricting to computationally-bounded (polynomial-time) adversaries, then assuming the existence of one-way functions (resp. subexponentially-hard one-way functions), for every $\epsilon > 0$, there exists a knowledge-preserving interactive coding schemes with constant error rate and information rate $n^{-\epsilon}$ (resp. $1/\text{polylog}(n)$) where n is the security parameter; additionally to achieve an error of even $1/m$ requires the existence of one-way functions.
- Finally, even if we restrict to computationally-bounded adversaries, knowledge-preserving interactive coding schemes with constant error rate can have an information rate of at most $o(1/\log n)$. This results applies even to *non-constructive* interactive coding schemes.

*Pass is supported in part by a Alfred P. Sloan Fellowship, Microsoft New Faculty Fellowship, NSF Award CNS-1217821, NSF CAREER Award CCF-0746990, NSF Award CCF-1214844, AFOSR YIP Award FA9550-10-1-0093, and DARPA and AFRL under contract FA8750-11-2-0211. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the Defense Advanced Research Projects Agency or the US Government.

1 Introduction

The study of how to communicate over a noisy channel dates back to the seminal works of Shannon [Sha48] and Hamming [HAM50] from the 1940s, initiating the study of error-correcting codes. Roughly speaking, an error-correcting code encodes a k -bit message m into a ck bit message with the property that even if a fraction $\eta < 1$ of the bits of the encoded messages are adversarially changed, the original message m can be decoded; $R = 1/c$ is referred to as the *information rate* of the code, and η as the *error rate*. Efficiently encodable and decodable error correcting codes with constant information rate and error rate are known [Jus]; in fact, such codes can even be made linear-time encodable and decodable [Spi96].

In this work, we are interested in the question of how to encode interactive communication: Given an interactive protocol $\pi = (A, B)$ between two parties, how can we encode this protocol to become resilient to adversarial errors? A naive approach would be to simply apply a “good” (i.e., constant information and error rate) error correcting code to each message of the protocol. This results in a poor error rate: if the protocol has m rounds and each round requires sending a k -bit message, then it suffices to corrupt $O(k)$ out of the $O(km)$ communicated bits (that is, a fraction $O(1/m)$) to ensure an incorrect decoding. To address this problem, Schulman [Sch92, Sch93, Sch96] introduced the notion of *interactive coding*. Roughly speaking, an interactive coding scheme is an algorithm $Q = (Q_1, Q_2)$ such that for any interactive protocol $\pi = (A, B)$, (Q_1^A, Q_2^B) emulates the interaction of (A, B) in the sense that (with overwhelming probability) the execution of the actual protocol (A, B) and the “encoded protocol” (Q_1^A, Q_2^B) yield the same outputs. Additionally, the protocol (Q_1^A, Q_2^B) is error-resilient: the execution of (Q_1^A, Q_2^B) yields the same outputs even if a η fraction of the communication is adversarially corrupted, where η is an error rate. Schulman [Sch96] presented an interactive coding scheme with constant information and error rate. Schulman’s construction achieved an error rate of $1/240$, which was later improved by Braverman and Rao [BR11] and Braverman [Bra12] to (close to) $1/8$. The interactive coding scheme Q in their works, however, requires exponential or subexponential time. Gelles, Moitra and Sahai [GMS11] showed to get a polynomial-time interactive coding (with constant information and error rate) for the case of uniformly distributed (as opposed to adversarial) errors. More recently, the elegant work of Brakerski and Kalai [BK12] showed how to get a polynomial-time interactive coding handling also adversarial errors, with a constant information rate and an error rate of (close to) $1/32$, and even more recently Brakerski and Naor [BN13] show how to get quasi-linear time interactive coding with constant information and error rate.

Interactive Coding, revisited When we encode messages using error correcting codes we ensure that the encoded messages carry exactly the same information as the original messages; in other words, they carry all the information in the original messages (or else we cannot decode), and additionally they do not carry any other information (say, about future messages). Consider, for instance, transmitting an “interactive exam” (e.g., an oral exam) in an error resilient way. The exam has the property that question 2 in the exam reveals the answer to question 1. Ideally, we would like to guarantee that the error resilient version of the exam does not allow the student (taking the exam) to see question 2 before it needs to provide the answer to question 1 (or else it can trivially answer question 1). Clearly this property would hold if we use the “naive approach” of separately encoding every message using an error correcting code, but as we shall see shortly, this property may no longer hold if we use interactive coding. Intuitively, the problem is that interactive coding (and in particular, the above-mentioned solutions), while guaranteeing that the encoded protocol carries at least the same amount of information as the original protocol, does not necessarily guarantee that the encoded protocol does not reveal more information than the original

protocol.

As another example, consider two mutually distrustful players that wish to run some secure cryptographic protocol (A, B) over a noisy channel. Can these players instead run an interactive coding (Q_1^A, Q_2^B) of (A, B) ? In other words, does the interactive coding preserve the security of the original underlying protocol? It is easy to see that the “naive approach” of separately encoding every message using an error correcting code preserves security of the underlying protocol. However, if we use interactive coding, this may no longer be the case. The problem is that the notion of interactive coding only requires that the encoded protocol (Q_1^A, Q_2^B) emulates (A, B) as long as both of the communicating parties are *honestly executing* the protocol. In particular, if one of the players is adversarial, it could be the case that the player gains more information when participating in the encoded protocol than it would have in the original protocol; for instance, player 1 may, by deviating from the protocol instructions in the encoded protocol (Q_1^A, Q_2^B) , learn something about player 2’s private input that is guaranteed to remain secret in the original protocol (A, B) (no matter what player 1 does).

The reason interactive coding schemes do not necessarily provide the desired guarantees in the above scenarios is that such schemes typically “bundle together” multiple rounds of interactions of the original protocol, and when an error in the communication is detected, the whole bundle is “replayed”. (Looking forward, as we show in Theorem 2, any interactive coding with a “good” error rate in fact needs to replay messages in this way.) This may allow an attacker to “fake” an error in the communication in order to get the bundle replayed, but this time change its messages, and as a consequence may learn two (or more) partial transcripts where the attacker’s messages are different: in essence, the encoded protocol gives the attacker the opportunity to “rewind” the honest player in original protocol. In the above “interactive exam” example such rewindings mean that the student may get knowledge of the second question before having to provide the answer to the first one; for the cryptographic protocol example, it is well-known that most (but not all, see [CGGM00]) cryptographic protocols are not secure under such rewindings: Consider, for instance, any of the classic zero-knowledge protocols (e.g., [GMR89, Blu86]); if the verifier can rewind the prover just once, it can completely recover the NP-witness used by the prover, although the protocols are zero-knowledge without such rewindings.

Knowledge-preserving interactive coding Towards addressing this problem, we here put forward, and study, the notion of *knowledge-preserving interactive coding*: Roughly speaking, we require not only that (Q_1^A, Q_2^B) conveys at least as much “knowledge” as (A, B) , but also that it does not convey more, even if one of the players adversarially deviates from the protocol instructions; that is, (Q_1^A, Q_2^B) preserves the knowledge transmitted in (A, B) . In other words, we require not only that (Q_1^A, Q_2^B) emulates (A, B) when the players are honest (only caring about their correct output and not trying to extract any other knowledge), but also that it is a good emulation when one of the players adversarially deviates from the protocol instructions (e.g., trying to obtain more knowledge about the other player’s input and potentially use it in the interaction). We formalize this notion through the classic *zero-knowledge* “simulation-paradigm” from cryptography [GMR89, GMW91]: We require that for every adversarial strategy $\tilde{A}^*$ for player 1 (resp. $\tilde{B}^*$ for player 2) participating in the encoded protocol $(\tilde{A}, \tilde{B}) = (Q_1^A, Q_2^B)$, there exists a “simulator” A^* (resp. B^*) such that the output of both players in the execution of $(\tilde{A}^*, \tilde{B})$ (resp. $(\tilde{A}, \tilde{B}^*)$) are indistinguishable from the outputs of players in the execution of (A^*, B) (resp. (A, B^*)). In other words, an adversary participating in the encoded protocol does not gain any more “knowledge” than it would have in the original protocol, and cannot affect the honest parties’ output more than it could have in the original protocol.

As we shall see, achieving knowledge-preserving interactive coding is significantly harder than “plain” interactive coding, and studying *resilience against only computationally bounded adversaries*, as was done by Lipton [Lip94] and Micali, Peikert, Sudan and Wilson [MPSW10] in the context of error correcting codes, is actually *essential* for achieving good error rates in the context of knowledge-preserving interactive coding.

1.1 Our Results

We are interested in knowledge-preserving interactive coding schemes $Q = (Q_1, Q_2)$ where Q_1 and Q_2 are efficient; we formalize this by requiring that Q_1, Q_2 receive as input the communication complexity ℓ and number of rounds m of the protocol (A, B) , and a security parameter n , and require that Q_1, Q_2 run in time polynomial in ℓ, m and n ; in the sequel, when referring to a knowledge-preserving interactive coding scheme, we only refer to such efficiently computable schemes.

The information-theoretic regime We start by stating the folklore result that the “naive approach” of separately encoding each message in the protocol with a good error correcting code is a knowledge-preserving interactive coding:

Theorem 1. *[Informally stated] There exists a knowledge-preserving interactive coding scheme Q with polynomial information rate and error rate $O(1/m)$ where m is the number of communication rounds in the original protocol.*

Our first result is a strong negative result for knowledge-preserving interactive coding, showing that the naive approach is essentially optimal (if requiring resilience against computationally unbounded adversaries).

Theorem 2. *[Informally stated] For every knowledge-preserving interactive coding scheme $Q = (Q_1, Q_2)$, every polynomial $m(\cdot)$, there exists an $m(n)$ -round protocol (A, B) such that (Q_1^A, Q_2^B) has an error rate of at most $1/m(n)$, where n is the security parameter. (In particular, no knowledge preserving coding scheme can have error rate $1/\text{poly}(n)$ where n is the security parameter).*

Let us provide a high-level overview of the proof of the theorem. The key idea is to come up with a protocol π having the property that the only way to make the protocol error resilient makes it possible for an attacker to “rewind” the honest players (just as what is done in known interactive protocols, as described above). Consider some interactive coding protocol $Q = (Q_1, Q_2)$ and let $M(n, m, \ell)$ be a polynomial upper bound on the number queries made by Q_1, Q_2 to its oracles (where n is the security parameter, m is the number of round in the protocol π to be encoded, and ℓ is the communication complexity of π). Consider the $m(n) = \text{poly}(n)$ -round “ping-pong” protocol π where each player $d \in \{1, 2\}$ gets an $M(n, 2mn, m) + 1$ -wise independent hash function $H_d : \{0, 1\}^{\text{poly}(n)} \rightarrow \{0, 1\}^n$ as input and proceeds as follows: player 1 computes and sends $a_1 = H_1(\emptyset)$ to player 2; player 2 computes and sends $b_1 = H_2(a_1)$ to player 1; player 1 computes and send $a_2 = H_1(a_1, b_1)$ to player 2, etc, for m rounds, and finally both player output the transcript of the interaction. That is, at each round, each player d , computes its next message by applying its hash function H_d to the current transcript. Note that in this protocol, by the unpredictability of the output of the hash functions, player 1, even if maliciously deviating from the protocol, will with overwhelming probability be able to obtain at most m distinct pairs $(q, H_2(q))$.

In contrast, as we show, unless the encoded protocol has an error rate less than $1/m$, a malicious player 1 in the encoded protocol can with inverse polynomial probability get $m + 1$ such pairs (and as such a malicious player 1 can learn something new in Q^π that it couldn’t have learnt in π). The key lemma needed to establish this shows that the encoded protocol “implicitly executes” the

original ping-pong protocol. More precisely, the rounds of the encoded protocol can be divided into “chunks”, where each chunk in the encoded protocol corresponds to a round in ping-pong protocol¹, and additionally by observing the oracle queries made by Q , we can read out a polynomial list of candidates for the current transcript of the ping-pong protocol; to establish this lemma we rely on the “elusiveness” property of the output of the hash functions (and the fact that Q queries the hash functions at most $M(n, 2mn, m)$ times).

Next, by an averaging argument, one of these chunks, say chunk i , must be shorter than a fraction $1/m$ of the total communication complexity of the encoded protocol. The idea now is for a malicious player 1 to honestly execute the encoded protocol using its actual input, except that during the i ’th chunk, the player acts as if its input was a random hash function H'_1 consistent with the transcript up until the end of chunk $i - 1$; that is, we switch the input only in chunk i , but make sure we pick an input that is consistent with the transcript so far. (Note that this attack is not necessarily efficient since picking an input consistent with the current transcript may not be computationally feasible.) Now, intuitively, since the chunk was “small”, by the error resilience property of the interactive coding scheme, with overwhelming probability, player 1 will finally output the same transcript (including m distinct pairs $(q, H_2(q))$) as if it had been running the protocol honestly. (Formalizing this requires showing that the attack performed by player 1 can be perfectly emulated by the channel).

Additionally, as we show, by observing the oracle queries made by Q_1 during the i ’th chunk (which corresponds to the i ’th round in the implicitly executed ping-pong protocol), player 1 may learn a *new* pair $(q', H_2(q'))$; intuitively, this follows since player 1 is using a new input in round i of the implicitly executed ping-pong protocol (but formally proving this claim is quite non-trivial). Thus, in essence, player 1, by using a different input in only chunk i manages to “rewind” player 2 in the implicit ping-pong protocol.

So, if player 1 could just identify the i ’th chunk, it can learn $m + 1$ distinct pairs $(q, H_2(q))$, which was not possible in π ; but it can simply guess the starting round of the i ’th chunk with inverse polynomial probability. Summarizing, in π , an attacker can learn $m + 1$ distinct pairs $(q, H_2(q))$ only with negligible probability, whereas in Q^π this can be done with inverse polynomial probability (if Q^π has a “non-trivial” error rate); thus, we are “blatantly” violating knowledge-preservance of Q .

The computational regime We next turn to consider *computational knowledge-preserving interactive coding*, where we only require resilience against computationally bounded adversaries: we only require the error-resilience property to hold against computationally-bounded channel adversaries, and the knowledge-preserving property to hold against computationally-bounded adversaries. We first present a positive result, showing that constant-error rate is possible (albeit at a sub-constant information rate):

Theorem 3. *[Informally stated] Assume the existence of one-way functions. Then, for every $\epsilon > 0$, there exists a knowledge-preserving interactive coding scheme with error rate $(1/12) - \epsilon$ and information rate $O(1/n^\epsilon)$ where n is the security parameter. If additionally subexponentially-hard one-way functions exists, the information rate can be improved to $O(1/\text{polylog} n)$.*

The idea behind this scheme is simple: The players start by exchanging verification keys for a signature scheme; the verification keys are appropriately padded to become “long” and then encoded using a good error-correcting code. Next, we run the original protocol, except that all messages in the protocol are signed and additionally encoded using a good error-correcting code.

¹These chunks may depend on the inputs of the players and the randomness of Q .

Whenever a player receives a message that does not have a valid signature, it requests to hear the message again. It is easy to show that this coding scheme is knowledge preserving (even against unbounded attackers); additionally, if the verification keys exchanged in the first round are long enough, then the scheme has error rate close to $\eta/4$, where η is the error rate of the error-correcting code. Using state of the art error-correcting codes this would yield an error rate of $1/16 - \epsilon$ (where $\epsilon > 0$ is an arbitrarily small constant). We can further improve the error rate by relying on an idea from [MPSW10]: since messages are signed and we only consider a computationally bounded channel, it in fact suffices to “list-decode” the error-correcting code used to encode the messages in the protocol (while still using unique decoding for error-correcting code used to encode the verification keys).² This allows us to improve the error rate to $1/12 - \epsilon$.

As our next result demonstrates, one-way functions are *necessary* to achieve a “non-trivial” error rate.

Theorem 4. *[Informally stated] Assume the existence of a computational knowledge-preserving interactive coding scheme with error rate $1/m$, where m is the number of communication rounds in the original protocol. Then one-way functions exist.*

The proof of Theorem 4 follows by carefully showing that the attack constructed in the proof of Theorem 2 can be “approximately” implemented in polynomial-time, if one-way functions do not exist.

We finally show that every computational knowledge-preserving interactive coding scheme with constant error rate must have an information rate of $\omega(1/\log n)$.

Theorem 5. *[Informally stated] Assume the existence of a computational knowledge-preserving interactive coding scheme with information rate R and error rate η . Then $R\eta \in \omega(1/\log(n))$, where n is the security parameter.*

Let us first mention that a weaker version of the above theorem, demonstrating that the information rate needs to be sub constant (as opposed to $\omega(1/\log n)$) can be obtained by carefully “scaling down” the proof of Theorem 2 by considering a constant-round ping-pong protocol where the length of each message is $O(\log n)$. To give the tight bound, we need to rely on an even more scaled down version where the length of the messages in the ping-pong protocol is just 1. In this regime, the previous proof no longer works: we can no longer ensure that the transcript from the ping-pong protocol can be decoded by observing all the oracle calls made by Q . (In the proof of Theorem 2 this was proven by relying on the elusiveness property of the image of the hash function, but since we now consider the range $\{0, 1\}$, this no longer holds.) Rather, we here provide a different information-theoretic definition of chunks and rely on the fact that the protocol is knowledge preserving to show that chunks are well-defined (to simplify the proof of this, we actually rely on a simpler variant of the ping-pong protocol). The idea, which turns out to be quite subtle to formalize, is that if a partial transcript contained information about, say, player 2’s message in round j , before player 1’s message in round j has been fully determined in the partial transcript, then intuitively, player 1 has the opportunity to (with non-negligible probability) change its message in round j as a function of player 2’s message in the same round, which isn’t possible in the original ping-pong protocol.

²[MPSW10] relies on this idea to show how to achieve an error-correcting code with error rate $1/2 - \epsilon$ if assuming a (noiseless) public-key infrastructure and a computationally-bounded channel. In our context, we do not have a public-key infrastructure, but our initial exchange of verification keys using a uniquely decodable error-correcting code can be viewed as a way to set-up the appropriate public-key infrastructure needed for their results.

Non-constructive knowledge-preserving interactive coding All the above-mentioned results rely on the standard notion of interactive coding where the algorithm $Q = (Q_1, Q_2)$ only uses the original protocol $\pi = (A, B)$ as a black-box (i.e., the encoded protocol is (Q_1^A, Q_2^B)). One may also consider a more relaxed notion of coding, where the encoded protocol uses the description of the protocol π in a *non-black-box* way, or is even *non-constructive*. We note that the proof of Theorem 6 is actually stronger than stated; we actually show that *every* protocol (not just those protocols obtained by accessing the original protocol π as a black-box) that preserves the knowledge transmitted in the 1-bit ping-pong protocol and has an error rate of $O(1)$, must have a communication complexity of at least $\omega(\log n)$.

Theorem 6. *[Informally stated] For every function $\eta(n) \geq O(1/\log n)$, there exists a protocol π with communication complexity $O(1/\eta(n))$ such that for every protocol π' that is a knowledge-preserving variant of π (even just w.r.t. computationally-bounded adversaries) and is computationally η -error resilient, the communication complexity of π' is at least $\omega(\log n)$.*

It is worthwhile to compare Theorem 6 with Theorem 2. As mentioned above, Theorem 6 is stronger than Theorem 2 in that it rules out also non-constructive interactive coding schemes. On the other hand, it is weaker in several other aspects: First, in Theorem 2 we “blatantly” violate knowledge preservance: we exhibit some explicit information that can only be learnt with negligible probability in π , but can be learnt with inverse polynomial probability in the encoded protocol. In contrast, in the proof of Theorem 6 we rely on the knowledge-preservance property in a stronger way (in particular, as mentioned above, we rely the knowledge-preservance property to show that the encoded protocol implicitly executed π , whereas in Theorem 2 this could be showed unconditionally). Secondly, the error rate achieved in Theorem 6 is weaker than the one rate achieved in Theorem 2. This, to some extent, is necessary, since Theorem 6 also rules out computational knowledge-preserving interactive coding, and as showed in our positive result (Theorem 3), an error rate of $O(1)$ can be achieved in this setting.

1.2 Overview of the Paper

In Section 2 we provide some notation and preliminaries. In Section 3 we formally define the notion of knowledge-preserving interactive coding. Section 4 contains our results for the information-theoretic setting, and Section 5 contains our result for the computational setting; finally, in Section 6 we present our impossibility results for non-constructive interactive coding.

2 Notation and Preliminaries

2.1 Notation

Basic Notation Let $\mathbb{N}$ denote the set of positive integers, and $[n]$ denote the set $\{1, 2, \dots, n\}$. By a probabilistic algorithm we mean a Turing machine that receives an auxiliary random tape as input. If M is a probabilistic algorithm, then for any input x , the notation “ $M_r(x)$ ” denotes the output of the M on input x when M ’s random tape is fixed to r , while $M(x)$ represents the distribution of outputs of $M_r(x)$ when r is chosen uniformly. We say that a function $\epsilon : \mathbb{N} \rightarrow [0, 1]$ is *negligible* if for every constant $c \in \mathbb{N}$, $\epsilon(n) < n^{-c}$ for sufficiently large k . We say that a function $\mu : \mathbb{N} \rightarrow [0, 1]$ is *overwhelming* if there exists a negligible function ϵ such that for all $n \in \mathbb{N}$, $\mu(n) \geq 1 - \epsilon(n)$.

Probabilistic Notation We use probabilistic notation from [GMR89]: By $x \leftarrow \mathcal{S}$, we denote that an element x is sampled from a distribution $\mathcal{S}$. If F is a finite set, then $x \leftarrow F$ means x is sampled uniformly from the set F . To denote the ordered sequence in which the experiments happen we use comma, e.g. $(x \leftarrow \mathcal{S}, (y, z) \leftarrow A(x))$. Using this notation we can describe probability of events. For example, if $p(\cdot, \cdot)$ denotes a predicate, then $\Pr[x \leftarrow \mathcal{S}, (y, z) \leftarrow A(x) : p(y, z)]$ is the probability that the predicate $p(y, z)$ is true in the ordered sequence of experiments $(x \leftarrow \mathcal{S}, (y, z) \leftarrow A(x))$. The notation $\{(x \leftarrow \mathcal{S}, (y, z) \leftarrow A(x) : (y, z))\}$ denotes the resulting probability distribution $\{(y, z)\}$ generated by the ordered sequence of experiments $(x \leftarrow \mathcal{S}, (y, z) \leftarrow A(x))$.

Notation for Interactive protocols An interactive protocol is a tuple $\pi = (A, B, \mathcal{X}_A, \mathcal{X}_B)$ where A and B are interactive probabilistic Turing machines and $\mathcal{X}_A$ and $\mathcal{X}_B$ specify the set of inputs to A and B (parametrized by a security parameter n). A and B , on input $x \in \mathcal{X}_A(1^n)$ and $y \in \mathcal{X}_B(1^n)$ interact with each other and generate some output at the end of the interaction. We denote this interaction by $A(x) \leftrightarrow B(y)$ (formally, it is a random variable over joint views of A and B , including the randomness of both players, their inputs, and all the messages received). Given an interaction e , we denote by $\text{out}_i[e]$ the output of player $i \in \{1, 2\}$, by $\text{out}[e]$ the output of both parties, and by $\text{trans}[e]$ the transcript of the interaction.

2.2 Statistical Distance and Computational Indistinguishability

We recall the definitions of *statistical distance* and *computational indistinguishability*.

Definition 7 (Statistical distance). The *statistical distance* between two probability distributions X, Y is defined by $\Delta(X, Y) = (1/2) \cdot \sum_x |\Pr[x \leftarrow X] - \Pr[x \leftarrow Y]|$. X and Y are ϵ -close if $\Delta(X, Y) \leq \epsilon$.

The *statistical distance* between two ensembles $\{X_k\}_k$ and $\{Y_k\}_k$ is a function δ defined by $\delta(k) = \Delta(X_k, Y_k)$. Two probability ensembles are said to be *statistically close* if their statistical distance is negligible. We also say X_k and Y_k are statistically close if $\Delta(X_k, Y_k) \leq \epsilon(k)$ for some negligible function ϵ .

Definition 8 (Computational Indistinguishability). Two ensembles $\{X_k\}, \{Y_k\}$ are *computationally indistinguishable* if for every probabilistic polynomial time distinguisher D , there exists a negligible function μ such that for every $k \in \mathbb{N}$,

$$|\Pr[D(1^k, X_k) = 1] - \Pr[D(1^k, Y_k) = 1]| \leq \mu(k).$$

2.3 Hash Functions

We recall the standard definition of t -wise independent hash functions.

Definition 9 (t -wise Independent Hash Functions). A family of hash functions $H = \{h : S_1 \rightarrow S_2\}$ is *t -wise independent* if the following two conditions hold:

1. $\forall x \in S_1$, the random variable $h(x)$ is uniformly distributed over S_2 , where $h \leftarrow H$.
2. $\forall x_1 \neq \dots \neq x_t \in S_1$, the random variables $h(x_1), \dots, h(x_t)$ are independent, where $h \leftarrow H$.

2.4 One-way Functions and Distributionally One-way Functions

We start by recalling the definition of a *one-way function*.

Definition 10 (One-way functions). A function $f : \{0, 1\}^* \rightarrow \{0, 1\}^*$ is *one-way* if it is computable in polynomial time and for every non-uniform probabilistic polynomial time machine A , there exists a negligible function $\mu(\cdot)$ such that for every $n \in \mathbb{N}$,

$$\Pr [x \leftarrow \{0, 1\}^n : A(f(x)) \in f^{-1}(f(x))] < \mu(|x|)$$

Additionally, if there exists some ϵ such that above holds for every $\text{poly}(2^{n^\epsilon})$ -sized circuits A , f is a *subexponentially-hard one-way function*.

A *distributionally one way function* is weaker primitive: here it is only computationally infeasible to find a *random* pre-image to $f(x)$ (but finding *some* pre-image may be easy).

Definition 11 ([IL89]:Distributionally one-way functions). We say a function $f : \{0, 1\}^* \rightarrow \{0, 1\}^*$ is *distributionally one-way* if it is computable in polynomial time and there exists a constant $c > 0$ such that for every non-uniform probabilistic polynomial-time algorithm A , for sufficiently large $n \in \mathbb{N}$, the statistical distance between the following distributions is at least $\frac{1}{n^c}$:

- $\{x \leftarrow \{0, 1\}^n : (f(x), x)\}$
- $\{x \leftarrow \{0, 1\}^n : (f(x), A(f(x_n)))\}$

While distributionally one-way functions are a weaker primitive than one way functions, [IL89] shows that the existence distributionally one-way functions implies the existence of one-way functions.

Theorem 12 ([IL89]). *Distributionally one-way functions exist if and only if one way functions exist.*

2.5 Signature schemes

We recall the definition of an (adaptive-secure) signature schemes.

Definition 13 (Signature scheme [GMR89]). A *secure signature scheme* with *signature-length* $l(\cdot)$ and *key-length* $v(\cdot)$ is a triple $(\text{Gen}, \text{Sig}, \text{Ver})$ of probabilistic polynomial time algorithms, such that

- for all $n \in \mathbb{N}, m \in \{0, 1\}^*$,

$$\Pr[(sk, vk) \leftarrow \text{Gen}(1^n), \sigma \leftarrow \text{Sig}_{sk}(m) : |\sigma| \leq l(n) \wedge |vk| \leq v(n) \wedge \text{Ver}_{vk}(m, \sigma) = 1] = 1$$

- for every non-uniform probabilistic polynomial time adversary A , there exists a negligible function $\mu(\cdot)$ such that

$$\Pr[(sk, vk) \leftarrow \text{Gen}(1^n), (m, \sigma) \leftarrow A^{\text{Sig}_{sk}(\cdot)}(1^n) : \text{Ver}_{vk}(m, \sigma) = 1 \wedge m \notin L] \leq \mu(n)$$

where L denotes the list of A 's queries to its oracle.

The existence of signatures schemes is implied by the existence of one-way functions [NY89, Rom90]:

Theorem 14. *Assume the existence of one-way functions (resp. sub-exponentially hard one-way functions). Then there exists a secure signature scheme (resp. a secure signature scheme with signature-length and key-length $\text{polylog}n$).*

2.6 Error-correcting Codes

We recall the definition of error correcting codes.

Definition 15 (Coding function). An (n, ℓ) -coding function $C = (E, D)$ is an encoding function $E : \{0, 1\}^n \rightarrow \{0, 1\}^\ell$ and a decoding function $D : \{0, 1\}^\ell \rightarrow \{0, 1\}^n$ for some positive integers $\ell \geq n$. The *information rate* of the scheme, denoted R is defined as n/ℓ . The scheme has *error rate* (or *decoding distance*) η if, for all $m \in \{0, 1\}^n$ and all $r \in \{0, 1\}^\ell$ such that the *codeword* $E(m)$ and r differ in at most ηm bits, $D(r) = m$.

Definition 16 (ECC). An family of coding functions $\{C_n = (E_n, D_n)\}_{n \in \mathbb{N}}$ is an *efficient error correcting code* (ECC) $C = (E, D)$ with *error rate* $\eta : \mathbb{N} \rightarrow (0, 1)$ and *information rate* $R : \mathbb{N} \rightarrow (0, 1)$ if for every $n \in \mathbb{N}$, (E_n, D_n) is a $(n, n/R(n))$ coding function with error rate $\eta(n)$, and $\{E_n\}$ and $\{D_n\}$ can be computed by uniform polynomial time algorithms.

As shown by Justesen in [Jus], ECCs with constant error and information rate exist.

Theorem 17 ([Jus]). *There exists an ECC with error rate $O(1)$ and information rate $O(1)$.*

Justesen codes, however, do not give a tight error rate. The concatenation of the Reed-Solomon code [RS60] and the Hadamard code, however, yield an ECC with error rate close to $1/4$ (which is optimal), but require a polynomial information rate.

Theorem 18 ([GS00]). *For every $\epsilon > 0$, there exists an ECC with error rate $\frac{1}{4} - \epsilon$ and information rate $R(n) = O(1/n)$.*

When unique decoding is impossible, it may be still possible to decode to a short list of candidate messages; the notion of list-decoding captures this.

Definition 19. A (n, ℓ) -coding function is (ϵ, L) -list decodable if for any $r \in \{0, 1\}^\ell$, there exists a list of at most $l \leq L$ distinct $m_1, m_2, \dots, m_l$ such that $E(m_i)$ and r differ in at most ϵm bits, for all $i \in [l]$. A family of coding functions $\{(E_n, D_n)\}_{n \in \mathbb{N}}$ with *information rate* $R : \mathbb{N} \rightarrow (0, 1)$ is *efficiently ϵ -list decodable* if for every $n \in \mathbb{N}$, (E_n, D_n) is a $(n, n/R(n))$ coding function that is $(\epsilon(n), L_n)$ -list decodable for some $L_n \in \mathbb{N}$, and there exists a polynomial time algorithm LD that finds this list.

As shown by Guruswami and Sudan [GS00], the concatenation of the Reed-Solomon code and Hadamard code is efficiently list decodable up to an error rate close to $1/2$.

Theorem 20. *For every $\epsilon > 0$, there exists an ECC with information rate $R(n) = O(1/n)$ that is efficiently $\frac{1}{2} - \epsilon$ -list decodable.*

3 Knowledge-Preserving Interactive Coding

In this section we provide a formal definition of knowledge-preserving interactive coding.

3.1 Error Resilience for Interactive Protocols

Let us start by defining *error rate* for interactive protocols; in contrast to earlier works on interactive coding, we here consider error-resilience against both unbounded adversarial channels (as is typically done [Sch96]) and also error-resilience against computationally bounded channels (as was done in the context of error correcting codes in [Lip94, MPSW10]).

Towards providing these definitions, we need some additional notation. The *communication complexity* of a protocol π on security parameter n , denoted by $CC_n(\pi)$, is the worst-case total number of bits transmitted in the interaction $A(x) \leftrightarrow B(y)$, over all possible input $x \in \mathcal{X}_A(1^n), y \in \mathcal{X}_B(1^n)$ and randomness of both players. The round complexity $m(n)$ of a protocol π on security parameter n is the worst-case number of communication rounds (one round corresponds to two message) in the interaction $A(x) \leftrightarrow B(y)$, over all possible input $x \in \mathcal{X}_A(1^n), y \in \mathcal{X}_B(1^n)$ and randomness of both players.

We consider interactive protocols running over noisy/adversarial channels, which may flip some of the bits transmitted by both players. We model channels as interactive Turing machines that relay messages between the two players.

Definition 21 (Channel). A channel is an interactive Turing machine C that on input a security parameter 1^n interacts with two interactive machines by relaying messages for the machines as follows: upon receiving a message m from one machine, C sends a message m' to the other machine of length $|m'| = |m|$.

We denote by $A(x) \leftrightarrow_{C(1^n)} B(y)$ the interaction between $A(x)$ and $B(y)$ over the channel C given the security parameter n . (Note that the interaction $A(x) \leftrightarrow B(y)$ is identical to the interaction $A(x) \leftrightarrow_{C_0} (1^n)B(y)$, where C_0 is the “honest” channel that simply relays messages between A and B without flipping any bits.)

Definition 22 (Communication Complexity over Noisy Channels). Let $\pi = (A, B, \mathcal{X}_A, \mathcal{X}_B)$ be a protocol. The *communication complexity of π over noisy channels* on security parameter n , denoted by $CC_n^*(\pi)$, is the worst-case number of bits transmitted in the interaction $A(x) \leftrightarrow B(y)$, over all possible inputs $x \in \mathcal{X}_A(1^n), y \in \mathcal{X}_B(1^n)$, randomness of both players, and the channel C .

We are now ready to define *error resilience*.

Definition 23. A protocol $\pi = (A, B, \mathcal{X}_A, \mathcal{X}_B)$ with round-complexity $m(\cdot)$ is (*computationally*) $\eta(\cdot, \cdot)$ -*error resilient* if there exists a negligible function μ such that for every security parameter n , inputs $x \in \mathcal{X}_A(1^n), y \in \mathcal{X}_B(1^n)$, and any (non-uniform probabilistic polynomial-time in the security parameter n)³ channel C that flips at most $\eta(n, m(n)) \cdot CC_n^*(\pi)$ bits, the following holds:

$$\Pr [\text{out}[A(x) \leftrightarrow B(y)] = \text{out}[A(x) \leftrightarrow_{C(1^n)} B(y)] \geq 1 - \mu(n).$$

3.2 Knowledge Preservance

Let us move on to defining what it means for a protocol $\tilde{\pi} = (\tilde{A}, \tilde{B}, \mathcal{X}_A, \mathcal{X}_B)$ to be convey “as much knowledge” as a protocol $\pi = (A, B, \mathcal{X}_A, \mathcal{X}_B)$. We formalize this using the classic “simulation-paradigm” from cryptography [GMR89, GMW91]. We require that for every adversarial strategy $\tilde{A}^*$ for player 1 (resp. $\tilde{B}^*$ for player 2) participating in $\tilde{\pi}$, there exists a simulator A^* such that the output of both players in the execution of $(\tilde{A}^*, \tilde{B})$ (resp. $(\tilde{A}, \tilde{B}^*)$) are indistinguishable from the outputs of players in the execution of (A^*, B) (resp. (A, B^*)). That is, any “harm” $\tilde{A}^*$ can do in $\tilde{\pi}$, the simulator A^* could have also done in π .

Definition 24. A protocol $\tilde{\pi} = (\tilde{A}, \tilde{B}, \mathcal{X}_A, \mathcal{X}_B)$ is a (*computationally*) *knowledge-preserving variant* of $\pi = (A, B, \mathcal{X}_A, \mathcal{X}_B)$ if the following two properties hold:

³We could also have defined the channel as a *uniform* polynomial-time algorithm. All our result hold for both choices.

- *Completeness*: There exists a negligible function μ such that the following ensembles are statistically close as a function of n .

$$\begin{aligned} & - \{\text{out}[\tilde{A}(x) \leftrightarrow \tilde{B}(y)]\}_{n \in \mathbb{N}, x \in \mathcal{X}_A(1^n), y \in \mathcal{X}_B(1^n)} \\ & - \{\text{out}[A(x) \leftrightarrow B(y)]\}_{n \in \mathbb{N}, x \in \mathcal{X}_A(1^n), y \in \mathcal{X}_B(1^n)} \end{aligned}$$

- *(Computational) Knowledge Preservance*: For every (probabilistic polynomial-time) adversary strategy $\tilde{A}^*$ for player 1, there exists a (probabilistic polynomial-time) strategy A^* such that the following ensembles are statistically close (resp. computationally indistinguishable) as a function of n .

$$\begin{aligned} & - \{\text{out}[\tilde{A}^*(x, z) \leftrightarrow \tilde{B}(y)]\}_{n \in \mathbb{N}, x \in \mathcal{X}_A(1^n), y \in \mathcal{X}_B(1^n), z \in \{0,1\}^*} \\ & - \{\text{out}[A^*(x, z) \leftrightarrow B(y)]\}_{n \in \mathbb{N}, x \in \mathcal{X}_A(1^n), y \in \mathcal{X}_B(1^n), z \in \{0,1\}^*} \end{aligned}$$

We make the analogous requirement for every (probabilistic polynomial-time) adversary strategy $\tilde{B}^*$ for player 2.

A Remark on Auxiliary Input Just as in the classic definitions of zero-knowledge [GMR89, GO94] and secure computation [GMW91], the additional input z to $\tilde{A}^*$ (and A^*) models any auxiliary information available to the attacker. All our results hold regardless of whether we allow the attacker to receive such auxiliary information.

Single-session v.s. Multiple session Knowledge Preservance Our notion of knowledge preservance assumes that the attacker is only participating in a single execution of π' , and stipulates that this execution of π' emulates π . A stronger notion of *concurrent* knowledge preservance (in analogy with the notion of concurrent zero-knowledge [DNS04]) would instead require that multiple concurrent executions of π' still emulate π (in the sense that an attacker participating in an arbitrary polynomial number of sessions of π' can be emulated by a simulator participating in concurrent sessions of π .) We omit a formal definition of concurrent knowledge preservance, and simply remark that our positive results extend also to the concurrent multi-session setting.

A Remark on Preserving Cryptographic Protocol Security In this comment we assume the reader is familiar with classic definitions of protocol security [GMW91]; see [Gol04] for details. It easily follows from the definition of knowledge preservance that if a protocol π is a “secure implementation” of some functionality $\mathcal{F}$ (in the sense of [Gol04]), then any knowledge-preserving variant π' of π will also be a secure implementation of $\mathcal{F}$. Indeed, if π' is a knowledge-preserving variant of π , then π' is “as secure as” π . (Additionally, if π is a concurrently secure implementation of $\mathcal{F}$ and π' is a concurrent knowledge preserving variant of π , then π' is a concurrently secure implementation of $\mathcal{F}$.)

3.3 Knowledge-Preserving Interactive Coding

We are now ready to define *knowledge-preserving interactive coding*. An *interactive coding scheme* is a pair of oracle-aided interactive probabilistic Turing machines $Q = (Q_1, Q_2)$. For every interactive protocol $\pi = (A, B, \mathcal{X}_A, \mathcal{X}_B)$, Q induces an *encoded interactive protocol* $Q^\pi = (Q_1^A, Q_2^B, \mathcal{X}_A, \mathcal{X}_B)$, defined as follows. In the interaction of Q^π , Q_1 and Q_2 do not receive the input directly. Instead, Q_1 and Q_2 receive as input the security parameter 1^n , the round complexity 1^m and the communication complexity $1^{C_n(\pi)}$ of π , and are given oracle access to $A_{r_A}(x)$ and $B_{r_B}(y)$ respectively, where $x \in \mathcal{X}_A(1^n), y \in \mathcal{X}_B(1^n)$ are the inputs and $r_A, r_B \in \{0, 1\}^\infty$ are uniformly sampled. More precisely,

Q_1 (resp., Q_2) gets oracle access to the next-message functions of $A_{r_A}(x)$ (resp., $B_{r_B}(y)$), which on input a partial transcript T returns the next message (or the final output, in case T is a complete transcript). The interaction is denoted by $Q_1^{A(x)} \leftrightarrow Q_2^{B(y)}$ where the inputs 1^n , 1^m , and $1^{CC_n(\pi)}$ are omitted for notational simplicity. When we are explicit about the randomness used by Q_1 and Q_2 , we write $Q_1^{A(x)}(r_1) \leftrightarrow Q_2^{B(y)}(r_2)$.

Definition 25 (Knowledge-Preserving Interactive Coding Schemes). Let $\eta(\cdot, \cdot), r(\cdot, \cdot) \in (0, 1)$ be functions. A pair of oracle-aided interactive probabilistic Turing machines $Q = (Q_1, Q_2)$ is a (computational) knowledge-preserving interactive coding scheme with error rate $\eta(\cdot, \cdot)$ and information rate $R(\cdot, \cdot)$ if for every interactive protocol $\pi = (A, B, \mathcal{X}_A, \mathcal{X}_B)$, the corresponding encoded protocol Q^π satisfies the following properties.

- *Efficiency*: Q_1 and Q_2 run in polynomial time in $n, m(n)$ and $CC_n(\pi)$.
- *Information Rate*: $CC_n^*(Q^\pi) \leq CC_n(\pi)/R(n, m(n))$; that is the worst-case “blow-up” of the encoded protocol is bounded by $1/R(n, m(n))$.
- *Error Resilience*: Q^π is (computationally) η -error resilient.
- *Knowledge Preservance*: Q^π is a (computationally) knowledge-preserving variant of π .

Q is a (computational) knowledge-preserving interactive coding scheme with information rate $R(\cdot)$ and error rate $\eta(\cdot)$ if Q is (computational) knowledge-preserving interactive coding scheme with information rate $R'(n, m) = R(n)$ and error rate $\eta'(n, m) = \eta(n)$.

4 The Information-Theoretic Regime

As we shall see, achieving knowledge-preserving interactive coding is significantly harder than “plain” interactive coding, and studying *resilience against only computationally bounded adversaries* (as was done in [Lip94, MPSW10] in the context of error correcting codes) actually is *essential* for achieving good error rates in the context of knowledge-preserving interactive coding.

To put our result in context, let us start by showing that the “naive approach” of separately encoding each message in the protocol with a good error correcting code is a knowledge-preserving interactive coding:

Theorem 26. *There exists a knowledge-preserving interactive coding scheme Q with information rate $R(n, m) = O(m)$ and error rate $\eta(n, m) = O(1/m)$.*

Proof. We simply pad each message in the protocol π to become of equal length (this increases the communication complexity by at most a factor m) and next encode each message using a constant-rate error correcting code (see Theorem 17); let $\tilde{\pi}$ denote the encoded protocol. Clearly, $\tilde{\pi}$ is error resilient as long as we corrupt at most one message; thus we have an error rate of $O(1/m)$. It easily follows that $\tilde{\pi}$ is a security preserving variant of π ; the simulator A^* for an attacker $\tilde{A}^*$ for player 1 emulates an execution of $\tilde{\pi}$ for $\tilde{A}^*$ by simply encoding all messages in π (using the error correcting code) and decoding all messages received by $\tilde{A}^*$ before sending them to player 2. The simulator for player 2 is defined analogously. $\square$

Let us now turn to our main impossibility result for the information-theoretic setting. We show that naive approach is essentially optimal: namely, any knowledge preserving interactive coding scheme must have an error rate of at most $1/m$.

Theorem 27. *Let Q be a knowledge-preserving interactive coding scheme with information rate $R(\cdot, \cdot)$ and error rate $\eta(\cdot, \cdot)$. Then for every polynomial $m(\cdot)$, we have that for sufficiently large n , $\eta(n, m(n)) < 1/m(n)$. (In particular, there does not exist a knowledge-preserving interactive coding scheme with error rate $\eta'(n) = 1/\text{poly}(n)$.)*

Proof. Consider some knowledge-preserving interactive coding protocol $Q = (Q_1, Q_2)$ with information rate $R(\cdot, \cdot)$ and error rate $\eta(\cdot, \cdot)$ and let $M(n, \ell, m)$ be a polynomial upper bound on the number queries made by Q_1, Q_2 to its oracles (where n is the security parameter, ℓ is the communication complexity of the protocol π to be encoded and m is the number of round in π). Assume for contradiction that there exists a polynomial $m(\cdot)$ such that for infinitely many n , $\eta(n, m(n)) \geq 1/m(n)$. We will construct a “ping-pong” protocol $\pi = (A, B, \mathcal{X}_A, \mathcal{X}_B)$ with round complexity $m(\cdot)$ such that Q^π cannot be a knowledge-preserving variant of π .

The ping-pong protocol π . Let $t(n) = M(n, m(n), 2m(n)n) + 1$, and $\mathcal{H}_n = \{H_k : \cup_{i \in \{0, \dots, 2m(n)n\}} \{0, 1\}^i \rightarrow \{0, 1\}^n\}$ be a t -wise independent hash function family. On security parameter n , let $\mathcal{X}_A(1^n) = \mathcal{X}_B(1^n) = \mathcal{H}_n$, i.e., the inputs $x \in \mathcal{X}_A(1^n)$ and $y \in \mathcal{X}_B(1^n)$ for both players specify hash functions H_x and H_y in $\mathcal{H}_n$. π is a deterministic protocol that on the inputs $x \in \mathcal{X}_A(1^n)$ and $y \in \mathcal{X}_B(1^n)$ proceeds as follows: First, A sends $a_1 = H_x(\emptyset)$ to B , who sends back $b_1 = H_y(a_1)$ to A . Then at each round i , A sends $a_i = H_x(a_1, b_1, \dots, a_{i-1}, b_{i-1})$ to B , who sends back $b_i = H_y(a_1, b_1, \dots, a_{i-1}, b_{i-1}, a_i)$ to A ; namely, both parties generates their next messages by applying their hash function to the current transcript. At the end of the interaction, both A and B output the whole transcript $(a_1, b_1, \dots, a_m, b_m)$.

Since π is deterministic, the transcript $(a_1, b_1, \dots, a_m, b_m)$ is determined by the inputs x and y . Let $a_i(x, y)$ (and $b_i(x, y)$ resp.) denote the i -th messages A (and B resp.) send in the interaction of $A(x) \leftrightarrow B(y)$, and $\hat{a}_i(x, y) = (a_1, b_1, \dots, a_{i-1}, b_{i-1}, a_i)$ and $\hat{b}_i = (a_1, b_1, \dots, a_i, b_i)$ denote the partial transcripts of the interaction of $A(x) \leftrightarrow B(y)$ up until round i ; by definition, $b_i(x, y) = H_y(\hat{a}_i(x, y))$ for $i \in [m]$ and $a_i(x, y) = H_x(\hat{b}_{i-1}(x, y))$ for $i \in [m]$, where $\hat{b}_0(x, y)$ is defined to be $\emptyset$. Let $m'(\cdot)$ denote the round complexity of the encoded protocol Q^π .

Privacy of the Ping-pong Protocol Our first observation is that in the ping-pong protocol, by the unpredictability of the output of the hash functions, player 1, even if maliciously deviating from the protocol instructions, can only guess more than m distinct pairs $(q, H_y(q))$ with negligible probability. Let the predicate $\text{PrivacyBreach}(o, y) = 1$ iff o contains $m + 1$ distinct pairs $(q, H_y(q))$.

Claim 28. *For every adversarial strategy A^* and every $n \in \mathbb{N}$,*

$$\Pr[x \leftarrow \mathcal{X}_A(1^n), y \leftarrow \mathcal{X}_B(1^n) : \text{PrivacyBreach}(\text{out}_1[A^*(x) \leftrightarrow B(y)], y) = 1] \leq L/2^n,$$

where L denotes the length of A^* 's output (i.e., $L = |\text{out}_1[A^*(x) \leftrightarrow B(y)]|$).

Proof. Note that the interaction with $B(y)$ only allows A^* to make m queries to H_y . Thus, for the predicate PrivacyBreach to output 1, A^* needs to predict one extra pair $(q', H_y(q'))$ that A^* does not learn from B ; that is, q' is different from any partial transcript of the interaction. However, since H_y is t -wise independent, $H_y(q')$ remains uniformly random for every such q . Therefore, each guess of A^* in its output can only be correct with probability 2^{-n} . Since A^* can only make at most L guesses in its output, by an union bound, $\Pr[\text{PrivacyBreach}(\text{out}_1[A^*(x) \leftrightarrow B(y)], y) = 1] \leq L/2^n$. $\square$

We shall now see that in the encoded protocol Q^π , this “privacy-property” no longer holds, and as such Q^π cannot be a knowledge preserving variant of π . As a first step towards showing this, we demonstrate a structural property of the encoded protocol Q^π . In the sequel of the proof, we

assume without loss of generality that Q never makes the same query twice to its oracle (since the oracle anyway is deterministic).

Implicit Ping-pong Computation. We show that (with overwhelming probability) the encoded protocol Q^π “implicitly executes” the original ping-pong protocol in a chronological order. More precisely, as formalized below, the rounds of the encoded protocol can be divided into (non-empty) “chunks”, where each chunk in the encoded protocol corresponds to a single round (i.e., two consecutive messages) in ping-pong protocol; additionally by observing the oracle queries made by Q , we can read out a polynomial list of candidates for the current transcript of the (implicitly executed) ping-pong protocol. We emphasize here that definition of the chunks may depend on the inputs of the players and the randomness of Q .

Formally, let the predicate $\text{ImplicitComp}(x, y, r_1, r_2) = 1$ iff Q_1 asks its oracle (that is, $A(x)$) the queries $\hat{b}_0(x, y)^4, \hat{b}_1(x, y), \hat{b}_2(x, y), \dots, \hat{b}_{m-1}(x, y)$ in order during the interaction of $Q_1^{A(x)}(r_1) \leftrightarrow Q_2^{B(y)}(r_2)$ and all m queries are made in different rounds. When $\text{ImplicitComp}(x, y, r_1, r_2) = 1$, Q_1 ’s queries partition the rounds of the interaction into non-empty chunks in the natural way: for every $i \in [m]$, the i -th chunk of the interaction $Q_1^{A(x)}(r_1) \leftrightarrow Q_2^{B(y)}(r_2)$ starts at the round where Q_1 makes the query $\hat{b}_{i-1}(x, y)$ and finishes when makes Q_1 the query $\hat{b}_i(x, y)$.⁵ The following lemma shows that with overwhelming probability (over random inputs x, y and the execution of $Q_1^{A(x)} \leftrightarrow Q_2^{B(y)}$) ImplicitComp holds and thus chunks are well-defined.

Lemma 29 (Implicit Computation Lemma). *There exists a negligible function $\mu(\cdot)$ such that for every $n \in \mathbb{N}$,*

$$\Pr[x \leftarrow \mathcal{X}_A(1^n), y \leftarrow \mathcal{X}_B(1^n), r_1, r_2 \in \{0, 1\}^\infty : \text{ImplicitComp}(x, y, r_1, r_2) = 1] \geq 1 - \mu(n)$$

Intuitively, the lemma follows by the “elusiveness” property of the output of the hash function H_x used by A : if Q_1 is not asking its oracle all the queries in order it must be able to guess the output of H_x on a new point q , which contradicts its t -wise independent property. Note that we here rely on the fact that the output of the hash functions are “long” (i.e., super-logarithmic); otherwise, it is easy to guess the output of the hash function with inverse polynomial probability. We proceed to a formal proof.

Proof (of Lemma 29). For convenience, we actually prove a slightly stronger statement: We show that with overwhelming probability, the following three properties holds during the execution of $Q_1^{A(x)}(r_1) \leftrightarrow Q_2^{B(y)}(r_2)$: (a) Q_1 makes the queries $\hat{b}_0(x, y), \hat{b}_1(x, y), \hat{b}_2(x, y), \dots, \hat{b}_{m-1}(x, y)$ to $A(x)$, (b) Q_2 makes the queries $\hat{a}_1(x, y), \hat{a}_2(x, y), \dots, \hat{a}_m(x, y)$ to $B(y)$, and (c) the queries are made in chronological order; that is, every $i \in [m]$, Q_1 make the query $\hat{b}_{i-1}(x, y)$ before Q_2 makes the query $\hat{a}_i(x, y)$ and Q_2 make the query $\hat{a}_i(x, y)$ before Q_1 makes the query $\hat{b}_i(x, y)$. It easily follows that if the above three properties hold with respect to (x, y, r_1, r_2) then $\text{ImplicitComp}(x, y, r_1, r_2) = 1$. We now show that except with negligible probability (over x, y, r_1, r_2), each of these properties (individually) hold; we can then conclude by a union bound that with overwhelming probability, all of them simultaneously hold.

For (a), suppose for the sake of contradiction that with some noticeable probability $\epsilon(n)$, Q_1 does *not* make the query $\hat{b}_i(x, y)$ for some $i \in \{0, \dots, m-1\}$. By completeness of Q^π , Q_1 outputs $\hat{b}_m(x, y)$ with overwhelming probability. Thus, by an union bound, with noticeable probability

⁴Recall that $\hat{b}_0(x, y)$ is defined to be $\emptyset$.

⁵We can assume without loss of generality that Q_1 always queries the full transcript $\hat{b}_m(x, y)$ before generating the output (at the cost of at most one extra query), so that the last chunk m also is well defined.

$\epsilon'(n)$, Q_1 does not query $\hat{b}_i(x, y)$ but outputs $\hat{b}_m(x, y)$, which contains $a_i(x, y) = H_x(\hat{b}_i(x, y))$. But, this can only happen with probability at most 2^{-n} since H_x is t -wise independent and Q_1 makes at most $t - 1$ queries to its oracle, none of which is $\hat{b}_i(x, y)$, and its oracle is exactly computing $H_x(\cdot)$; thus, even conditioned on Q_1 view of the interaction, $H_x(\hat{b}_i(x, y))$ is uniform and can thus only be guessed by Q_1 with probability 2^{-n} . By identically the same argument it follows that (b) happens except with negligible probability.

For (c), suppose for the sake of contradiction that with some noticeable probability $\epsilon(n)$, Q_2 make the query $\hat{a}_i(x, y)$ before Q_1 makes the query $\hat{b}_{i-1}(x, y)$ for some $i \in [m]$. Note that $\hat{a}_i(x, y)$ contains $a_i(x, y) = H_x(\hat{b}_{i-1}(x, y))$. Thus, by guessing which query of Q_2 is $\hat{b}_{i-1}(x, y)$, we can construct an algorithm R that predicts the value of $H_x(\hat{b}_{i-1}(x, y))$ with probability $\epsilon(n)/t(n)$, while querying H_x on at most $t(n) - 1$ points, none of which is $\hat{b}_{i-1}(x, y)$, contradicting t -wise independence of H_x . By identically the same argument it follows that Q_1 only make the query $\hat{b}_i(x, y)$ before Q_2 makes the query $\hat{a}_i(x, y)$ for some $i \in [m]$ with negligible probability. $\square$

Obtaining a Privacy Breach in the Encoded Protocol We are now ready show how to obtain a privacy breach in the encoded protocol Q^π .

Claim 30. *There exists an adversarial strategy $\tilde{A}^*$ for player 1 in Q^π such that for sufficiently large $n \in \mathbb{N}$, the output length of $\tilde{A}^*$ is bounded by $2M(n, m, 2mn)m(n)n$ and*

$$\Pr[x \leftarrow \mathcal{X}_A(1^n), y \leftarrow \mathcal{X}_B(1^n) : \text{PrivacyBreach}(\text{out}_1[\tilde{A}^*(x) \leftrightarrow Q_2^{B(y)}], y) = 1] \geq 1/4m'(n)$$

Proof. The idea behind the attacker $\tilde{A}^*$ (acting as player 1) is the following. By an averaging argument, one of the chunks, say chunk i , in the encoded protocol must be shorter than a fraction $1/m$ of the total communication complexity of the encoded protocol. The idea now is for $\tilde{A}^*$ to honestly execute the encoded protocol using its actual input x and randomness r_1 , except that during the i 'th chunk, $\tilde{A}^*$ samples a fresh input x' (and a fresh randomness r'_1 for Q_1) consistent with the transcript up until (and including) chunk $i - 1$,⁶ and executes the chunk using the input x' (and randomness r'_1) instead of x . Intuitively, since the chunk was “small”, by the error resilience property of the interactive coding scheme, player 1 will finally produce the same output (including m pairs $(q, H_y(q))$) as if it had been running the protocol honestly (that is, without “deviating” in chunk i). Formally proving this requires showing that the attack performed by $\tilde{A}^*$ can be modelled as channel that flips a sufficiently small number of bits; indeed, note that the way $\tilde{A}^*$ deviates from the protocol does not rely the original random coins r_1 used by Q_1 or the input x , and this property is crucial for us to be able to emulate the attacker by a channel.

Finally, by observing the oracle queries made by Q_1 during the i 'th chunk—which corresponds to the i 'th round in the ping-pong protocol, by the “implicit computation” property— $\tilde{A}^*$ may learn an additional pair $(q', H_y(q'))$; intuitively, the reason for this is that, with overwhelming probability, the messages in the i -th round in “implicit computation” of π remain uniformly distributed, even after conditioning on the first $i - 1$ rounds. (We, however, warn the reader that proving this is quite subtle; see Sub-claim 32). Thus, if $\tilde{A}^*$ could just identify the i 'th chunk, it can learn $m + 1$ pairs $(q, H_y(q))$ of H_y . Towards this, $\tilde{A}^*$ simply guesses the starting round of the i 'th chunk.

We proceed to a formal description of the attacker $\tilde{A}^*$. (Recall that $m'(\cdot)$ denotes the round complexity of Q^π .) On input $x \in \mathcal{X}_A(1^n)$, and randomness r_1 , $\tilde{A}^*$ performs the follow “forking” attack:

⁶Note that this step may not be efficiently computable in general, but this is not a problem since we here consider information-theoretic security.

1. $\tilde{A}^*$ uniformly picks a random round $j \leftarrow [m'(n)]$ and honestly executes the encoded protocol Q_1 up to the end of $(j-1)$ -th round. Let T be the resulting partial transcript.
2. $\tilde{A}^*$ samples a fresh input-randomness pair (x', r'_1) conditioned on the partial transcript T ; namely, (x', r'_1) are uniformly random over all input-randomness pair such that $Q_1^{A(x')}(r'_1)$ is consistent with T .⁷ Then, $\tilde{A}^*$ continues executing Q_1 but now with inputs x' and randomness r'_1 , for as many rounds as possible, subject to the restriction that the number of bits it transmitted since round j does not exceed $\eta(n, m) \cdot \text{CC}_n(Q^\pi)$.
3. $\tilde{A}^*$ continues the rest of the interaction honestly, with the “true” input x and randomness r_1 of Q_1 pretending that its own messages were honestly sent all along (including the messages sent since round j), but may have been incorrectly received by player 2. At the end of the interaction, $\tilde{A}^*$ outputs (o, L) , where o is the output of Q_1 , and L is the list of queries Q_1 made during the “deviation” (using the new input x').

We first show that, with overwhelming probability, $\tilde{A}^*$ can still learn the “valid” m pairs $(q, H_y(q))$ (that it would have learn even if it didn’t deviate).

Sub-claim 31. *There exist a negligible function $\mu(\cdot)$ such that for every $n \in \mathbb{N}$,*

$$\Pr[x \leftarrow \mathcal{X}_A(1^n), y \leftarrow \mathcal{X}_B(1^n), (o, L) \leftarrow \text{out}_1[\tilde{A}^*(x) \leftrightarrow Q_2^{B(y)}] : o = \text{out}_1[A(x) \leftrightarrow B(y)] \geq 1 - \mu(n).$$

Proof. Note that the deviation performed by $\tilde{A}^*$ in Step 2 can be implemented by a channel C , since it only relies on knowledge of the transcript of the interaction and not the internal state (inputs and randomness) of either of the players. Also, by construction of $\tilde{A}^*$, C never needs to flip more than $\eta \cdot \text{CC}_n(Q^\pi)$ bits. The claim follows directly by the η -error resilience and completeness of Q^π . $\square$

Note that $o = \text{out}_1[A(x) \leftrightarrow B(y)]$ contains m distinct pairs $(q, H_y(q))$, namely, $(\hat{a}_i(x, y), b_i(x, y) = H_y(\hat{a}_i(x, y)))$ for $i \in [m]$; below we abuse of notation and let o denote the set of these pairs. We next show that L contains one additional distinct pair $(q', H_y(q'))$ with noticeable probability. Recall that a query of Q_1 is of the form $(a_1, b_1, \dots, a_i, b_i)$; below we sometimes abuse of notation and interpret each such query as a pair (q, v) where $q = (a_1, b_1, \dots, a_i)$ and $v = b_i$ (that is, q is the vector containing all but the last component, and v contains only the last component).

Sub-claim 32. *For sufficiently large $n \in \mathbb{N}$, the following holds*

$$\Pr \left[\begin{array}{l} x \leftarrow \mathcal{X}_A(1^n), y \leftarrow \mathcal{X}_B(1^n), (o, L) \leftarrow \text{out}_1[\tilde{A}^*(x) \leftrightarrow Q_2^{B(y)}] : \\ \exists (q, H_y(q)) \in L \text{ and } (q, H_y(q)) \notin \text{out}_1[A(x) \leftrightarrow B(y)] \end{array} \right] \geq 1/2m'(n).$$

Proof. Note that in the experiment $\{x \leftarrow \mathcal{X}_A(1^n), y \leftarrow \mathcal{X}_B(1^n) : \text{out}_1[\tilde{A}^*(x) \leftrightarrow Q_2^{B(y)}]\}$, for every choice of j (by $\tilde{A}^*$), the tuple (x', y, r'_1, r_2) is uniformly distributed (since we can view the experiment as first sampling a uniform $j-1$ -round transcript T and then then sampling (x', y, r'_1, r_2) conditioned on T). It follows that the distribution of (x', y, r'_1, r_2) is independent of j .

Additionally, note that $\tilde{A}^*$ could have picked x', r'_1 in a somewhat more convoluted way, generating exactly the same distribution: instead of directly sampling x', r'_1 conditioned on T , first sample a list L_1 of (at most $t(n) - 1$) query-answer pairs corresponding to Q_1 ’s queries to its oracle $A(x)$ up until round $j - 1$, conditioned on the transcript being T ; next, sample x' conditioned on L_1 and T , and finally r' conditioned on x', L_1 and T . The following observation will be useful in what follows:

⁷Note that this is the only inefficient step in the forking attack.

Sampling x' conditioned on L_1 and T is equivalent to sampling x' just conditioned on just the query-answer pairs L_1 .

The reason this holds is that conditioned on L_1 , x' and T are independent (recall that T is determined as a function of just L_1, y, r_1, r_2).

Now, by observing the list of queries L_1 (up to the transcript T) and the input y of player 2, let us determine the next round in the “implicit computation” of π (or said otherwise, the “next chunk” in the encoded protocol after the transcript T). If L_1 does not contain the query $\emptyset$, let $i = 1$ (i.e., the implicit computation has not begun yet since player 1 has not generated its first input a_1 ; consequently, the next round is 1). If L_1 contains the query-answer pair $(\emptyset, a_1)$ (since Q_1 ’s oracle A is deterministic, there can be at most one such query), check if L_1 contains a query of the form $(b_1 = H_y(a_1), a_2)$ (again, there can be at most one such query); if not set $i = 2$, otherwise, check if L_1 contains a query of the form $(b_2 = H_y(a_1, b_1, a_2), a_3)$, and so on.

Below we show the following two statements:

1. With probability at least $1/m'(n) - \text{negl}(n)$, it holds that $i \in [m(n)]$ (i.e., the implicit computation of π has not ended) and L contains the query $\hat{b}_i(x', y)$ (which corresponds to the pair $(\hat{a}_i(x', y), b_i(x', y) = H_y(\hat{a}_i(x', y)))$)
2. With probability at most 2^{-n} , it holds that $(\hat{a}_i(x', y), b_i(x', y)) \in \text{out}_1[A(x) \leftrightarrow B(y)]$.⁸

The claim then follows by a union bound.

To prove statement (1), consider some fixed (x', y, r'_1, r_2) where chunks are well-defined (i.e. $\text{ImplicitComp}(x', y, r'_1, r_2) = 1$). Note that the event in (1) means that the whole i -th chunk of $Q_1^{A(x')}(r'_1) \leftrightarrow Q_2^{B(y)}(r_2)$ is completed during the deviation, i.e., the chunk starts at or after round j , and ends before the “cut-off”. By an averaging argument, there must exist some chunk i^* that is shorter than $(1/m) \cdot \text{CC}_n(Q^\pi) \leq \eta(n, m) \cdot \text{CC}_n(Q^\pi)$. Clearly, when j equals to the starting round of chunk i^* , which happens with probability $1/m'(n)$ since j is uniformly distributed and independent of (x', y, r'_1, r_2) , chunk $i = i^*$ and thus chunk i will be completed before the cut-off; consequently, since $\text{ImplicitComp}(x', y, r'_1, r_2) = 1$, L contains the query $\hat{b}_i(x', y)$. Since, by Lemma 29 (and the observation that x', y, r'_1, r_2 are uniformly distributed) $\text{ImplicitComp}(x', y, r'_1, r_2) = 1$ holds with overwhelming probability, we have $\Pr[\hat{b}_i(x', y) \in L] \geq 1/m'(n) - \text{negl}(n)$ for some negligible function $\text{negl}(n)$.

To prove statement (2), note that if $a_i(x, y) \neq a_i(x', y)$, then $(\hat{a}_i(x', y), b_i(x', y)) \notin \text{out}_1[A(x) \leftrightarrow B(y)]$. Thus, it is sufficient to show that $\Pr[a_i(x, y) = a_i(x', y)] \leq 2^{-n}$; that is $\Pr[H_x(\hat{b}_{i-1}(x, y)) = H_{x'}(\hat{b}_{i-1}(x', y))] \leq 2^{-n}$. Consider some fixed x, y, r_1, r_2, L_1 ; note that i is determined as a function of these (recall that it is a function of L_1, y), and trivially $H_x(\hat{b}_{i-1}(x, y))$ is determined. Additionally, note that for every x' consistent with L_1 , $\hat{b}_{i-1}(x', y) = H_y(\hat{a}_{i-1}(x', y))$ is determined and furthermore $\hat{b}_{i-1}(x', y)$ is not contained in L_1 (since by definition of i , $\hat{b}_{i-2}(x', y)$ is the “last” ping-pong query in L_1). Since, as observed above, x' is uniformly sampled conditioned only on L_1 (and since L_1 contains at most $t(n) - 1$ queries, none of which is $\hat{b}_{i-1}(x', y)$), it follows that *every* fixed x, y, r_1, r_2, L_1 , $\Pr[H_x(\hat{b}_{i-1}(x, y)) = H_{x'}(\hat{b}_{i-1}(x', y))] \leq 2^{-n}$, and thus this condition also holds for random x, y, r_1, r_2 . □

By combining Sub-claim 31 and 32, and applying the union bound we get that

$$\Pr[\text{PrivacyBreach}(\text{out}_1[\tilde{A}^*(x) \leftrightarrow Q_2^{B(y)}], y) = 1] \geq 1/2m'(n).$$

which concludes the proof of Claim 30. □

⁸Pedantically, in case $i > m(n)$, $\hat{a}_i$ is not defined; in this case the event is simply defined to not hold.

Combining Claim 28 and 30 shows that Q^π is not a knowledge-preserving variant of π , which leads to a contradiction and completes the proof of Theorem 27. $\square$

5 The Computational Regime

We here turn to studying knowledge-preserving interactive coding in the presence of only computationally-bounded adversaries (i.e., computational knowledge-preserving interactive coding).

5.1 Positive Results

We first present a positive result, showing that assuming the existence of one-way function, computational knowledge-preserving interactive coding with constant error rate (more specifically, close to $1/12$) and sub-polynomial (or even poly logarithmic, if assuming subexponentially-hard one-way functions) information rate is possible.

Theorem 33. *Assume the existence of one-way functions. For every $\epsilon > 0$, there exists a computational knowledge-preserving interactive coding scheme with perfect completeness, error rate $\eta = \frac{1}{12} - \epsilon$ and information rate $R(n) = O(1/n^\epsilon)$. If additionally sub-exponentially hard one-way functions exists, the information rate is $1/\text{polylog}n$.*

Roughly speaking, the idea behind the scheme is the following. In a “preamble phase”, the players start by exchanging verification keys for a signature scheme; the verification keys first are padded with ρ 0’s to become “long” enough (where ρ is a parameter to be set) and then encoded using a good ECC (E_p, D_p) ; let $\alpha(n)$ denote the length of the encoded verification key. Next, in the “main execution phase” we run the original protocol π , except that each messages is signed and encoded using a good error correcting code (E_m, D_m) (which may be different from the code (E_p, D_p)). More precisely, player 1 keep track of the “current round” number i_1 in the protocol π , and encode its i^{th} -round message a_i as $c_i = E_m(i, a_i, \sigma_i)$ where σ_i is a signature of (i, a_i) . Upon receiving a message c while having the “current round” number (in the protocol π) being i , player 1 decodes the message $((i', b), \sigma) = D_m(c)$ and interprets b as the i' ’th round message b_i in π , as long as 1) $i' = i - 1$, and 2) σ is a valid signature on (i', b) ; if not, player 1 “signals” that the received message was corrupted by simply resending its message $c_{i-1} = E_m(i - 1, a_{i-1}, \sigma_i)$ from the previous round. Player 2’s strategy is defined analogously except that player 2 accepts the received message if $i' = i$ (since player 2 is sending the second message in each round). Finally, we impose a bound c on the communication complexity of the protocol (or else the protocol may run forever, due to “resend” message); both players simply abort outputting nothing if the communication complexity would exceed c if they send their message. Let $\beta(n)$ denote the length of each encoded message, and let $\gamma(n, m) = 2\alpha(n) + 2m\beta(n)$ be the length of the protocol if all messages get sent through on the first trial, and there is no cut-off.

We must set ρ such that the length $\alpha(n)$ of the encoded verification keys is within a constant factor of c (or else, either the preamble phase can be fully corrupted, or the main phase can be fully corrupted). On the other hand, c must be long enough to execute the encoded version of π (that is $c > \gamma(n, m)$), and additionally handle sufficiently many “resend” requests, before the error-quota of the adversary runs out. By appropriately setting ρ and c , this leads to an error rate of $\eta/4$ if η is the error rate of both (E_p, D_p) and (E_m, D_m) ; roughly speaking, we lose a factor of two because the adversary may choose to corrupt either the verification key of player 1 or that of player 2; we loose another (additively) factor of two due to the fact that the length of the encoded messages must be within a constant factor of c , and the fact that each time the attacker corrupts the message of

a single player in the main phase protocol execution, we need to resend the whole round (i.e., 2 messages).

We can further improve the error rate by relying on an idea from [MPSW10]: since the messages in the main phase are signed and we only consider a computationally bounded channel, it in fact suffices to list-decode the error-correcting code (E_m, D_m) used in the main phase.⁹ It follows using the same argument as in [MPSW10] that (with overwhelming probability) list-decoding can only yields at most a single valid message (since all the messages are signed), and thus using list-decoding here yields unique decoding.

We provide a formal description of the scheme in Figure 1. We assume without loss of generality that in the original protocol π each player sends a *single bit* at each round in the protocol (we refer to such protocols as “bit protocols”).

The following key lemma shows that the above-described scheme is a knowledge preserving interactive coding schemes. The proof of Theorem 33 is then concluded by appropriately setting ρ and c .

Lemma 34. *Let $\pi = (A, B, \mathcal{X}_A, \mathcal{X}_B)$ be a bit protocol with round complexity $m(\cdot)$. Let $(\text{Gen}, \text{Sig}, \text{Ver})$ be a secure signature scheme with signature length $l(n)$ and verification-key length $v(n)$, let (E_p, D_p) be an ECC with constant error rate η_p and information rate $R_p(\cdot)$ and (E_m, D_m) be an error correcting codes that is efficiently η_m -list-decodable and has information rates $R_p(\cdot)$. Consider Q, α, β, γ defined in Figure 1 w.r.t the functions $\rho(\cdot, \cdot), c(\cdot, \cdot)$. If $c(n, m) \geq \gamma(n, m)$, then Q is a computational knowledge preserving interactive coding scheme with perfect completeness, and:*

- information rate $R(\cdot)$ s.t.

$$R(n, m) = c(n, m)/2m;$$

- error rate $\eta(\cdot, \cdot)$ s.t.

$$\eta(n, m) = \min \left(\eta_m \cdot \frac{c(n, m) - \gamma(n, m)}{2c(n, m)}, \eta_p \cdot \frac{\alpha(n)}{c(n, m)} \right).$$

Proof. We separately prove each of the required properties.

Efficiency and Information rate: Q is clearly polynomial-time computable, and by definition of Q , we have that $CC^*(Q^\pi) = c(n, m)$; it follows that Q has information rate $R(n, m) = c(n, m)/(2m)$.

Perfect Completeness: It easily follow from the definition of Q that Q^π perfectly emulates π if both players are honest. In fact, for every n , input pair $x \in \mathcal{X}_A(1^n), y \in \text{cal}X_B(1^n)$ and randomness pair $r_A, r_B \in \{0, 1\}^\infty$,

$$\Pr \left[\text{out}[Q_1^{A_{r_A}(x)} \leftrightarrow Q_2^{B_{r_B}(y)}] = \text{out}[A_{r_A}(x) \leftrightarrow B_{r_B}(y)] \right] = 1$$

We refer to this property as *perfect completeness*, and it will be useful shortly.

⁹[MPSW10] relies on this idea to show how to acheive an error-correcting code with error rate $1/2 - \epsilon$ if assuming a (noiseless) public-key infrastructure and a computationally-bounded channel. In our context, we do not have a public-key infrastructure, but our initial exchange of verification keys using a uniquely decodable error-correcting code can be viewed as a way to set-up the appropriate public-key infrastructure needed for their results.

Input protocol: A bit protocol $\pi = (A, B, \mathcal{X}_A, \mathcal{X}_B)$ with round complexity $m = m(n)$.

Parameters:

- Let $\rho = \rho(n, m)$ be a *padding* parameter.
- Let $c = c(n, m)$ be a *cut-off* parameter.

Primitives used:

- Let $(\text{Gen}, \text{Sig}, \text{Ver})$ be a signature scheme with signature length $l = l(n)$ and verification key length $v = v(n)$.
- Let (E_p, D_p) be an error correcting code with (constant) error rate η_p and information rate $R_p = R_p(\cdot)$.
- Let (E_m, D_m) be an error correcting code that is efficiently η_m -list decodable and has information rate $R_m = R_m(\cdot)$.

Initialization: On input a security parameter 1^n , round complexity 1^m , and communication complexity 1^{2m} , Q_1 (resp., Q_2) initializes a counter $i_1 = 1$ (resp., $i_2 = 1$).

Preamble: Q_1 runs $(sk_1, vk_1) \leftarrow \text{Gen}(1^n)$ and sends $c_{1,0} = E_p(vk_1 || 0^\rho)$ to Q_2 . Then Q_2 runs $(sk_2, vk_2) \leftarrow \text{Gen}(1^n)$ and sends $c_{2,0} = E_p(vk_2 || 0^\rho)$ to Q_1 . Both Q_1 and Q_2 decode the received message $c'_{2,0}$ and $c'_{1,0}$ and store $vk'_2 = D_p(c'_{2,0})$ and $vk'_1 = D_p(c'_{1,0})$, respectively. Let $\alpha(n)$ denote the length of each message $c'_{1,0}, c'_{2,0}$ in the preamble phase; that is $\alpha(n) = (v(n) + \rho(n))/R_p(v(n) + \rho(n))$.

Main: We first define the strategy of Q_1 :

- First, Q_1 queries its oracle A to obtain the first message a_1 , appends a_1 to t_1 , computes $\sigma_1 \leftarrow \text{Sig}_{sk_1}((i_1, a_1))$, sends $c_{1,1} = E_m((i_1, a_1, \sigma_1))$ to Q_2 , and increases the counter by 1 (i.e., $i_1 := i_1 + 1$).
- Upon receiving a message c' from Q_2 , Q_1 list-decodes c , and verifies that there exists a *unique* message (i', b', σ') such that (a) $i' = i_1 - 1$ and (b) $\text{Ver}_{vk'_1}((i', b'), \sigma') = 1$. If the verification rejects, Q_1 resends its previous message c_{1,i_1-1} . If the verification accepts, then Q_1 appends b' to t_1 and queries t_1 to its oracle A . If A returns an output o_1 , then Q_1 outputs o_1 and terminates. If A returns a next message a_{i_1} , then Q_1 appends a_{i_1} to t_1 , computes $\sigma_{i_1} \leftarrow \text{Sig}_{sk_1}((i_1, a_{i_1}))$, sends $c_{1,i_1} = E_m((i_1, a_{i_1}, \sigma_{i_1}))$ to Q_2 , and increases the counter by 1 (i.e., $i_1 := i_1 + 1$).

The strategy of Q_2 is defined analogously, except that in step (a), Q_2 verifies that $i' = i_2$ (as opposed to $i_2 - 1$).

- Upon receiving a message c' from Q_1 , Q_2 list-decodes c , and verifies that there exists a *unique* message (i', b', σ') such that (a) $i' = i_2$ and (b) $\text{Ver}_{vk'_1}((i', a'), \sigma') = 1$. If the verification rejects, Q_2 resends its previous message c_{2,i_2-1} . If the verification accepts, then Q_2 appends a' to t_2 and queries t_2 to its oracle B . If B returns an output o_2 , then Q_2 outputs o_2 and terminates. If B returns a next message b_{i_2} , then Q_2 appends b_{i_2} to t_2 , computes $\sigma_{i_2} \leftarrow \text{Sig}_{sk_2}((i_2, a_{i_2}))$, sends $c_{2,i_2} = E_m((i_2, b_{i_2}, \sigma_{i_2}))$ to Q_1 , and increases the counter by 1 (i.e., $i_2 := i_2 + 1$).

Let $\beta(n)$ denote the length of each round in the main phase. Let $\gamma(n, m) = 2\alpha(n) + 2m\beta(n)$.

Abort condition: At any point, if sending the next message causes the total communication to exceed $c(n, m)$ bits, the scheme Q aborts (with the players outputting $\perp$).

Figure 1: Interactive coding scheme $Q = (Q_1, Q_2)$ encoding an interactive bit protocol π .

Knowledge Preservance: First note that if $c(n, m) \geq \gamma(n, m)$ then Q is complete. Note that Q executes π in a straight-line fashion (without any “rewindings”). We now use this observation to show that $\tilde{\pi} = Q^\pi$ is a knowledge preserving variant of π . More precisely, for every polynomial-time attacker $\tilde{A}^*$ for player 1, we show the existence of a polynomial-time simulator A^* such that for every $x \in \mathcal{X}_A$, $y \in \mathcal{X}_B$ and $z \in \{0, 1\}^*$ we have that $\text{out}_{\tilde{A}^*}[\tilde{A}^*(x, z) \leftrightarrow Q_2^{B(y)}]$ is identically distributed to $\text{out}_{A^*}[A^*(x, z) \leftrightarrow B(y)]$. The simulator $A^*(x, z)$, in essence, is just the encoding algorithm Q_2 : $A^*(x, z)$ simply emulates an interaction between $\tilde{A}^*(x, z)$ and Q_2 , while externally forwarding all the oracle queries by Q_2 and answering those queries by forwarding back all the external message. Since Q_2 never rewinds its oracle, the view of $\tilde{A}^*$ in the simulation is identical to its view in the real execution. (Note that the knowledge preservance property holds unconditionally.) A simulator for an adversarial player 2 is constructed in the analogous way.

Error resilience We turn to showing that Q^π is η -error-resilient, where

$$\eta(n) = \min \left(\eta_m \cdot \frac{c(n, m(n)) - \gamma(n, m(m))}{2c(n, m(m))}, \eta_p \cdot \frac{\alpha(n)}{c(n, m(m))} \right)$$

Consider some non-uniform polynomial-time channel C , security parameter n , inputs x, y and randomness r_A, r_B and an execution $e \in \text{supp}(Q_1^{A_{r_A}(x)} \leftrightarrow_{C(1^n)} Q_2^{B_{r_B}(y)})$ such that $\text{out}(e) \neq \text{out}(A_{r_A}(x) \leftrightarrow B_{r_B}(y))$ (recall that by perfect completeness, for every $e' \in \text{supp}(Q_1^{A_{r_A}(x)} \leftrightarrow Q_2^{B_{r_B}(y)})$, we have that $\text{out}(e') = \text{out}(A_{r_A}(x) \leftrightarrow B_{r_B}(y))$.) For this to happen, either of two things must have happened:

1. either execution gets cut-off (in which case both players output $\perp$); or,
2. either Q_1 or Q_2 queries its oracle on a partial transcript t' that is not a partial transcript in the execution of $A(x) \leftrightarrow B(y)$.

We show that neither of these cases can happen with inverse polynomial probability for infinitely many n (and selections of x, y, r_A, r_B) as long as C corrupts at most $\eta(n)c(n, m(n))$ bits.

Let us first prove that case 1 only can happen with negligible probability. First, note that since $c(n, m(n)) \geq \gamma(n, m(n))$ we have that Q^π does not abort when run over a noiseless channel. Next, note that since $\eta(n) \leq \eta_p \alpha(n) / c(n, m(n))$, the channel can corrupt at most $\eta_p \cdot \alpha(n)$ bits. It follows that each message in the preamble phase will always be correctly decoded, since (E_p, D_p) has error rate η_p and each message in the preamble phase is of length $\alpha(n)$.

Additionally, since $\eta(n) \leq \eta_m \cdot (c(n, m(n)) - \gamma(n, m(n))) / (2c(n, m(n)))$, the channel can corrupt at most

$$\eta_m \cdot \frac{(c(n, m(n)) - \gamma(n, m(n)))}{2}$$

bits. Note that unless channel corrupts $\eta_m \beta(n)$ bits in a message in the main phase, the message can still be list decoded. Furthermore, it follows (as in [MPSW10]), relying on the security of the signature scheme (against non-uniform polynomial-time attackers)¹⁰ that except with negligible probability, the correct message is the unique one that has an accepting signature (since the channel has seen at most a single signed message of the form $(i, \cdot)$ for each verification key). On the other hand, if the channel corrupts more than $\eta_m \beta(n)$ bits in one message, list decoding is no longer guaranteed to work, and this may cause the players to resend *two* messages (recall that the player

¹⁰Note that the reason we require non-uniform security of the signature scheme is that the attacker needs to get the inputs x, y and the non-uniform advice of C .

that notices a corrupted message, resends its previously send message, which forces the other player to resend the corrupted one). Thus, every time the channel corrupts $\eta_m \beta(n)$ bits, it may cause the interaction to become $2\beta(n)$ bits longer. So, to make the interaction become cut-off (with non-negligible probability), we need j such corruptions of $\eta_m \beta(n)$ bits, where

$$\gamma(n) + 2\beta(n)j > c(n, m(n)).$$

In other words,

$$j > \frac{c(n, m(n)) - \gamma(n, m(n))}{2\beta(n)},$$

which means that the channels needs to corrupt more than

$$\eta_m \cdot \frac{c(n, m) - \gamma(n, m)}{2}$$

bits, which is a contradiction.

We proceed to show that case 2 only can happen with negligible probability. The key observation needed to prove this is that, as mentioned above, the preamble phase is always uniquely decoded and thus C cannot change the verification keys vk_1, vk_2 . Consider the first time that, say, Q_1 queries its oracle on a partial trascript t' that is not a partial transcript in the execution of $A(x) \leftrightarrow B(y)$. It means that Q_1 accepts an incorrect message (i', b', σ') such that b' is different from the message b_{i_1-1} it should have received in π in round $i_1 - 1$, it must be the case that a) $i' = i_1 - 1$ (or else Q_1 would reject it), and b) C provided a valid signature (for the verification key vk_2) on (i', b') . But the channel has seen at most a single signature (for the verification key vk_2) on a message of the type $(i_1 - 1, \cdot)$ (since Q_2 will only send a single message of this type); it thus follows from the non-uniform security of the signature scheme that player 2 accepts an incorrect message with at most negligible probability. It follows analogously that C can only make player 2 accept an incorrect message with negligible probability. $\square$

Equipped with Lemma 34, we now turn to proving Theorem 33.

Proof of Theorem 33. Assume the existence of one-way functions, fix some $\epsilon > 0$. By scaling down the security parameter in the construction from Theorem 14, there exists a signature scheme with both verification-key length and signature length $O(n^\epsilon)$. Additionally, by Theorem 18 and 20, there exists ECCs (E_p, D_p) , (E_m, D_m) with information rate $R(n) = O(1/n)$, and such that (E_p, D_p) has error-rate $1/4 - \epsilon$ and (E_m, D_m) is $1/2 - \epsilon$ efficiently list decodable. By Lemma 34, we get that the error rate $\eta(n)$ is (approximately) maximized¹¹ when

$$c(n, m) = \gamma(n, m) + \alpha(n) = 3\alpha(n) + 2m(n)\beta(n).$$

(that is, the two expressions inside the $\min$ are the same). In this case,

$$\eta(n) = \eta_p \cdot \frac{\alpha(n)}{3\alpha(n) + 2m(n)\beta(n)}.$$

Note that we can set the padding parameter $\rho(n)$ to be a sufficiently big polynomial such that $\epsilon \cdot \alpha(n) \geq 2m(n)\beta(n)$; it follows that the error rate is at least

$$\frac{\eta_p}{3 + \epsilon} = \frac{(1/4) - \epsilon}{3 + \epsilon} \geq \frac{1}{12} - \epsilon$$

¹¹For simplicity, we ignore ϵ terms.

By Lemma 34, the information rate of Q is $c(n, m)/2m = O(n^\epsilon)$. This concludes the first part of the theorem.

If additionally assuming the existence of subexponentially-hard one-way functions, it instead follows that $c(n, m) \leq O(m \cdot \text{polylog} n)$. $\square$

5.2 Negative Results

We show that our positive result is optimal in two ways:

1. The existence of one-way functions is necessary.
2. If a constant error rate is desired, it is impossible to achieve an information rate of $\Omega(1/\log n)$.

The necessity of one-way functions We show that the existence of one-way functions is necessary to achieve computational knowledge-preserving interactive coding with error rate $1/\text{poly}(n)$.

Theorem 35. *For every polynomial $m(\cdot)$, the existence of a computational knowledge-preserving interactive coding scheme Q with error rate $\eta(n, m) \geq 1/m(n)$ implies the existence of one-way functions.*

Proof. At a high level, the theorem follows by observing that the forking attacker $\tilde{A}^*$ in the proof of Theorem 27 can be approximated efficiently if one-way functions do not exist, and so can the channel adversary. We turn to a formal proof. Assume for contradiction that one-way functions do not exist, we show that there does not exist a computational knowledge-preserving interactive coding scheme with polynomial error and information rate.

Consider some polynomial $m(\cdot)$ and computational knowledge-preserving interactive coding protocol $Q = (Q_1, Q_2)$ with error rate $\eta(n, m) \geq 1/m(n)$; let $M(n, m, \ell)$ be a polynomial upper bound on the number queries made by Q_1, Q_2 to its oracles (where n is the security parameter, m and ℓ are the round complexity and communication complexity of the protocol π to be encoded).

Let $t = M(n, m, 2mn) + 1$. Let $\pi = (A, B, \mathcal{X}_A, \mathcal{X}_B)$ be the ping-pong protocol defined in the proof of Theorem 27 with m rounds and using t -wise independence hash functions as inputs. Let $m'(\cdot)$ be the round complexity of the encoded protocol Q^π . We show that Q^π cannot be a computational knowledge-preserving variant of π by obtaining a privacy breach using the same forking attacker $\tilde{A}^*$ as in the proof of Theorem 27, but relying on the assumed non-existence of one-way functions, to make $\tilde{A}^*$ and the channel C that emulates the attacker $\tilde{A}^*$, efficient.

Recall that the only inefficient step of $\tilde{A}^*$ is in Step 2, where $\tilde{A}^*$ needs to sample a fresh input-randomness pair (x', r'_1) conditioned on the first $j - 1$ rounds partial transcript T of the execution $Q_1^{A(x)}(r_1) \leftrightarrow Q_2^{B(y)}(r_2)$. Let f denote the function that maps (x, y, r_1, r_2, j) to T . Clearly, f is efficient. By Theorem 12 and the assumed non-existence of one-way functions, there exists a polynomial-time inverter M such that $(T, M(T))$ and $(T, (x, y, r_1, r_2, j))$ has statistical distance at most $1/8m'(n)$ for infinitely many $n \in \mathbb{N}$. (Note that M only works for infinitely many $n \in \mathbb{N}$. Nevertheless, this is sufficient.) Namely, M can approximately sample a random pre-image of a random transcript T with small inverse polynomial error. Relying on the inverter M , $\tilde{A}^*$ can be made efficient straightforwardly, with the following modification; let us denote the modified attacker $\tilde{A}_{\text{eff}}^*$.

- In the second step, $\tilde{A}_{\text{eff}}^*$ applies M to the partial transcript T to obtain a pre-image (x', y', r'_1, r'_2, j') . Then $\tilde{A}_{\text{eff}}^*$ uses (x', r'_1) to continue the attack as before. Namely, $\tilde{A}_{\text{eff}}^*$ continues the interaction, but with the input and randomness to Q_1 switched to x' and r'_1 , for as many rounds as possible, subject to that the number of bits it transmits since round j does not exceed $\eta(n) \cdot \text{CC}_n(Q^\pi)$.

By construction, the attacker $\tilde{A}_{\text{eff}}^*$ runs in polynomial time; additionally, since the channel C used in the proof of Theorem 27 perfectly mimics the attacker, it is also efficient.

We now show that $\tilde{A}_{\text{eff}}^*$ approximate $\tilde{A}^*$ well over random inputs, which implies that $\tilde{A}_{\text{eff}}^*$ can also obtain a privacy breach with inverse polynomial probability for infinitely many n . Specifically, we show that the following two experiments have a statistical distance of at most $1/8m'(n)$ for infinitely many $n \in \mathbb{N}$.

- $\text{Exp}_1(1^n) = \left\{ x \leftarrow \mathcal{X}_A(1^n), y \leftarrow \mathcal{X}_B(1^n) : \tilde{A}^*(x) \leftrightarrow Q_2^{B(y)} \right\}$
- $\text{Exp}_2(1^n) = \left\{ x \leftarrow \mathcal{X}_A(1^n), y \leftarrow \mathcal{X}_B(1^n) : \tilde{A}_{\text{eff}}^*(x) \leftrightarrow Q_2^{B(y)} \right\}$

Claim 36. *For infinitely many $n \in \mathbb{N}$, the statistical distance between $\text{Exp}_1(1^n)$ and $\text{Exp}_2(1^n)$ is at most $1/8m'(n)$.*

Proof. Recall that both experiments are determined by the values of $(x, y, r_1, r_2, j, x', r'_1)$, where (x, y, r_1, r_2, j) are independently and uniformly sampled, and then in Exp_1 , (x', r'_1) are sampled conditioned on the first $j - 1$ round partial transcript T of the execution $Q_1^{A(x)}(r_1) \leftrightarrow Q_2^{B(y)}(r_2)$, and in Exp_2 , (x', r'_1) are sampled by first applying the efficient inverter to obtain (x', y', r'_1, r'_2, j') and then discarding (y', r'_2, j') .

Fix a security parameter $n \in \mathbb{N}$ such that M works; that is, such that $(T, M(T))$ and $(T, (x, y, r_1, r_2, j))$ have a statistical distance of at most $1/8m'(n)$. This implies that the distributions of (T, x', r'_1) in $\text{Exp}_1(1^n)$ and $\text{Exp}_2(1^n)$ have a statistical distance of at most $1/8m'(n)$, since projection (i.e., removing (y', r'_2, j')) cannot increase statistical distance. We claim that additionally, the distribution of the whole tuple $(x, y, r_1, r_2, j, x', r'_1)$ in $\text{Exp}_1(1^n)$ and $\text{Exp}_2(1^n)$ have a statistical distance of at most $1/8m'(n)$ as well. To see this, as a thought experiment, we can view the experiments as sampled in the following different order: first, T is sampled, then (x', r'_1) are sampled conditioned on T , and finally (x, y, r_1, r_2, j) are sampled conditioned on T (note that conditioned on T , (x', r'_1) and (x, y, r_1, r_2, j) are independent). Note that in both experiments, (x, y, r_1, r_2, j) are sampled in an identical way after sampling (T, x', r'_1) , so it does not increase the statistical distance. Therefore, the statistical distance between $\text{Exp}_1(1^n)$ and $\text{Exp}_2(1^n)$ on this security parameter n is at most $1/8m'(n)$. $\square$

Now, since $\tilde{A}^*$ obtains a privacy breach with probability $\geq 1/4m'(n)$ in $\text{Exp}_1(1^n)$ for sufficiently large $n \in \mathbb{N}$, and $\text{Exp}_1(1^n)$ and $\text{Exp}_2(1^n)$ have statistical distance at most $1/8m'(n)$ for infinitely many $n \in \mathbb{N}$, it follows that $\tilde{A}_{\text{eff}}^*$ can also obtain a privacy breach with probability $\geq 1/8m'(n)$ in $\text{Exp}_2(1^n)$ for infinitely many $n \in \mathbb{N}$. This shows that Q^π is not a computationally knowledge-preserving variant of π and completes the proof. $\square$

The necessity of a communication complexity blow-up We show that every computational knowledge-preserving interactive coding scheme with constant error rate must have an information rate of $o(1/\log n)$, showing that the inverse polylogarithmic rate achieved in Theorem 33 (assuming subexponentially hard one-way functions) is essentially optimal.

Theorem 37. *Assume the existence of a computational knowledge-preserving interactive coding scheme with information rate $R(n)$ and error rate $\eta(n)$. Then $R(n)\eta(n) \in o(1/\log(n))$.*

Proof. The theorem is a direct consequence of Theorem 38 proven in Section 6. $\square$

6 Lower Bound for Non-constructive Schemes

In this section, we present an impossibility result for the computational setting, which applies even to *non-constructive* interactive coding scheme. In particular, for every $\eta(n) > 1/\log n$, we demonstrate the existence of protocol π with communication complexity $1/\eta(n)$ such that *every* computationally η -error resilient, computationally knowledge-preserving variant of π must have communication complexity at least $\omega(\log n)$. In particular, for the case $\eta(n) = O(1)$, we get that the information rate also for non-constructive interactive coding is at most $o(1/\log n)$. (Note that this result is interesting also in the information theoretic setting, since in contrast to Theorem 27, we here provide an impossibility result also for non-constructive interactive coding.)

Theorem 38. *For every function $\eta(\cdot)$ such that $\eta(n) \geq O(1/\log n)$, there exists a protocol $\pi = (A, B, \mathcal{X}_A, \mathcal{X}_B)$ with communication complexity $\text{CC}_n(\pi) = O(1/\eta(n))$ such that for every protocol $\pi' = (Q_1, Q_2, \mathcal{X}_A, \mathcal{X}_B)$ that is a computationally knowledge-preserving variant of π and is computationally η -error resilient, the communication complexity of π' is at least $\omega(\log n)$.*

Proof. We here consider a somewhat simpler variant of the ping-pong protocol, which we refer to as the “bit-exchange” protocol π (the protocol is almost identical to a protocol used in [CP11] in a quite different context). We show that *any* protocol π' with communication complexity $O(\log n)$ that is computational η -error resilient cannot be a computationally knowledge preserving variant of π .

Similarly to the proof of Theorem 27, let us first formally define π and formalize a security property that it satisfies.

The “bit-exchange” protocol π . Let $m(\cdot) = \lceil 2/\eta(\cdot) \rceil$ [**Kai-Min’s Note: Make $m = 2/\eta$ instead of $1/\eta$ so that we can do Markov on the length of i -th block.**] and consider the following simple bit-exchange protocol $\pi = (A, B, \mathcal{X}_A, \mathcal{X}_B)$, where both players simply send their inputs to each other, bit-by-bit. On security parameter n , let $m = m(n)$ and $\mathcal{X}_A(1^n) = \mathcal{X}_B(1^n) = \{0, 1\}^m$. π is a deterministic m -round protocol that on the inputs $x \in \{0, 1\}^m$ and $y \in \{0, 1\}^m$ proceeds as follows: at each round $i \in [m]$, A sends $a_i = x_i$ to B , who sends back $b_i = y_i$ to A , where x_i, y_i denote the i -th bit of x, y , respectively. At the end of the interaction, both A and B output the whole transcript (a, b) , where $a = (a_1, \dots, a_m) \in \{0, 1\}^m$ and $b = (b_1, \dots, b_m) \in \{0, 1\}^m$.

“Guess-the-next-bit” security of the bit-exchange protocol. We say that player *matches* in a round $i \in [m]$ if it guesses the bit sent by its opponent in the next message. Formally, for $i \in [m]$, defined the predicates $\text{Match}_{1,i}(a, b) = 1$ iff $a_i = b_i$, $\text{Match}_{2,i}(a, b) = 1$ iff $b_i = a_{i+1}$. Note that if the inputs are uniformly distributed, then for every $i \in [m]$, the probability that each player matches in round i with probability *exactly* $1/2$.

Claim 39. *For every adversarial strategy A^* , every $n \in \mathbb{N}$ and $i \in [m]$,*

$$\Pr[x, y \leftarrow \{0, 1\}^m : \text{Match}_{1,i}(\text{out}_2[A^*(x) \leftrightarrow B(y)]) = 1] = 1/2.$$

Similarly, for every adversarial strategy B^* , every $n \in \mathbb{N}$ and $i \in [m-1]$,

$$\Pr[x, y \leftarrow \{0, 1\}^m : \text{Match}_{2,i}(\text{out}_1[A(x) \leftrightarrow B^*(y)]) = 1] = 1/2.$$

Proof. The claim trivially follows by noting that for every A^* (resp., B^*), y_i (resp., x_{i+1}) remains uniformly random conditioned on the view of A^* (resp., B^*) when A^* (resp., B^*) needs to send its i -th message. $\square$

Now, consider some computationally knowledge preserving variant $\pi' = (Q_1, Q_2, \{0, 1\}^{m(n)}, \{0, 1\}^{m(n)})$ of π that has $O(\log n)$ communication complexity and is η -error resilient. Let $c(n) = \max\{\text{CC}_n(\pi'), \log n\}$, and $m'(\cdot)$ be the round complexity of π' . As in the proof of Theorem 27, we first show that π' needs to be “implicitly executing” π in a chronological order. In contrast to this step in the proof of Theorem 27, we here rely on the knowledge preservice property of π' to demonstrate this.

Implicit bit-exchange computation. Our formalization of implicit computation here is quite different from how it was formalized in the proof of Theorem 27. Here we rely on information-theoretic definitions of what it means to implicitly execute π (which is what allows us to provide an impossibility result also for non-constructive coding schemes). Towards formalizing it, let us first introduce some additional definitions.

For two strings T and T' , we denote by $T \preceq T'$ (resp., $T \prec T'$) that T is a prefix (resp., proper prefix) of T' . Fix a security parameter n and let $m = m(n)$. If $x, y \in \{0, 1\}^m$ and T is a partial transcript, then let $p(x, y, T) = \Pr[T \preceq \text{trans}[Q_1(x) \leftrightarrow Q_2(y)]]$; that is, the probability that T is consistent with the execution of $Q_1(x) \leftrightarrow Q_2(y)$. Let $\delta > \epsilon \in (0, 1)$ be two parameters (to be determined later). For inputs $x, y \in \{0, 1\}^m$, a partial transcript T , $i \in [m]$ and $\beta \in \{0, 1\}$, we say that (y, T) (resp., (x, T)) is (δ, ϵ) -binding for (i, β) if the following two conditions hold:

1. There exists $x' \in \{0, 1\}^m$ with $x'_i = \beta$ (resp., $y' \in \{0, 1\}^m$ with $y'_i = \beta$) such that $p(x', y, T) \geq \delta$ (resp., $p(x, y', T) \geq \delta$).
2. For every $x' \in \{0, 1\}^m$ with $x'_i \neq \beta$ (resp., $y' \in \{0, 1\}^m$ with $y'_i \neq \beta$), $p(x', y, T) \leq \epsilon$ (resp., $p(x, y', T) \leq \epsilon$).

Intuitively, this means that (y, T) “determines” $x_i = \beta$. Consider an execution $T \leftarrow \text{trans}[Q_1(x, r_1) \leftrightarrow Q_2(y, r_2)]$ for some inputs $x, y \in \{0, 1\}^m$ and randomness $r_1, r_2 \in \{0, 1\}^\infty$. Note that since the probability $p(x, y, T)$ can be estimated by sampling, the above two conditions can be checked in time $\text{poly}(2^c, 1/\delta, 1/\epsilon)$, which allows player 1 to “decode” the bit y_i from (x, T) when (x, T) is binding for player 2’s i -th bit input. Specifically, there is an algorithm Dec_1 with runtime $\text{poly}(n, 2^c, 1/\delta, 1/\epsilon)$ that takes $(x, T, i, \delta, \epsilon)$ as input such that (a) if (x, T) is (δ, ϵ) -binding for (i, β) , then Dec_1 outputs β with probability at least $1 - 2^{-n}$, and (b) if (x, T) is *not* $(\delta/2, 2\epsilon)$ -binding for both $(i, 0)$ and $(i, 1)$, then Dec_1 outputs $\perp$ with probability at least $1 - 2^{-n}$. Analogously, there is an algorithm Dec_2 to “decode” player 1’s bit x_i from (y, T) .

Define the i -th binding-point for player 1 (resp., 2) of the execution (x, y, r_1, r_2) with parameters (δ, ϵ) to be the shortest partial transcript $T_{1,i} \preceq T$ (resp., $T_{2,i} \preceq T$) such that (i) the last message in $T_{1,i}$ (resp., $T_{2,i}$) is sent by player 1 (resp. player 2), and (ii) $(y, T_{1,i})$ (resp., $(x, T_{2,i})$) is (δ, ϵ) -binding for (i, x_i) (resp., (i, y_i)); if no such partial transcript exists, define $T_{1,i} = \perp$ (resp., $T_{2,i} = \perp$). For convenient, we use notation $T_{d,i}(x, y, r_1, r_2; \delta, \epsilon)$ to denote the i -th binding-point for player d of the execution (x, y, r_1, r_2) with parameters (δ, ϵ) .

Intuitively, $T_{d,i}$ is the point where player d sends its i -bit to the other player (corresponding to the i -th message in the implicit computation of π). Formally, let the predicate $\text{ImplicitComp}(x, y, r_1, r_2; \delta, \epsilon) = 1$ iff (i) $T_{d,i}(x, y, r_1, r_2; \delta, \epsilon) \neq \perp$ for every $d \in \{1, 2\}$ and $i \in [m]$, (ii) $T_{1,i} \prec T_{2,i}$ for every $i \in [m]$, and (iii) $T_{2,i} \prec T_{1,i+1}$ for every $i \in [m-1]$; that is, the i -th binding-points for both players occur in a chronological order corresponding to the execution of π and do not occur at the same time. When $\text{ImplicitComp}(x, y, r_1, r_2; \delta, \epsilon) = 1$, the interaction of π' can be partitioned into non-empty chunks as before: for every $i \in [m]$, the i -th chunk of the execution $Q_1(x, r_1) \leftrightarrow Q_2(y, r_1)$ starts after $T_{2,i-1}$ and finishes at the end of $T_{2,i}$; that is, the i -th chunk starts when player 1 starts sending i -message and finished when player 1 receives the i -th message from player 2.

The following lemma shows that **ImplicitComp** holds with high probability for every input pair (x, y) and appropriate (sufficiently small) inverse polynomial δ and ϵ .

Lemma 40 (Implicit Computation Lemma). *For every polynomial $q(n) \geq n^5 \cdot 2^{5(m(n)+c(n))}$, for sufficiently large $n \in \mathbb{N}$,*

$$\Pr[x, y \leftarrow \{0, 1\}^{m(n)}, r_1, r_2 \in \{0, 1\}^\infty : \text{ImplicitComp}(x, y, r_1, r_2; 1/q(n), 1/q^2(n)) = 1] \geq 1 - 1/n.$$

Proof. Let $\delta(n) = 1/q(n)$ and $\epsilon(n) = 1/q^2(n)$. Let $\mu(n)$ be a negligible function such that π' is a computational knowledge preserving variant of π with respect to $\mu(n)$ (for both completeness and computational knowledge preservance). Recall that the completeness property says that for every $n \in \mathbb{N}$, every $x, y \in \{0, 1\}^{m(n)}$,

$$\Pr[\text{out}[Q_1(x) \leftrightarrow Q_2(y)] = \text{out}[A(x) \leftrightarrow B(y)] \geq 1 - \mu(n).$$

Fix n to be a sufficiently large security parameter such that $\mu(n) \leq 1/q^5(n)$. We first show that for every $x, y \in \{0, 1\}^m$,

$$\Pr[r_1, r_2 \leftarrow \{0, 1\}^\infty : \forall d \in \{1, 2\}, i \in [m], T_{d,i}(x, y, r_1, r_2; \delta, \epsilon) \neq \perp] \geq 1 - 1/2n.$$

Namely, condition (i) of the predicate **ImplicitComp** is satisfied for every inputs $x, y \in \{0, 1\}^{m(n)}$ with high probability. Note that it suffices to show that with probability at least $1 - 1/2n$ over $T \leftarrow \text{trans}[Q_1(x) \leftrightarrow Q_2(y)]$, for every $i \in [m]$, (y, T) is (δ, ϵ) -binding for (i, x_i) and (x, T) is (δ, ϵ) -binding for (i, y_i) ; that is, every bit of both players' input are eventually binding at the end of the execution. To show this, it suffices to show the following two statements.

1. For every $x, y \in \{0, 1\}^{m(n)}$, $\Pr[T \leftarrow \text{trans}[Q_1(x) \leftrightarrow Q_2(y)] : p(x, y, T) \geq \delta(n)] \geq 1 - 1/2n$.
2. For every $x \neq x' \in \{0, 1\}^{m(n)}$, $y \neq y' \in \{0, 1\}^{m(n)}$, and a full transcript $T \in \{0, 1\}^*$,

$$\min\{p(x, y, T), p(x', y, T)\} \leq 3\mu(n) \text{ and } \min\{p(x, y, T), p(x, y', T)\} \leq 3\mu(n).$$

Statement (1) implies that condition (1) in the definition of binding is satisfied with probability at least $1 - 1/2n$, and when the event in statement (1) holds, statement (2) guarantees that condition (2) is also satisfied. Now, statement (1) follows by noting that

$$\Pr[T \leftarrow \text{trans}[Q_1(x) \leftrightarrow Q_2(y)] : p(x, y, T) \leq \delta(n)] \leq \delta(n) \cdot 2^{c(n)} \leq 1/2n.$$

For statement (2), suppose that there exist $x \neq x' \in \{0, 1\}^{m(n)}$, $y \in \{0, 1\}^{m(n)}$ and a full transcript $T \in \{0, 1\}^*$ such that both $p(x, y, T), p(x', y, T) \geq 3\mu(n)$. Consider two executions $Q_1(x) \leftrightarrow Q_2(y)$ and $Q_1(x') \leftrightarrow Q_2(y)$. Note that conditioned on the full transcript, the output of player 2 is independent of the input of player 1. Thus, when the transcript is T , player 2 must generate an incorrect output for one of the two executions (i.e., $\text{out}_2[Q_1(x) \leftrightarrow Q_2(y)] \neq \text{out}_2[A(x) \leftrightarrow B(y)]$, which is simply (x, y)). Since in both executions, T occurs with probability at least $3\mu(n)$, it follows that player 2 produces incorrect output in one of the execution with probability $\geq 3\mu(n)/2 > \mu(n)$, violating the completeness property. This shows that $\min\{p(x, y, T), p(x', y, T)\} \leq 3\mu(n)$ for every $x \neq x' \in \{0, 1\}^{m(n)}$, $y \in \{0, 1\}^{m(n)}$, and full transcript $T \in \{0, 1\}^*$. An analogous argument shows that $\min\{p(x, y, T), p(x, y', T)\} \leq 3\mu(n)$ for every $x \in \{0, 1\}^{m(n)}$, $y \neq y' \in \{0, 1\}^{m(n)}$, and full transcript $T \in \{0, 1\}^*$.

We proceed to show that condition (ii) and (iii) hold with high probability, relying on the following claim.

Claim 41. For every $i \in [m]$, suppose one of the following holds.

- $\Pr[x, y \leftarrow \{0, 1\}^{m(n)}, r_1, r_2 \leftarrow \{0, 1\}^\infty : x_i \neq y_i \wedge T_{2,i}(x, y, r_1, r_2; \delta, \epsilon) \prec T_{1,i}(x, y, r_1, r_2; \delta, \epsilon)] \geq 1/q(n)$, or
- $\Pr[x, y \leftarrow \{0, 1\}^{m(n)}, r_1, r_2 \leftarrow \{0, 1\}^\infty : x_i = y_i \wedge T_{2,i}(x, y, r_1, r_2; \delta, \epsilon) \prec T_{1,i}(x, y, r_1, r_2; \delta, \epsilon)] \geq 1/q(n)$.

Then there exists an adversarial strategy $\tilde{A}^*$ for player 1 such that

$$\left| \Pr \left[x, y \leftarrow \{0, 1\}^m, o \leftarrow \text{out}_2[\tilde{A}^*(x) \leftrightarrow Q_2(y)] : \text{Match}_{1,i}(o) = 1 \right] - 1/2 \right| \geq 1/8q(n).$$

Similarly, for every $i \in [m-1]$, suppose one of the following holds.

- $\Pr[x, y \leftarrow \{0, 1\}^{m(n)}, r_1, r_2 \leftarrow \{0, 1\}^\infty : y_i \neq x_{i+1} \wedge T_{1,i+1}(x, y, r_1, r_2; \delta, \epsilon) \prec T_{2,i}(x, y, r_1, r_2; \delta, \epsilon)] \geq 1/q(n)$, or
- $\Pr[x, y \leftarrow \{0, 1\}^{m(n)}, r_1, r_2 \leftarrow \{0, 1\}^\infty : y_i = x_{i+1} \wedge T_{1,i+1}(x, y, r_1, r_2; \delta, \epsilon) \prec T_{2,i}(x, y, r_1, r_2; \delta, \epsilon)] \geq 1/q(n)$, or

Then there exists an adversarial strategy $\tilde{B}^*$ for player 2 such that

$$\left| \Pr \left[x, y \leftarrow \{0, 1\}^m, o \leftarrow \text{out}_2[Q_1(x) \leftrightarrow \tilde{B}^*(y)] : \text{Match}_{2,i}(o) = 1 \right] - 1/2 \right| \geq 1/8q(n).$$

Note that the conclusions of the claim contradict to the knowledge preserving property. Thus, it follows that for every $i \in [m]$,

$$\Pr[x, y \leftarrow \{0, 1\}^{m(n)}, r_1, r_2 \leftarrow \{0, 1\}^\infty : T_{2,i}(x, y, r_1, r_2; \delta, \epsilon) \prec T_{1,i}(x, y, r_1, r_2; \delta, \epsilon)] \leq 2/q(n), \text{ and}$$

$$\Pr[x, y \leftarrow \{0, 1\}^{m(n)}, r_1, r_2 \leftarrow \{0, 1\}^\infty : T_{1,i+1}(x, y, r_1, r_2; \delta, \epsilon) \prec T_{2,i}(x, y, r_1, r_2; \delta, \epsilon)] \leq 2/q(n).$$

It now follows by a union bound that condition (ii) and (iii) hold with probability at least $1 - 4m/q(n) \geq 1 - 1/2n$, and another union bound concludes that

$$\Pr[x, y \leftarrow \{0, 1\}^{m(n)}, r_1, r_2 \in \{0, 1\}^\infty : \text{ImplicitComp}(x, y, r_1, r_2; 1/q(n), 1/q^2(n)) = 1] \geq 1 - 1/n.$$

It remains to prove the claim.

Proof of Claim 41. We start by proving the first case of the claim. Namely, we show that if

$$\Pr[x, y \leftarrow \{0, 1\}^{m(n)}, r_1, r_2 \leftarrow \{0, 1\}^\infty : x_i \neq y_i \wedge T_{2,i}(x, y, r_1, r_2; \delta, \epsilon) \prec T_{1,i}(x, y, r_1, r_2; \delta, \epsilon)] \geq 1/q(n),$$

then

$$\Pr \left[x, y \leftarrow \{0, 1\}^m, o \leftarrow \text{out}_2[\tilde{A}^*(x) \leftrightarrow Q_2(y)] : \text{Match}_{1,i}(o) = 1 \right] \geq 1/2 + 1/8q(n).$$

At a high level, the idea is simple: whenever y_i is bound at point $T_{2,i}$ before x_i is bound, $\tilde{A}^*$ can first decode y_i from $(x, T_{2,i})$, and then if $x_i = y_i$, $\tilde{A}^*$ simply continues honestly, but if $x_i \neq y_i$, $\tilde{A}^*$ changes its input from x to some x' with $x'_i = y_i$ ($\tilde{A}^*$ can do so since x_i is not bound yet). Formally, on input $x \in \{0, 1\}^m$, $\tilde{A}^*$ performs the following strategy to interact with Q_2 :

- $\tilde{A}^*$ samples a uniform randomness $r_1 \leftarrow \{0, 1\}^\infty$ and starts by honestly executing the protocol Q_1 , but at the end of each round j , $\tilde{A}^*$ runs $\text{Dec}_1(x, T_j, i, \delta, \epsilon)$ where T_j is the current partial transcript. If Dec_1 outputs $\perp$, then $\tilde{A}^*$ simply continues honestly at round $j + 1$. If Dec_1 outputs $\beta \in \{0, 1\}$, let T^* denote the current transcript and $\tilde{A}^*$ proceeds as follows.

- If $\beta = x_i$, then $\tilde{A}^*$ simply continues the execution honestly throughout.
- If $\beta \neq x_i$, then $\tilde{A}^*$ uses rejection sampling to sample a random (x', r'_1) conditioned on $x'_i = \beta$ and the current transcript T^* . $\tilde{A}^*$ cuts-off the rejection sampling procedure when it fails for more than $M = O(n2^m/\delta\epsilon)$ samples; in this case, $\tilde{A}^*$ sets $(x', r'_1) = (x, r_1)$ (i.e., does not change the input-randomness pair). Then $\tilde{A}^*$ continues executing the protocol Q_1 with the alternative input-randomness pair (x', r'_1) throughout.

We proceed to analyze $\tilde{A}^*$. The following sub-claim says that $\text{Match}_{1,i}$ holds with high probability when $x_i = y_i$.

Sub-claim 42. $\Pr \left[x, y \leftarrow \{0, 1\}^m, o \leftarrow \text{out}_2[\tilde{A}^*(x) \leftrightarrow Q_2(y)] : x_i = y_i \wedge \text{Match}_{1,i}(o) = 1 \right] \geq 1/2 - 1/8q(n).$

Proof. First note that by completeness, we have

$$\Pr [x, y \leftarrow \{0, 1\}^m, o \leftarrow \text{out}_2[Q_1(x) \leftrightarrow Q_2(y)] : x_i = y_i \wedge \text{Match}_{1,i}(o) = 1] \geq 1/2 - \mu(n).$$

Additionally, note that when $x_i = y_i$ and there is no “decoding error” (i.e., during the execution Dec_1 does not return $\beta = \bar{y}_i$), then $\tilde{A}^*$ simply executes Q_1 honestly. Thus, to lower bound the probability of the desired event, it suffices to upper bound the probability that a decoding error occur during the execution. Now, observe that during the execution, the property of Dec_1 ensures that when $p(x, y, T) > 2\epsilon$, $\text{Dec}_1(x, T, i, \delta, \epsilon)$ outputs incorrect answer $\beta = \bar{y}_i$ with probability at most 2^{-n} . Noting that for every (x, y) , $\Pr[T \leftarrow \text{trans}[Q_1(x) \leftrightarrow Q_2(y)] : p(x, y, T) \leq 2\epsilon] \leq 2^{c(n)} \cdot 2\epsilon$, and that $\tilde{A}^*(x)$ invokes Dec_1 at most $m'(n)$ times, the probability that a decoding error occur can be upper bounded by $m'(n) \cdot 2^{-n} + 2^{c(n)} \cdot 2\epsilon$. A final union bound implies that

$$\begin{aligned} & \Pr \left[x, y \leftarrow \{0, 1\}^m, o \leftarrow \text{out}_2[\tilde{A}^*(x) \leftrightarrow Q_2(y)] : x_i = y_i \wedge \text{Match}_{1,i}(o) = 1 \right] \\ & \geq 1/2 - \mu(n) - m'(n) \cdot 2^{-n} - 2^{c(n)} \cdot 2\epsilon \geq 1/2 - 1/8q(n). \end{aligned}$$

□

We next show that when $x_i \neq y_i$, $\text{Match}_{1,i}$ still holds with noticeable probability.

Sub-claim 43. $\Pr \left[x, y \leftarrow \{0, 1\}^m, o \leftarrow \text{out}_2[\tilde{A}^*(x) \leftrightarrow Q_2(y)] : x_i \neq y_i \wedge \text{Match}_{1,i}(o) = 1 \right] \geq 1/4q(n).$

Proof. Let **Good** denote the event that during the execution, the tuple (x, y, T^*) satisfies the following three conditions: (i) $x_i \neq y_i$, (ii) $p(x, y, T^*) \geq \delta$, (iii) $\text{Dec}_1(x, T^*, i, \delta, \epsilon)$ outputs y_i (i.e., Dec_1 decodes correctly), and (iv) let T^- denote the partial transcript obtained by removing the last message from T^* (thus, the last message is from player 1 to player 2); (y, T^-) is *not* (δ, ϵ) -binding for (i, x_i) .

We first show that **Good** happens with probability at least $1/2q(n)$. Recall that $\tilde{A}^*$ executes Q_1 honestly before the end of T^* , and that in the execution of $Q_1(x) \leftrightarrow Q_2(y)$,

$$\Pr[x, y \leftarrow \{0, 1\}^{m(n)}, r_1, r_2 \leftarrow \{0, 1\}^\infty : x_i \neq y_i \wedge T_{2,i}(x, y, r_1, r_2; \delta, \epsilon) \prec T_{1,i}(x, y, r_1, r_2; \delta, \epsilon)] \geq 1/q(n).$$

Now, consider any tuple (x, y, r_1, r_2) such that the above event hold, and let $T_{2,i} = T_{2,i}(x, y, r_1, r_2; \delta, \epsilon)$ and $T_{2,i}^-$ be $T_{2,i}$ with the last message removed. Note that $\text{Dec}_1(x, T_{2,i}, i, \delta, \epsilon)$ outputs y_i with probability at least $1 - 2^{-n}$, and that $(y, T_{2,i}^-)$ is *not* (δ, ϵ) -binding for (i, x_i) . Therefore, in the execution of $\tilde{A}^*(x) \leftrightarrow Q_2(y)$, conditioned on such (x, y, r_1, r_2) , we have that with probability at

least $1 - m'(n) \cdot 2^{-n}$, it holds that the above four conditions hold with $T^* \preceq T_{2,i}$ (the $m'(n) \cdot 2^{-n}$ term comes from union bound over at most $m'(n)$ invocation of Dec_1 ; note that it is possible that $T^* \prec T_{2,i}$ and in this case, the probability that Dec_1 outputs $\bar{y}_i$ is also upper bounded by 2^{-n}). Therefore, the probability that the **Good** event happens is at least $1/q(n) \cdot (1 - m'(n) \cdot 2^{-n}) \geq 1/2q(n)$.

We next show that conditioned on any tuple (x, y, T^*) such that the **Good** event holds, with probability at least $(1 - 1/8q(n))$, (a) $\tilde{A}^*$ can successfully sample a fresh input-randomness pair (x', r'_1) through rejection sampling, and (b) Q_2 outputs (x', y) at the end of the execution. For (a), the fact that (y, T^-) is not (δ, ϵ) -binding for (i, x_i) and $p(x, y, T^-) \geq p(x, y, T^*) \geq \delta$ implies that there exists some x^* such that $x_i^* \neq x_i$ and $p(x^*, y, T^-) \geq \epsilon$. Now, the facts that the last message of T^* is sent by player 2 and that $p(x, y, T^*) \geq \delta$ imply that $p(x^*, y, T^*) \geq \epsilon\delta$. Therefore, the chance of sampling a pair (x', r'_1) that is consistent with T^* is at least $2^{-m}\epsilon\delta$, and rejection sampling with $M = O(n2^m/\delta\epsilon)$ samples can succeed with probability at least $1 - 2^{-n}$. For (b), note that the execution now is equivalent to an honest execution of $Q_1(x') \leftarrow Q_2(y)$ conditioned on transcript T^* . By completeness, Q_2 outputs (x', y) with probability at least $1 - \mu(n)/p(x', y, T^*)$. Now, recall that there exists x^* with $p(x^*, y, T^*) \geq \epsilon\delta$. A Markov argument shows that the rejection sampling procedure returns an x' with $p(x', y, T^*) \leq \epsilon\delta 2^{-m}/8q(n)$ is at most $1 - 1/8q(n)$. Therefore, when condition (a) holds, we have that condition (b) also holds with probability at least $(1 - 1/8q(n)) \cdot (1 - \mu(n)8q(n)2^m/\epsilon\delta) \geq (1 - 1/4q(n))$. Therefore, the probability that both conditions hold is at least $(1 - 2^{-n}) \cdot (1 - 1/4q(n)) \geq 1 - 1/2q(n)$.

Finally, we can conclude that

$$\begin{aligned} & \Pr \left[x, y \leftarrow \{0, 1\}^m, o \leftarrow \text{out}_2[\tilde{A}^*(x) \leftrightarrow Q_2(y)] : x_i \neq y_i \wedge \text{Match}_{1,i}(o) = 1 \right] \\ & \geq 1/2q(n) \cdot (1 - 1/2q(n)) \geq 1/4q(n). \end{aligned}$$

□

Combining the above two sub-claims, we have that

$$\begin{aligned} & \Pr \left[x, y \leftarrow \{0, 1\}^m, o \leftarrow \text{out}_2[\tilde{A}^*(x) \leftrightarrow Q_2(y)] : \text{Match}_{1,i}(o) = 1 \right] \\ & \geq 1/2 - 1/8q(n) + 1/4q(n) \geq 1/2 + 1/8q(n). \end{aligned}$$

This completes the proof of the first case of the claim. The remaining three cases of the claim can be proved by an analogous argument.

□

□

Obtaining a match-deviation in π' Now that we have established that implicit computation holds in π' with high probability, we can obtain a deviation on the matching probability in π' relying on a similar forking attack as in the proof of Theorem 27; such deviation shows that π' cannot be a knowledge-preserving variant of π .

Let $\delta(n) = 1/n^5 \cdot 2^{5(m(n)+c(n))}$ and $\epsilon(n) = \delta^2(n)$. For $i \in [m]$ and $e \in \{0, 1\}$, consider the following adversarial strategy $\tilde{A}_{i,e}^*$ for player 1. On input $x \in \{0, 1\}^m$, and randomness r_1 , $\tilde{A}^*$ performs the follow “forking” attack:

1. $\tilde{A}_{i,e}^*$ uniformly picks a random round $j \leftarrow [m'(n)]$ and honestly executes the encoded protocol Q_1 up to the end of $(j - 1)$ -th round. Let T be the resulting partial transcript.

2. $\tilde{A}_{i,e}^*$ samples a fresh input-randomness pair (x', r'_1) conditioned on the partial transcript T using rejection sampling with a sufficiently large cuts-off parameter $M = 1/\epsilon^5$. Then, $\tilde{A}^*$ continues executing Q_1 but now with inputs x' and randomness r'_1 , for as many rounds as possible, subject to the restriction that the number of bits it transmitted since round j does not exceed $\eta(n) \cdot \text{CC}_n(\pi')$. Let T' be the resulting partial transcript.
3. $\tilde{A}_{i,e}^*$ invokes $\text{Dec}_1(x, T', i, \delta, \epsilon)$, and proceeds with the following two cases.
 - If Dec_1 outputs $\perp$ or $\beta \in \{0, 1\}$ such that $\beta \oplus x_i = e$, then $\tilde{A}_{i,e}^*$ continues the rest of the interaction honestly, with the “true” input x and randomness r_1 of Q_1 5 (pretending that it was player 2 that deviated since round j , but its own messages were correctly sent). (pretending that it sent its own messages correctly but was corrupted by the channel).
 - If Dec_1 outputs $\beta \in \{0, 1\}$ such that $\beta \oplus x_i \neq e$, then $\tilde{A}_{i,e}^*$ samples another fresh input-randomness pair (x'', r''_1) , conditioned on the partial transcript T and that $\beta \oplus x''_i = e$, using rejection sampling with a sufficiently large cuts-off parameter $M = 1/\epsilon^5$; if the rejection sampling fails, $\tilde{A}_{i,e}^*$ sets $(x'', r''_1) = (x, r_1)$ (i.e., $\tilde{A}_{i,e}^*$ does not change the input-randomness pair). Then $\tilde{A}_{i,e}^*$ continues the rest of the interaction honestly, with the “new” input x'' and randomness r''_1 of Q_1 (again, pretending that it sent its own messages correctly but was corrupted by the channel).
4. $\tilde{A}_{i,e}$ produces no output.

Claim 44. *For sufficiently large n , there exists $i \in [m(n)]$ and $e \in \{0, 1\}$ such that*

$$\left| \Pr \left[x, y \leftarrow \{0, 1\}^m, o \leftarrow \text{out}_2[\tilde{A}_{i,e}^*(x) \leftrightarrow Q_2(y)] : \text{Match}_{1,i}(o) = 1 \right] - 1/2 \right| \geq 1/32m'(n).$$

Proof. (sketch.) The claim is proved by combining the analysis of Sub-claim 32 in Theorem 27 and the proof of Claim 41 above.

Recall that implicit computation holds with probability at least $1 - 1/n$, and in such case, chunks are well-defined. Let $i^* \in [m]$ be such that the i^* -th chunk has shortest expected length when chunks are well-defined. By an averaging argument, the expected length of the i^* -th chunk is at most $(1/m) \cdot \text{CC}_n(\pi')$. By an identical argument to the analysis of Sub-claim 32, for both $e \in \{0, 1\}$, in the experiment $\{x, y \leftarrow \{0, 1\}^m : \tilde{A}_{i^*,e}^*(x) \leftrightarrow Q_2(y)\}$, (x', y, r'_1, r_2) is uniformly distributed and independent of j . By a Markov argument, with probability at least $1/2$, the i^* -th chunk has length at most $(2/m) \cdot \text{CC}_n(\pi) < \eta(n) \cdot \text{CC}_n(\pi)$. Define **Good** to be the event that (i) chunks are well-defined, (ii) the i^* -th chunk has length at most $(2/m) \cdot \text{CC}_n(\pi)$ and (iii) j equals to the starting round of the i^* -th chunk. Note that when **Good** happens, the i^* -th chunk finished before T' and when $\tilde{A}_{i^*,e}^*$ invokes Dec_1 , it returns a correct bit $\beta = y_i$ with overwhelming probability. Additionally, by definition, (y, T^-) is not (δ, ϵ) -binding for x_i , where T^- is obtained by removing the last message from T . Note that **Good** happens with probability at least $(1 - 1/n) \cdot (1/2) \cdot (1/m'(n)) \cdot (1 - \text{negl}(n)) \geq 1/4m'(n)$.

By a similar argument to the proof of Sub-claim 43, when **Good** happens, Q_2 outputs (a, b) with $a_i \oplus b_i = e$ with high probability. Roughly, the reason is that in this case, with high probability (a) $\tilde{A}_{i^*,e}^*$ can sample a input-randomness pair (x'', r''_1) such that $x''_i \oplus y_i = e$ and (b) the error-resilient property implies that Q_2 outputs (x'', y) . Therefore, for either $e = 0$ or $e = 1$, $\tilde{A}_{i^*,e}^*$ can gain at least $1/16m'(n)$ advantage from the **Good** event. On the other hand, as argued in the proof of Sub-claim 42, the only chance that $\tilde{A}_{i^*,e}^*$ is when Dec_1 returns incorrect answer $\beta = \bar{y}_i$, which happens with negligible probability. Therefore, the overall advantage of $\tilde{A}_{i^*,e}^*$ is at last $1/32m'(n)$. $\square$

References

- [BK12] Zvika Brakerski and Yael Tauman Kalai. Efficient interactive coding against adversarial noise. In *FOCS*, pages 160–166, 2012.
- [Blu86] M. Blum. How to prove a theorem so no one else can claim it. *Proc. of the International Congress of Mathematicians*, pages 1444–1451, 1986.
- [BN13] Zvika Brakerski and Moni Naor. Fast algorithms for interactive coding. In *SODA '13*, 2013. To appear.
- [BR11] Mark Braverman and Anup Rao. Towards coding for maximum errors in interactive communication. In *STOC*, pages 159–166, 2011.
- [Bra12] Mark Braverman. Interactive information complexity. In *STOC*, pages 505–524, 2012.
- [CGGM00] Ran Canetti, Oded Goldreich, Shafi Goldwasser, and Silvio Micali. Resettable zero-knowledge (extended abstract). In *STOC '00*, pages 235–244, 2000.
- [CP11] Kai-Min Chung and Rafael Pass. The randomness complexity of parallel repetition. In *FOCS*, pages 658–667, 2011.
- [DNS04] Cynthia Dwork, Moni Naor, and Amit Sahai. Concurrent zero-knowledge. *J. ACM*, 51(6):851–898, 2004.
- [GMR89] Shafi Goldwasser, Silvio Micali, and Charles Rackoff. The knowledge complexity of interactive proof-systems. *SIAM Journal on Computing*, 18(1):186–208, 1989.
- [GMS11] Ran Gelles, Ankur Moitra, and Amit Sahai. Efficient and explicit coding for interactive communication. In *FOCS*, pages 768–777, 2011.
- [GMW91] Oded Goldreich, Silvio Micali, and Avi Wigderson. Proofs that yield nothing but their validity, or all languages in np have zero-knowledge proof systems. *Journal of the ACM*, 38(1):691–729, 1991.
- [GO94] Oded Goldreich and Yair Oren. Definitions and properties of zero-knowledge proof systems. *J. Cryptology*, 7(1):1–32, 1994.
- [Gol04] Oded Goldreich. *The Foundations of Cryptography - Volume 2*. Cambridge University Press, 2004.
- [GS00] Venkatesan Guruswami and Madhu Sudan. List decoding algorithms for certain concatenated codes. In *STOC*, pages 181–190, 2000.
- [HAM50] R. W. HAMMING. Error detecting and error correcting codes. *BELL SYSTEM TECHNICAL JOURNAL*, 29(2):147–160, 1950.
- [IL89] R. Impagliazzo and M. Luby. One-way functions are essential for complexity based cryptography. In *Proceedings of the 30th Annual IEEE Symposium on Foundations of Computer Science*, pages 230–235, 1989.

- [Jus]
- [Lip94] Richard J. Lipton. A new approach to information theory. In *STACS*, pages 699–708, 1994.
- [MPSW10] Silvio Micali, Chris Peikert, Madhu Sudan, and David A. Wilson. Optimal error correction for computationally bounded noise. *IEEE Transactions on Information Theory*, 56(11):5673–5680, 2010.
- [NY89] Moni Naor and Moti Yung. Universal one-way hash functions and their cryptographic applications. In *STOC '89*, pages 33–43, 1989.
- [Rom90] J. Rompel. One-way functions are necessary and sufficient for secure signatures. In *Proceedings of the 22nd Annual ACM Symposium on Theory of Computing*, pages 387–394, 1990.
- [RS60] I. S. Reed and G. Solomon. Polynomial codes over certain finite fields. *Journal of the Society of Industrial and Applied Mathematics*, 8:300–304, 1960.
- [Sch92] Leonard J. Schulman. Communication on noisy channels: A coding theorem for computation. In *FOCS*, pages 724–733, 1992.
- [Sch93] Leonard J. Schulman. Deterministic coding for interactive communication. In *STOC*, pages 747–756, 1993.
- [Sch96] Leonard J. Schulman. Coding for interactive communication. *IEEE Transactions on Information Theory*, 42(6):1745–1756, 1996.
- [Sha48] Claude E. Shannon. A mathematical theory of communication. *The Bell System Technical Journal*, 27:379–423, 623–656, July, October 1948.
- [Spi96] Daniel A. Spielman. Linear-time encodable and decodable error-correcting codes. *IEEE Transactions on Information Theory*, 42(6):1723–1731, 1996.

Non-Black-Box Simulation from One-Way Functions And Applications to Resettable Security

Kai-Min Chung
Cornell University
chung@cs.cornell.edu

Rafael Pass^{*}
Cornell University
rafael@cs.cornell.edu

Karn Seth
Cornell University
karn@cs.cornell.edu

ABSTRACT

The simulation paradigm, introduced by Goldwasser, Micali and Rackoff, is of fundamental importance to modern cryptography. In a breakthrough work from 2001, Barak (FOCS'01) introduced a novel non-black-box simulation technique. This technique enabled the construction of new cryptographic primitives, such as resettably-sound zero-knowledge arguments, that cannot be proven secure using just black-box simulation techniques. The work of Barak and its follow-ups, however, all require stronger cryptographic hardness assumptions than the minimal assumption of one-way functions.

In this work, we show how to perform non-black-box simulation assuming just the existence of one-way functions. In particular, we demonstrate the existence of a constant-round resettably-sound zero-knowledge argument based only on the existence of one-way functions. Using this technique, we determine necessary and sufficient assumptions for several other notions of resettable security of zero-knowledge proofs. An additional benefit of our approach is that it seemingly makes practical implementations of non-black-box zero-knowledge viable.

Categories and Subject Descriptors

F.1.2 [Theory of Computation]: Interactive and reactive computation

^{*}Pass is supported in part by an Alfred P. Sloan Fellowship, a Microsoft Research Faculty Fellowship, NSF Awards CNS-1217821 and CCF-1214844, NSF CAREER Award CCF-0746990, AFOSR YIP Award FA9550-10-1-0093, and DARPA and AFRL under contract FA8750-11-2-0211. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the Defense Advanced Research Projects Agency or the US Government.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

STOC '13 Palo Alto, California USA

Copyright 20XX ACM X-XXXXX-XX-X/XX/XX ...\$15.00.

General Terms

Theory

Keywords

zero-knowledge, non-black-box simulation, one-way functions, resettable security

1. INTRODUCTION

Zero-knowledge (ZK) interactive protocols [16] are paradoxical constructs that allow one player (called the Prover) to convince another player (called the Verifier) of the validity of a mathematical statement $x \in L$, while providing *zero additional knowledge* to the Verifier. Beyond being fascinating in their own right, ZK proofs have numerous cryptographic applications and are one of the most fundamental cryptographic building blocks.

The zero-knowledge property is formalized using the so-called *simulation paradigm*: for every malicious verifier V^* , we require the existence of a “simulator” S that, given just the input x , can indistinguishably reproduce the view of V^* in an interaction with the honest prover. (We note that the simulation paradigm extends well beyond the notion of zero-knowledge, and is a crucial component of modern definitions of protocol security.) The most typical way of performing such a simulation is using *black-box simulation* [15]: here we exhibit a universal simulator S that, given only black-box access to *any* (efficient) V^* , can reproduce the view of V^* in an interaction with the honest prover. Indeed most zero-knowledge protocols (and more generally protocols for secure computation) are analyzed using black-box simulators. But several limitations of black-box simulators are also known; see e.g. [13, 10, 3, 26].

In a breakthrough result from 2001, Barak [1] demonstrated a new, powerful *non-black-box* simulation technique, and used this technique to construct a constant-round *public-coin* zero-knowledge argument; by the result of [13] such protocols cannot be proved zero-knowledge using just black-box simulation. In the same year, Barak, Goldwasser, Goldreich and Lindell [3] demonstrated that this non-black-box simulation technique could be used to achieve a *new cryptographic primitive* that cannot be proven secure using black-box simulation, namely *resettably-sound zero-knowledge protocols*. In a resettably-sound zero-knowledge protocol, the soundness property is required to hold even if the malicious prover is allowed to “reset” and “restart” the verifier. This model is particularly relevant for cryptographic protocols being executed on embedded devices, such as smart cards. (Since

these devices have neither a built-in power supply, nor a non-volatile rewritable memory, they can be “reset” by simply disconnecting and reconnecting the power supply.) Roughly speaking, the reason why non-black-box simulation is crucial for resettably-sound zero-knowledge protocols is that a black-box simulator has essentially the same “powers” as a malicious resetting prover (i.e., it can only reset and restart the verifier); from this observation it follows that, unless $L \in \text{BPP}$, a “good” simulator can be as a successful cheating prover. Since these results, non-black-box simulation techniques have found applications in various other contexts (see e.g. [2, 24, 25, 11]).

One important limitation of the non-black-box simulation technique of Barak [1] (also present in its follow-up works) is that the technique requires stronger assumptions than those typically needed for constructing zero-knowledge protocols. In particular, the protocol of Barak (using the refinement in [2]) relies on the existence of families of collision-resistant hash functions (CRH), and as a consequence, such hash functions are needed in the above applications too.¹ In contrast, for “plain” zero-knowledge (i.e., without, for instance, resettable soundness) one-way functions are both sufficient and (essentially) necessary [14, 17, 23], leaving open the following question, which is the focus of this work.

Do one-way functions suffice for performing non-black-box simulation (for primitives that cannot be proven secure using black-box simulation techniques)?

A very recent elegant work by Bitansky and Paneth [6] takes us a step closer to answering this question. They present a resettably-sound zero-knowledge argument without relying on hash functions; instead, they rely on the existence of an *oblivious transfer (OT) protocol*. Although, the existence of an OT protocol is seemingly a more “complex” assumption than the existence of CRHs,² it is not known whether the existence of an OT protocol implies the existence of CRH (or vice versa). More important, to achieve this result, Bitansky and Paneth devise a quite different method for performing non-black-box simulation.

1.1 Our Result

In this work, we answer the above question in the affirmative. We show that for the case of resettably-sound zero-knowledge, the existence of one-way functions suffices.

THEOREM 1 (MAIN THEOREM). *Assume the existence of one-way function. Then there exists a constant-round resettably-sound zero-knowledge argument for all of NP.*

Interestingly, our protocol is quite close in spirit to Barak’s original protocol, while dispensing of the need for collision-resistant hash functions.

By relying on the above main theorem, we establish several other results on resettable security: Assuming one-way functions, all of NP has

- a constant-round resettably-witness-indistinguishable argument of knowledge;
- a $\tilde{O}(\log n)$ -round resettable-zero-knowledge argument of knowledge.

(Roughly speaking, in a resettably-witness indistinguishable (resp., zero-knowledge) argument, the witness indistinguishability (resp., zero-knowledge) property is required to hold also in the presence of a resetting verifier.) For the above-mentioned primitives, previous results required additional cryptographic assumptions (the existence of collision-resistant hash-functions or oblivious transfer protocols). We additionally show how to eliminate the need for CRHs in the construction of [11] of a simultaneously resettable zero-knowledge argument for NP—simultaneous resettability here means that security (both zero-knowledge and soundness) holds even with respect to resetting attackers.

We emphasize that for all the above results, the use of non-black-box techniques are inherent. Our results lead to improvements also for cases when black-box simulation can be used: prior to our results, resettable zero-knowledge arguments (without the argument of knowledge property) were known only based on the existence of CRHs, but these protocols were actually proven secure using black-box simulation. As mentioned above, we are able to establish even the stronger notion of a resettable zero-knowledge argument of knowledge assuming only one-way functions.

1.2 Our Techniques

To explain our techniques, let us start by very briefly recalling the idea behind Barak’s constant-round public-coin protocol; we will then explain how this protocol is used to get a resettably-sound zero-knowledge protocol. The protocol relies on the existence of a family of collision-resistant hash function $h : \{0, 1\}^* \rightarrow \{0, 1\}^n$; note that any such family of collision-resistant hash functions can be implemented from a family of collision-resistant hash functions mapping n -bit string into $n/2$ -bit strings using *tree hashing* [19].

Roughly speaking, on common input 1^n and $x \in \{0, 1\}^{\text{poly}(n)}$, the Prover P and Verifier V , proceed in two stages. In Stage 1, V starts by selecting a function h from a family of collision-resistant hash function and sends it to P ; P next sends a commitment $c = \text{Com}(0^n)$ of length n , and finally, V next sends a “challenge” $r \in \{0, 1\}^{2^n}$. In Stage 2, P shows (using a witness indistinguishable argument of knowledge) that either x is true, or that c is a commitment to a “hash” (using h) of a program M (i.e., $c = \text{Com}(h(M))$) such that $M(c) = r$.

Roughly speaking, soundness follows from the fact that even if a malicious prover P^* tries to commit to (the hash of) some program M (instead of committing to 0^n), with high probability, the a string r sent by V will be different from $M(c)$ (since r is chosen independently of c). To prove ZK, consider the non-black-box simulator S that commits to a hash of the code of the malicious verifier V^* ; note that, by definition, it thus holds that $M(c) = r$, and the simulator can use c as a “fake” witness in the final proof. To formalize this approach, the witness indistinguishable argument in Stage 2 must actually be a witness indistinguishable *universal argument* (WIUARG) [20, 2] since the statement that c is a commitment to a program M of arbitrary polynomial-size, and that proving $M(c) = r$ within some arbitrary polynomial time, is not in NP. WIUARG are known based on the

¹The original protocol of Barak relies on a very slightly super-polynomially hard collision-resistant hash function; the need for super-polynomial hardness was removed in [2].

²Most candidate constructions of OT protocols rely on “structured”, number-theoretic or lattice-based, assumptions. Additionally, all these assumptions are known to imply also the existence of collision-resistance hash function (but the converse is not true).

existence of CRH and those protocols are constant-round public-coin; as a result, the whole protocol is constant-round and public-coin.

Finally, Barak et al. [3] show that any constant-round public-coin zero-knowledge argument of knowledge can be transformed into a resettable-sound zero-knowledge argument, by simply having the verifier generate its (random) message by applying a pseudorandom function to the current partial transcript.³

Why hash functions are needed Note that hash functions are needed in two locations in Barak’s protocol. First, since there is no *a-priori* polynomial upper-bound of the length of the code of V^* , we require the simulator to commit to the hash of the code of V^* . Secondly, since there is no *a-priori* polynomial upper-bound on the running-time of V^* , we require the use of universal arguments (and such constructions are only known based on the existence of collision-resistant hash functions).

Using signature schemes instead of CRHs Our main idea is noticing that digital signature schemes—which can be constructed based on one-way functions—share many of the desirable properties of CRHs. In particular, we will show how to appropriately instantiate (a variant of) Barak’s protocol using signature schemes instead of using CRHs. Recall that “fixed-length” signature schemes, that allow signing messages of arbitrary polynomial-length (e.g. length $2n$) using a length n signature, are known based on just one-way functions [27]. In fact, based on the same assumption, *strong* fixed-length signature schemes are known: in a strong signature scheme no polynomial time attacker can obtain a *new* signature even for messages that it has seen a signature on [12]. We observe that such signature scheme share a lot of properties with CRHs. First of all, they are compressing. More importantly, we observe that by the unforgeability requirement of strong signatures, no attacker can find a single valid signature σ for two distinct messages m, m' —that is, signatures satisfy a collision-resistance property. Additionally, by using an appropriate analog of tree hashing, a *signature tree* could be used to compress arbitrary length messages into a signature of length n .

So, can we just replace the CRHs in Barak’s protocol with strong, fixed-length, signature schemes? The problem with naively implementing this idea is that the collision-resistance property of strong signature schemes only holds against an attacker that does *not* know the secret key. On the other hand, to generate signatures, knowledge of the secret key is needed. In our application, the simulator—acting as a prover—needs to be able to generate signature (in order to “hash down” the program, and in the universal argument) but at the same time, we need to ensure collision-resistance against cheating provers. So if we let the prover generate the signature keys, simulation is easy, but soundness no longer holds, whereas if we let the verifier generate the signature keys and only sends the verification key to the prover, then soundness holds, but it is no longer clear how to perform a simulation. We resolve this issue by using a “hybrid approach”: we let the verifier generate the signature keys, but gives the prover access to a *single* signing query. More pre-

cisely, in an initial stage of the protocol, the verifier generates a signature key-pair sk, vk and send only the verification key vk to the prover. Next, in a “signature slot”, the prover sends a message m to the verifier, and the verifier computes and returns a valid signature σ of m (using sk). (We note that such a signature slot previously used by [18] in a quite different context, but as we shall see shortly, some of their techniques will be useful also to us.) Finally, the protocol proceeds essentially as in Barak’s protocol, but where the CRH is replaced using the signature scheme. Implementing this is somewhat subtle: First, the statement proved in the WIUARG in Barak’s protocol considers the hash function h (e.g., prover needs to prove statements of the type $h(m) = q$). In our approach since “hashing” has been replaced by “signing”, this would require the honest prover to prove things related to the secret-key (e.g., $\text{Sign}_{sk}(m) = q$), but the honest prover does not know sk . This issue is easily resolved by instead of letting the prover show that signatures used (as “hashes”) verify—i.e., that $\text{Ver}_{vk}(m) = q$. Another issue is that in Barak’s protocol, the honest prover actually needs to perform hashes to complete the WIUARG. We resolve this second issue by relying on an instantiation of Barak’s protocol due to Pass and Rosen [25], which relies on a special-purpose WIUARG, in which the honest prover never needs to perform any hashing.⁴ Now completeness of this protocol follows in exactly the same way as in [1, 25].

For soundness, note that since the prover does not get to see sk , soundness follows in a similar way to Barak’s protocol. In fact, if the signature scheme used satisfies strong unforgeability, then the signature trees are collision-resistant with respect to attackers that get vk and *have access to a signing oracle*, and collision-resistance of the signature tree is the only property needed to prove soundness as in Barak’s protocol. (Note that we here only require collision-resistance with respect to attackers that get a single query to a signing oracle, but the more general result will be useful when we consider resettable-soundness.)

Let us turn to zero-knowledge. At first sight, it seems that we still have an issue. The prover just gets a single signature, but to complete the simulation, the simulator needs an a-priori unbounded polynomial number of signatures (to e.g., “hash down” a program of a-priori unbounded polynomial-size.⁵) Note, however, that the simulator can always *rewind* the verifier to get as many signatures as it wants and can thus complete the simulation in a similar way to the one used in Barak’s protocol. This approach doesn’t quite work: the malicious verifier V^* may not always agree to sign every message requested by the simulator; we deal with this issue in the same way as in [18], rather than having the simulator send the messages it wants to be signed in the clear, it simply sends a commitment to them. To make use of such a simulator strategy, we appropriately modify the notion of a signature tree to consist of signatures *of commitments* to signatures etc; we refer to this type of a signature tree as a “sig-com” tree.

So, we now have a zero-knowledge protocol that is based on one-way functions (and is constant-round). But it is no longer public-coin!

Nonetheless, let us still apply the PRF transformation of

³Strictly speaking, Barak’s protocol is not a argument of knowledge, but rather a “weak” argument of knowledge (see [2, 3] for more details), but the transformation of [3] applies also to such protocol.

⁴In fact, an early version of Barak’s protocol also had this property.

⁵Also in the implementation of the WIUARG, an a-priori unbounded number of “hashes” are needed.

[3] to the protocol (i.e., we have the verifier generate its random coins in each round by applying a PRF to the current partial transcript). Clearly, the protocol is still zero-knowledge (since we only modified the verifier strategy). As it turns out, the resulting protocol is actually also resettably-sound: note that, except for the signature slot added in the beginning of the protocol, the protocol still is public-coin, and the same argument as in [13, 3] can be used to show that in the public-coin part of the protocol, rewindings do not “help” a resetting cheating prover. So, in essence, the only “advantages” a resetting prover gets is that it may rewind the signature slot, and thus get an arbitrary polynomial number of signatures on messages of its choice. But, as noted above, signature trees are collision-resistant even with respect to an attacker that gets an arbitrary polynomial number of queries to a signing oracle and thus resettable-soundness follows in exactly the same way as the (non-resetting) soundness property.

Beyond resettably-sound zero-knowledge For the applications of a) a constant-round resettably witness-indistinguishable argument of knowledge, and b) $\tilde{O}(\log n)$ -round resettable-zero-knowledge argument of knowledge for NP, we simply plug in our resettably-sound zero-knowledge argument of knowledge into the protocols of [8, 3] with some minor modifications.

To achieve simultaneously resettable zero-knowledge, we instead instantiate the protocol of Deng, Goyal and Sahai [11] with signature trees, in exactly the same way as Barak’s protocol. Resettably-soundness follows exactly as in [11], relying on the collision-resistance property of signature trees. Resettably-zero-knowledge is more tricky though: [11] provides an intricate simulation strategy that combines black-box simulation, using rewinding, and non-black-box simulation (as in [1]). Roughly speaking, the protocol consists of polynomially many “rewinding slots” (say $2n^2$), and for each session started by the resetting verifier, the simulator of [11] rewinds a polynomial fraction (say $2n$) of them *twice*. Their argument shows that for each such slot, the rewinding “succeeds” with probability close to $1/2$ and the slot gets “solved”; as a consequence, except with negligible probability, for each session, there exists some slot that is “solved” and this suffices for simulating the session. In our instantiation of their protocol, rewinding a slot just once does not suffice to “solve” the session (and complete the simulation of that session). Rather we need polynomially many, say $g(n) = \text{poly}(|V^*|)$ where $|V^*|$ is the size of the verifier (including its auxiliary input), successful rewindings (in order to rewind the signature slot sufficiently many times to provide the signature trees). We deal with this issue in a straight-forward way: we use exactly the same rewinding strategy as in [11] but instead rewind each slot (that was being rewound once in [11]) $3g(n)$ times. It follows using a slight generalization of the argument in [11] that each slot that is rewound is successfully solved with probability close to $1/2$, and the rest of the simulation argument continues in identically the same way as [11]. Additionally, rewinding polynomially many times (as opposed to twice) only increases the running-time by a polynomial factor (the technical reason for this is that the [11] simulator only performs a constant-number of recursive rewindings).

A PCP-free construction Just as the construction of Barak’s protocol, our constructions rely on universal ar-

guments, which in turn rely on Probabilistically Checkable Proofs (PCPs). Intriguingly, the approach of Bitansky and Paneth [6] does not rely on PCPs; on the other hand, it relies on some other quite heavy machinery: “unobfuscatable functions” [4] and general secure two-party computation [14].

As we now sketch, our approach can be instantiated without the use of PCPs, and without introducing any other machinery. (Indeed, although we have not verified the details, it would seem that a practical implementation of our protocol can be given by relying on efficient signatures and zero-knowledge proofs of committed signatures, as in e.g., [7].) Recall that in Barak’s protocol the universal argument is used to prove a statement of the form c is a commitment to a hash of a program M such that $M(c) = r$. Also recall that (in the [25] variant of [1]) the honest prover never needs to engage in the universal argument, it is only the simulator that needs to prove the above statement. Rather than providing a universal argument, we let the simulator prove $M(c) = r$ in a *piecemeal* fashion, by making the verifier certify every step of the computation of M . This strategy is very similar to one employed in the “impossibility of instantiating random oracles” result of [9]⁶ (On a high-level, this type of piecemeal decomposition is also somewhat similar to what is done in the impossibility result of [4]; as such our approach brings out the connection between the techniques from [1] and [6].) More precisely, in the actual protocol, the verifier generates a key-pair vk', sk' for a signature scheme and sends vk' to the prover. The prover then provides the verifier with a commitment c_1 to a tree hash⁷ of a *current-configuration*, a commitment c_2 to a tree-hash of a *next-configuration*, and a witness indistinguishable argument of knowledge that either a) $x \in L$ or b) *next-configuration* is a starting configuration or c) performing *one step* of computation given *current-configuration* leads to *next-configuration*, and *current-configuration* has been previously signed. (Note that since we use tree-hashing, verification of condition b) and c) can both be done in time polylogarithmic in the length of the configurations). If the argument of knowledge is accepting, the verifier signs c_2 . Roughly speaking, the above “slot” makes it possible for the simulator to get a signature on (commitments to signature-trees of) s_0 , where s_0 is the initial configuration of $M(\sigma)$ (using condition b), and next by rewinding the verifier sufficiently many times to get signatures on later configurations s_t in the computation of $M(\sigma)$ (using condition c). Thus, finally, the simulator can get a signature on s_T where s_T is the terminating configuration of the computation of $M(\sigma)$. The simulator can then use this signature to convince the verifier that $M(c) = r$ where M is the program committed to in c . A complete formalization appears in the full version of this paper.

1.3 Subsequent Work

A very recent elegant work by Bitansky and Paneth [5] (developed subsequently to our results) shows an alternative approach for obtaining resettably-sound arguments (and re-

⁶They key difference is that construction of [9] only considers an honest “non-aborting” verifier, whereas we need to deal with also malicious “aborting” verifiers. This issue is analogous to why we rely on “sig-com” trees (consisting of signatures of commitments to signatures etc.) as opposed to “plain” signature trees.

⁷We may also instantiate tree-hashing with signature-trees to get an implementation based on one-way functions.

lated primitives) from one-way functions, by first constructing functions that are “approximately” unobfuscatable, and relying on the connection between resetttable-soundness and unobfuscatable functions from [6].

1.4 Outline

In Section 3 we provide formal definitions of signature trees, and provide collision-resistance properties of such trees. To formalize our construction of resetttable-sound zero-knowledge in a modular way, in Section 4, we first consider an “oracle-aided” model, in which players have access to a signing oracle. We first show that the universal argument construction of Barak and Goldreich [2] can be instantiated using one-way functions in such an oracle-aided model, by replacing “hashing” with “signing”. We next show how to instantiate Pass and Rosen’s [25] variant of Barak protocol in the same way (by relying on the oracle-aided construction of universal arguments). This leads to a constant-round oracle-aided public-coin zero-knowledge argument of knowledge, satisfying a key property: the honest prover never needs to access the oracle. We may next apply the transformation of [3] to this protocol to obtain an oracle-aided resetttable-sound zero-knowledge argument of knowledge satisfying the same key property (the results of [3] relativize and thus we can directly apply them also to oracle-aided protocols).

In Section 5, we present a general transformation, transforming any oracle-aided resetttable-sound zero-knowledge argument (of knowledge) satisfying the above key property, into a resetttable-sound zero-knowledge argument (of knowledge) in the “plain” model (i.e. without any oracle): the transformation simply consists of adding a signature slot at the beginning of the protocol. Taken together with our result in Section 4, this yields a constant-round resetttable-sound zero-knowledge argument of knowledge for NP based on one-way functions.

Applications (such as simultaneously resetttable zero-knowledge) are presented in the full version of the paper.

2. DEFINITIONS

We assume familiarity with interactive arguments, arguments of knowledge and witness indistinguishability; see the full version for more details.

We start by recalling the definition of zero knowledge from [16].

DEFINITION 1 (ZERO-KNOWLEDGE [16]). *An interactive protocol (P, V) for language L is zero-knowledge if for every PPT adversarial verifier V^* , there exists a PPT simulator S such that the following ensembles are computationally indistinguishable over $x \in L$:*

$$\{\text{View}_{V^*} \langle P, V^*(z) \rangle(x)\}_{x \in L, z \in \{0,1\}^*} \approx \{S(x, z)\}_{x \in L, z \in \{0,1\}^*}$$

Let us recall the definition of resetttable soundness due to [3].

DEFINITION 2 (RESETTABLE-SOUND ARGUMENTS [3]). *A resetting attack of a cheating prover P^* on a resetttable verifier V is defined by the following two-step random process, indexed by a security parameter n .*

1. *Uniformly select and fix $t = \text{poly}(n)$ random-tapes, denoted $r_1, \dots, r_t$, for V , resulting in deterministic strate-*

gies $V^{(j)}(x) = V_{x, r_j}$ defined by $V_{x, r_j}(\alpha) = V(x, r_j, \alpha)$,⁸ where $x \in \{0,1\}^n$ and $j \in [t]$. Each $V^{(j)}(x)$ is called an incarnation of V .

2. *On input 1^n , machine P^* is allowed to initiate $\text{poly}(n)$ -many interactions with the $V^{(j)}(x)$ ’s. The activity of P^* proceeds in rounds. In each round P^* chooses $x \in \{0,1\}^n$ and $j \in [t]$, thus defining $V^{(j)}(x)$, and conducts a complete session with it.*

Let (P, V) be an interactive argument for a language L . We say that (P, V) is a resetttable-sound argument for L if the following condition holds:

- *Resetttable-soundness: For every polynomial-size resetting attack, the probability that in some session the corresponding $V^{(j)}(x)$ has accepted and $x \notin L$ is negligible.*

We will also consider a slight weakening of the notion of resetttable soundness, where the statement to be proven is fixed, and the verifier uses a single random tape (that is, the prover cannot start many independent instances of the verifier).

DEFINITION 3 (FIXED-INPUT R.S. ARGUMENTS [?]). *An interactive argument (P, V) for a NP language L with witness relation R_L is fixed-input resetttable-sound if it satisfies the following property: For all non-uniform polynomial-time adversarial prover P^* , there exists a negligible function $\mu(\cdot)$ such that for every all $x \notin L$,*

$$\Pr[R \leftarrow \{0,1\}^\infty; (P^* V_R(x, \text{pp}), V_R)(x) = 1] \leq \mu(|x|)$$

As the following claim (which essentially follows from techniques in [3]) shows, any zero-knowledge argument of knowledge satisfying the weaker notion can be transformed into one that satisfies the stronger one, while preserving zero-knowledge (or any other secrecy property against malicious verifiers).

CLAIM 2. *Let (P, V) be a fixed-input resetttable sound zero-knowledge (resp. witness indistinguishable) argument of knowledge for a language $L \in \text{NP}$. Then there exists a protocol (P', V') that is a (full-fledged) resetttable-sound zero-knowledge (resp. witness indistinguishable) argument of knowledge for L .*

The proof is found in the full version.

3. SIGNATURE TREES

In this section, we define an analogue of Merkle-hash trees using signature schemes. Towards this, we will rely on the existence of strong, fixed-length, deterministic secure signature schemes. Recall that in a strong signature scheme, no polynomial-time attacker having oracle access to a signing oracle can produce a valid message-signature pair, unless it has received this pair from the signing oracle. The signature scheme being fixed-length means that signatures of arbitrary (polynomial-length) messages are of some fixed polynomial length. Deterministic signatures do not use any randomness in the signing process once the signing key has been chosen. In particular, once a signing key has been chosen, a message m will always be signed in the same way.

⁸Here, $V(x, r, \alpha)$ denotes the message sent by the strategy V on common input x , random-tape r , after seeing the message-sequence α .

DEFINITION 4 (STRONG SIGNATURES). A strong, length- ℓ , signature scheme SIG is a triple $(\text{Gen}, \text{Sign}, \text{Ver})$ of PPT algorithms, such that

1. for all $n \in \mathbb{N}, m \in \{0, 1\}^*$,

$$\Pr[(\text{sk}, \text{vk}) \leftarrow \text{Gen}(1^n), \sigma \leftarrow \text{Sign}_{\text{sk}}(m); \\ \text{Ver}_{\text{vk}}(m, \sigma) = 1 \wedge |\sigma| \leq \ell(n)] = 1$$

2. for every non-uniform PPT adversary A , there exists a negligible function $\mu(\cdot)$ such that

$$\Pr[(\text{sk}, \text{vk}) \leftarrow \text{Gen}(1^n), (m, \sigma) \leftarrow A^{\text{Sign}_{\text{sk}}(\cdot)}(1^n); \\ \text{Ver}_{\text{vk}}(m, \sigma) = 1 \wedge (m, \sigma) \notin L] \leq \mu(n),$$

where L denotes the list of query-answer pair of A 's query to its oracle.

Strong, length- ℓ , deterministic signature schemes with $\ell(n) = n$ are known based on the existence of OWFs; see [22, 27, 12] for further details. In the rest of this paper, whenever we refer to signature schemes, we always means strong, length- n deterministic signature schemes.

Let us first note that strong signatures satisfy a “collision-resistance” property.

CLAIM 3. Let $\text{SIG} = (\text{Gen}, \text{Sign}, \text{Ver})$ be a strong (length- n) signature scheme. Then, for all non-uniform PPT adversaries A , there exists a negligible function $\mu(\cdot)$ such that for every $n \in \mathbb{N}$,

$$\Pr[(\text{sk}, \text{vk}) \leftarrow \text{Gen}(1^n), (m_1, m_2, \sigma) \leftarrow A^{\text{Sign}_{\text{sk}}(\cdot)}(1^n, \text{vk}); \\ \text{Ver}_{\text{vk}}(m_1, \sigma) = \text{Ver}_{\text{vk}}(m_2, \sigma) = 1] \leq \mu(n)$$

PROOF. Assume for contradiction that there exists some non-uniform polynomial-time A such that A breaks “collision-resistance” property of SIG with probability $\frac{1}{p(n)}$ for infinitely many $n \in \mathbb{N}$, where p is a polynomial. We show that A can be used to break the strong unforgeability property of SIG . More precisely, note that if A outputs a valid signatures $(m_1, \sigma), (m_2, \sigma)$ without querying the signing oracle with m_1 and m_2 and receiving σ as a response to both queries, then A already breaks the security of the signature scheme. Thus w.l.o.g. we may assume A queries both m_1 and m_2 to the signing oracle and receives σ as a response. We then simulate A , recording the previous messages queried to the oracle along with the responses. At each point during the execution of A , before forwarding the next query m to the oracle, we test if any of the previously received signatures are valid signatures for m . If so, we output m together with such a signature σ . Notice that if A always queries m_1 and m_2 and receives σ as a response, then we will intercept whichever of the two A queries second. Thus, for infinitely many n , with probability $\geq \frac{1}{p(n)}$, we forge a signature σ for some m before ever querying the signing oracle and receiving σ as a response. $\square$

We now define an analog of Merkle-hash tree which we call *signature trees* and show that they also satisfy a collision-resistant property. We index each node of a complete binary tree Γ of depth d by a binary string of length at most d , where the root is indexed by the empty string λ , and each node indexed by γ has left and right children indexed $\gamma 0$ and $\gamma 1$, respectively.

DEFINITION 5 (SIGNATURE TREES). Let $\text{SIG} = (\text{Gen}, \text{Sign}, \text{Ver})$ be a strong, length- n signature scheme. Let (sk, vk) be a key-pair of SIG , and s be a string of length 2^d . A signature tree of the string s w.r.t. (sk, vk) is a complete binary tree of depth d , defined as follows.

- A leaf l_γ indexed by $\gamma \in \{0, 1\}^d$ is set as the bit at position γ in s .
- An internal node l_γ indexed by $\gamma \in \bigcup_{i=0}^{d-1} \{0, 1\}^i$ satisfies that $\text{Ver}_{\text{vk}}((l_{\gamma 0}, l_{\gamma 1}), l_\gamma) = 1$.

Note that to verify whether a Γ is a valid signature-tree of a string s w.r.t. the signature scheme SIG and the key-pair (sk, vk) knowledge of the secret key sk is not needed. However, to create a signature-tree for a string s , the secret key sk is needed.

The following notion of a signature path is the natural analog of an authentication path in a Merkle-tree.

DEFINITION 6 (SIGNATURE PATH). A signature path w.r.t. SIG, vk and the root l_λ for the bit b at leaf $\gamma \in \{0, 1\}^d$ is a vector $\vec{\rho} = ((l_0, l_1), ((l_{\gamma \leq 0}, l_{\gamma \leq 1}), \dots, (l_{\gamma \leq d-1}, l_{\gamma \leq d-1})))$ such that for every $i \in \{0, \dots, d-1\}$, $\text{Ver}_{\text{vk}}((l_{\gamma \leq i}, l_{\gamma \leq i+1}), l_{\gamma \leq i}) = 1$. Let $\text{PATH}^{\text{SIG}}(\vec{\rho}, b, \gamma, l_\lambda, \text{vk}) = 1$ if ρ is a signature path w.r.t. $\text{SIG}, \text{vk}, l_\lambda$ for b at γ .

The following claim states that signature trees also satisfy an appropriate collision-resistance property: no non-uniform PPT attacker having oracle access to a signing oracle can output a root and valid signature paths for both 0 and 1 at some leaf γ .

CLAIM 4. Let $\text{SIG} = (\text{Gen}, \text{Sign}, \text{Ver})$ be a strong, length- n , signature scheme. Then, for every non-uniform PPT adversary A , there exists a negligible function μ such that:

$$\Pr[(\text{sk}, \text{vk}) \leftarrow \text{Gen}(1^n), (\vec{\rho}_0, \vec{\rho}_1, \gamma, l_\lambda) \leftarrow A^{\text{Sign}_{\text{sk}}(\cdot)}(1^n, \text{vk}); \\ \forall b \in \{0, 1\} \text{ PATH}^{\text{SIG}}(\vec{\rho}_b, b, \gamma, l_\lambda, \text{vk}) = 1] \leq \mu(n)$$

PROOF. The claim directly follows from Claim 3 since any two valid signature-paths with the same root but different leaf value must contain a collision for the underlying signature scheme. $\square$

3.1 Sig-Com Schemes

For the technical reason explained in the introduction, we will rely on variant of signature trees consisting of alternating signatures and commitments. To formalize this, we consider the notion of a “sig-com” scheme:

DEFINITION 7 (SIG-COM SCHEMES). Let $\text{SIG} = (\text{Gen}, \text{Sign}, \text{Ver})$ be a strong, length- n , signature scheme, and let Com be a non-interactive commitment schemes. Define $\text{SIG}' = (\text{Gen}', \text{Sign}', \text{Ver}')$ to be a triple of PPT machines defined as follows:

- $\text{Gen}' = \text{Gen}$.
- $\text{Sign}'_{\text{sk}}(m)$: compute a commitment $c = \text{Com}(m; \tau)$ using a uniformly selected τ , and let $\sigma = \text{Sign}_{\text{sk}}(c)$; output (σ, τ)
- $\text{Ver}'_{\text{vk}}(m, \sigma, \tau)$: Output 1 iff $\text{Ver}_{\text{vk}}(\text{Com}(m, \tau), \sigma) = 1$.

We call SIG' the Sig-Com Scheme corresponding to SIG and Com .

Note that the above definition of a sig-com scheme assumes that **Com** is a non-interactive commitment scheme. This is only for convenience of notation; the above definition, as well as all subsequent results directly apply also to 2-round commitment (i.e., families of non-interactive commitment schemes), as in [21], by simply adding the first message q to the verification key of the sig-com scheme.

Sig-com schemes also satisfy a collision-resistant property:

CLAIM 5 (COLLISION RESISTANCE OF SIG-COMS). *Let $\text{SIG} = (\text{Gen}, \text{Sign}, \text{Ver})$ be a strong, length- n signature scheme, Com be non-interactive commitment scheme, and let $\text{SIG}' = (\text{Gen}', \text{Sign}', \text{Ver}')$ be a sig-com scheme corresponding to SIG and Com . Then, for any non-uniform PPT adversary A , there exists a negligible function μ such that for all $n \in \mathbb{N}$:*

$$\Pr[(\text{sk}, \text{vk}) \leftarrow \text{Gen}(1^n), (\sigma, m_1, m_2, \tau_1, \tau_2) \leftarrow A^{\text{Sign}_{\text{sk}}(\cdot)}(1^n, \text{vk}); m_1 \neq m_2, \text{Ver}'_{\text{vk}}(m_1, \sigma, \tau_1) = \text{Ver}'_{\text{vk}}(m_2, \sigma, \tau_2) = 1] \leq \mu(n)$$

PROOF. Note that by the binding property of Com , no non-uniform PPT can output a valid commitment c to two different messages $m_1 \neq m_2$ except with negligible probability. Thus, except with negligible probability, a successful non-uniform PPT attacker must output a signature for two different commitments $c_1 \neq c_2$, violating collision-resistance of SIG (i.e., Claim 3). $\square$

Note that in Claim 5, the attacker gets oracle access to a signature oracle (for SIG) as opposed to a sig-com oracle.

We may now define sig-com trees and sig-com path in an analogous way to (plain) signature trees and paths.

DEFINITION 8 (SIG-COM TREES). *Let $\text{SIG} = (\text{Gen}, \text{Sign}, \text{Ver})$ be a strong, length- n signature scheme, let Com be a non-interactive commitment and let $\text{SIG}' = (\text{Gen}', \text{Sign}', \text{Ver}')$ be the sig-com scheme corresponding to SIG and Com . Let (sk, vk) be a key-pair of SIG' , and s be a string of length 2^d . A signature tree of the string s w.r.t. (sk, vk) is a complete binary tree of depth d , defined as follows.*

- A leaf l_γ indexed by $\gamma \in \{0, 1\}^d$ is set as the bit at position γ in s .
- An internal node l_γ indexed by $\gamma \in \bigcup_{i=0}^{d-1} \{0, 1\}^i$ satisfies that there exists some τ_γ such that $\text{Ver}'_{\text{vk}}((l_{\gamma 0}, l_{\gamma 1}), l_\gamma, \tau_\gamma) = 1$.

DEFINITION 9 (SIG-COM PATH). *Let $\text{SIG}' = (\text{Gen}', \text{Sign}', \text{Ver}')$ be a sig-com scheme. A sig-com path w.r.t. SIG', vk and the root l_λ for the bit b at leaf $\gamma \in \{0, 1\}^d$ is a vector $\vec{\rho} = ((l_0, l_1, \tau_\lambda), ((l_{\gamma \leq 1 0}, l_{\gamma \leq 1 1}, \tau_{\gamma \leq 1}), \dots, (l_{\gamma \leq d-1 0}, l_{\gamma \leq d-1 1}, \tau_{\gamma \leq d-1}))$ such that for every $i \in \{0, \dots, d-1\}$, $\text{Ver}'_{\text{vk}}((l_{\gamma \leq i 0}, l_{\gamma \leq i 1}, \tau_{\gamma \leq i})) = 1$. Let $\text{PATH}^{\text{SIG}'}(\vec{\rho}, b, \gamma, l_\lambda, \text{vk}) = 1$ if $\vec{\rho}$ is a signature path w.r.t. $\text{SIG}', \text{vk}, l_\lambda$ for b at γ .*

Sig-com trees also satisfy a collision-resistance property:

CLAIM 6. *Let $\text{SIG} = (\text{Gen}, \text{Sign}, \text{Ver})$ be a strong, length- n signature scheme, let Com be a non-interactive commitment and let $\text{SIG}' = (\text{Gen}', \text{Sign}', \text{Ver}')$ be the sig-com scheme corresponding to SIG and Com . Then, for every non-uniform PPT adversary A , there exists a negligible function μ such that:*

$$\Pr[(\text{sk}, \text{vk}) \leftarrow \text{Gen}(1^n), (\vec{\rho}_0, \vec{\rho}_1, \gamma, l_\lambda) \leftarrow A^{\text{Sign}_{\text{sk}}(\cdot)}(1^n, \text{vk}); \forall b \in \{0, 1\} \text{ PATH}^{\text{SIG}'}(\vec{\rho}_b, b, \gamma, l_\lambda, \text{vk}) = 1] \leq \mu(n)$$

PROOF. As in Claim 4, the claim follows directly from Claim 5 since any two valid sig-com paths with the same root but different leaf values must contain a collision for the underlying sig-com scheme. $\square$

Canonical Sig-com Schemes Throughout the rest of the paper, we consider sig-com schemes SIG' and sig-com trees corresponding to a strong, length- n deterministic signature scheme SIG and a non-interactive commitment Com that generates n^2 bits long commitments to $2n$ bits strings. Thus, each node of the sig-com tree is an n -bit signature of an n^2 bits commitment of the two signatures of the children nodes. Hereafter, we refer to such a SIG' as a **canonical sig-com scheme**.

4. ORACLE-AIDED RS-ZK

In this section we show how to construct a resettably-sound ZK argument in an oracle-aided model where prover and verifier additionally have access to a public parameter generated prior to the interaction (in our protocol, this will be the verification key for a signature scheme), and, further the prover has access to an oracle, also generated prior to the interaction (in our protocol, this will be a signature/sig-com oracle).

More formally, let $\mathcal{O}$ be a probabilistic algorithm that on input a security parameter n , outputs a polynomial-length (in n) public-parameter pp , as well as the description of an oracle O . The oracle-aided execution of an interactive protocol with common input x between a prover P with auxiliary input y and a verifier V consist of first generating $\text{pp}, O \leftarrow \mathcal{O}(1^{|x|})$ and then letting $P^O(x, y, \text{pp})$ interact with $V(x, \text{pp})$.

DEFINITION 10 (ORACLE-AIDED INTERACTIVE ARG). *A pair of oracle algorithms (P, V) is an $\mathcal{O}$ -oracle aided argument for a NP language L with witness relation R_L if it satisfies the following properties:*

- **Completeness:** *There exists a negligible function $\mu(\cdot)$, such that for all $x \in L$, if $w \in R_L(x)$,*

$$\Pr[\text{pp}, O \leftarrow \mathcal{O}(1^{|x|}); (P^O(w), V)(x, \text{pp}) = 1] = 1 - \mu(|x|)$$

- **Soundness:** *For all non-uniform polynomial-time adversarial prover P^* , there exists a negligible function $\mu(\cdot)$ such that for every all $x \notin L$,*

$$\Pr[\text{pp}, O \leftarrow \mathcal{O}(1^{|x|}); (P^{*O}, V)(x, \text{pp}) = 1] \leq \mu(|x|)$$

We will also define an $\mathcal{O}$ -oracle aided version of arguments of knowledge, essentially analogously to their canonical definitions, except with a setup phase in which pp and O are generated and made available to the players. The formal definitions can be found in the full version of this paper.

Towards our goal of constructing of oracle-aided resettably-sound zero-knowledge, we now define and construct an oracle-aided version of universal arguments.

4.1 Oracle-aided Universal Arguments

Universal arguments (introduced in [2] and closely related to CS-proofs [20]) are used in order to provide “efficient” proofs to statements of the form $y = (M, x, t)$, where y is considered to be a true statement if M is a non-deterministic machine that accepts x within t steps. The corresponding

language and witness relation are denoted L_U and $\mathbf{R}_U$ respectively, where the pair $((M, x, t), w)$ is in $\mathbf{R}_U$ if M (viewed here as a two-input deterministic machine) accepts the pair (x, w) within t steps. Notice that every language in NP is linear time reducible to L_U . Thus, a proof system for L_U allows us to handle all NP-statements. In fact, a proof system for L_U enables us to handle languages that are presumably “beyond” NP, as the language L_U is NE-complete (hence the name universal arguments).⁹

DEFINITION 11 (ORACLE-AIDED UNIVERSAL ARGUMENT). *An oracle-aided protocol (P, V) is called an $\mathcal{O}$ -oracle-aided universal argument system if it satisfies the following properties:*

- **Efficient verification:** *There exists a polynomial p such that for any $y = (M, x, t)$, and for any $\mathbf{pp}, O$ generated by $\mathcal{O}$, the total time spent by the (probabilistic) verifier strategy V , on common input y , $\mathbf{pp}$, is at most $p(|y| + |\mathbf{pp}|)$. In particular, all messages exchanged in the protocol have length smaller than $p(|y| + |\mathbf{pp}|)$.*
- **Completeness by a relatively efficient oracle-aided prover:** *For every $(y = (M, x, t), w)$ in $\mathbf{R}_U$,*

$$\Pr[\mathbf{pp}, O \leftarrow \mathcal{O}(1^{|y|}); (P^O(w), V)(y, \mathbf{pp}) = 1] = 1.$$

Furthermore, there exists a polynomial q such that the total time spent by $P^O(w)$, on common input $y = (M, x, t)$, $\mathbf{pp}$, is at most $q(T_M(x, w) + |\mathbf{pp}|) \leq q(t + |\mathbf{pp}|)$, where $T_M(x, w)$ denotes the running time of M on input (x, w) .

- **Weak proof of knowledge for adaptively chosen statements:** *For every polynomial p there exists a polynomial p' and a probabilistic polynomial-time oracle machine E such that the following holds: for every non-uniform polynomial-time oracle algorithm P^* , if $\Pr[\mathbf{pp}, O \leftarrow \mathcal{O}(1^n); R \leftarrow \{0, 1\}^\infty; y \leftarrow P_R^{*O}(\mathbf{pp}) : (P_R^{*O}(\mathbf{pp}), V(y, \mathbf{pp})) = 1] > 1/p(n)$ then*

$$\begin{aligned} \Pr[\mathbf{pp}, O \leftarrow \mathcal{O}(1^n); R, r \leftarrow \{0, 1\}^\infty; y \leftarrow P_R^{*O}(\mathbf{pp}) : \\ \exists w = w_1, \dots, w_t \in \mathbf{R}_U(y) \text{ s.t. } \forall i \in [t], \\ E_r^{P_R^{*O}}(\mathbf{pp}, y, i) = w_i] > \frac{1}{p'(n)} \end{aligned}$$

$$\text{where } \mathbf{R}_U(y) \stackrel{\text{def}}{=} \{w : (y, w) \in \mathbf{R}_U\}.$$

Note that our proof of knowledge condition is somewhat different from the one used in [2] in that we allow the (cheating) prover to *adaptively* choose the statement to be proved, after having seen the public parameter, and having interacted with its oracle.

Nevertheless, as we shall see, the construction of [2] and their analysis will be useful to us. Recall that in the construction of [2] tree hashing is used to hash down a “long” PCP proof into a fixed-length “tree root”; the soundness property relies on collision resistant of this tree hashing. Let SIG' be a canonical sig-com scheme with $\text{SIG} = (\text{Gen}, \text{Sign}, \text{Ver})$ and Com being its underlying signature scheme and commitment scheme. We observe that if we replace the use of tree hashing in [2] scheme with a sig-com tree using SIG' , then the resulting protocol is an $\mathcal{O}^{\text{SIG}}$ -aided universal argument for the following signature oracle $\mathcal{O}^{\text{SIG}}$.

⁹Furthermore, every language in NEXP is polynomial-time (but not linear-time) reducible to L_U

DEFINITION 12 (SIGNATURE ORACLE). *A signature oracle $\mathcal{O}^{\text{SIG}}$ is defined as follows: On input a security parameter n , $\mathcal{O}^{\text{SIG}}(1^n)$ generates $(\text{vk}, \text{sk}) \leftarrow \text{Gen}(1^n)$ and lets $\mathbf{pp} = \text{vk}$ and $\mathcal{O}(m) = \text{Sign}_{\text{sk}}(m)$ for every $m \in \{0, 1\}^{\text{poly}(n)}$.*

In fact, the universal argument has an even stronger completeness property that will be useful for us: completeness hold even if the prover only gets access to a sig-com oracle (instead of a signature oracle), and even if this is an *arbitrary* (not necessarily using the honest sign and commit algorithms) sig-com oracle, as long as the oracle outputs valid sig-com’s (for messages of a certain fixed length) with overwhelming probability. More formally,

DEFINITION 13 (VALID SIG-COM ORACLE). *An oracle $\mathcal{O}'$ is a **valid** (SIG', ℓ) **oracle** if there is a negligible $\mu(\cdot)$ such that for every $n \in N$, the following holds with probability $1 - \mu(n)$ over $\mathbf{pp}, O \leftarrow \mathcal{O}'(1^n)$: for every $m \in \{0, 1\}^{\ell(n)}$, $O(m)$ returns (σ, τ) such that $\text{Ver}'_{\text{vk}}(m, \sigma, \tau) = 1$ with probability at least $1 - \mu(n)$.*

We note that oracles that use arbitrarily biased randomness for commitment are also considered *valid* sig-com oracles. (These are precisely the kind of oracles we will be forced to use later on).

DEFINITION 14. *An $\mathcal{O}^{\text{SIG}}$ -aided universal argument (P, V) has **(SIG', ℓ) -completeness** if there exists a prover P' such that the completeness condition holds for (P', V) when the oracle $\mathcal{O}^{\text{SIG}}$ is replaced by any valid (SIG', ℓ) oracle $\mathcal{O}'$.*

We now have the following theorem.

THEOREM 7. *Let SIG' be a canonical sig-com scheme with SIG and Com being its underlying signature scheme and commitment scheme. Then there exists a polynomial ℓ and a (SIG', ℓ) -complete $\mathcal{O}^{\text{SIG}}$ -aided universal argument Π .*

The proof of the theorem identically follows that of Barak and Goldreich [2], with a minor modification to deal with adaptively chosen statements. The proof is found in the full version.

4.2 Oracle-aided Zero-Knowledge Protocols

We now turn to constructing oracle-aided resettable-sound zero-knowledge protocols. We start by defining a strong notion of an $\mathcal{O}$ -oracle-aided version of ZK. First of all, we restrict to protocols where the honest prover does not access the oracle. Secondly, we require that simulation can be performed given oracle access to *any* valid SIG' oracle. These two restrictions will be important when we later instantiate the oracle-aided protocol in the plain model.

DEFINITION 15 (ORACLE-AIDED ZERO-KNOWLEDGE). *A pair of algorithms (P, V) is **(SIG', ℓ) -oracle aided zero-knowledge** for a NP language L with witness relation R_L if for every non-uniform adversarial verifier V^* , there exists a simulator S , such that for every valid (SIG', ℓ) oracle $\mathcal{O}'$, the following ensembles are indistinguishable over $x \in L$,*

$$\begin{aligned} \{\mathbf{pp}, O \leftarrow \mathcal{O}'(1^{|x|}); \text{View}_{V^*}(P(w), V^*(z))(x, \mathbf{pp})\}_{x, w, z} \\ \approx \{\mathbf{pp}, O \leftarrow \mathcal{O}'(1^{|x|}); S^O(x, z, \mathbf{pp})\}_{x, w, z} \end{aligned}$$

where the ensembles are over $x \in L, w \in R_L(x), z \in \{0, 1\}^*$.

We now turn to the question of constructing a protocol that satisfies the above requirements. Note that, as a first attempt, we could try constructing a constant-round public-coin ZK protocol by replacing the tree hashing in Barak's protocol [1] with sig-com trees, and then apply the PRF transformation of [3] to achieve resettable soundness. While this indeed could be used to get a resettable-sound ZK protocol in the oracle-aided model, the resulting protocol would require the honest prover to make polynomially many queries to the oracle (to complete the WIUARG). To get around this, we instead rely on a variant of Barak's protocol used in Pass and Rosen [25], which provides a "special-purpose" implementation of the WIUARG used in Barak's protocol in which the honest prover does not need to perform any "hashing".¹⁰

More precisely, our protocol proceeds as follows. In Stage 1, P_{ZK}^O sends a commitment $c = \text{Com}(0^n)$, and then V_{ZK} sends back a challenge $r \in \{0, 1\}^{2^n}$ as in Barak's protocol. In Stage 2, P_{ZK}^O and V_{ZK} first execute an "encrypted" universal argument (P_{UA}^O, V_{UA}) of the statement that " c is a commitment to a sig-com tree root of a program M and there is a short string $y \in \{0, 1\}^n$ such that $M(y) = r$," where instead of sending the message in the clear, the prover sends commitments to the messages. The honest prover simply sends commitments to 0 (and thus will fail in this encrypted universal argument). Finally, P_{ZK}^O and V_{ZK} execute a witness-indistinguishable argument of knowledge of the statement that " $x \in L$ OR V_{UA} accepts in the encrypted universal argument".

A formal description of the protocol can be found in Fig. 1 and Fig. 2. Note that, in this construction, the honest prover P_{ZK}^O can convince the verifier by proving $x \in L$ in the final witness indistinguishable argument without making any oracle queries.

THEOREM 8. *Let SIG' be a canonical sig-com scheme with SIG and Com being its underlying signature scheme and commitment scheme. Then there exists an $\mathcal{O}^{\text{SIG}}$ -oracle aided argument of knowledge (P, V) for NP; additionally,*

1. (P, V) is constant-round and public-coin;
2. P does not make any queries to its oracle;
3. (P, V) is (SIG', ℓ) -oracle-aided zero-knowledge for some polynomial ℓ .

The proof of Theorem 8 is found in the full version. The proof closely follows [1, 25] but the proof of the "argument of knowledge" property requires special care to deal with the fact that a cheating prover may adaptively choose the statements to be proved in the encrypted universal argument (after having interacted with its oracle).¹¹

Finally, we apply the PRF transformation of [3] to (P_{ZK}^O, V_{ZK}) to achieve resettable soundness. More precisely, we modify the public-coin verifier V_{ZK} to a "PRF-verifier" V_{ZK} that samples a seed s for a PRF f_s at beginning and then generates each verifier message by applying f_s to the current transcript. The proof in [3] relativizes and as a consequence we have the following theorem:

¹⁰In fact, early versions of Barak's protocol also relied on such a special-purpose implementation of WIUARG.

¹¹In [1, 25] this issue does not arise since different, independently chosen hash-functions are used in Stage 1 and in Stage 2.

Common Input: An instance x of a language $L \in \text{NP}$ with witness relation $\mathbf{R}_L$.

Auxiliary input to P : A witness w such that $(x, w) \in \mathbf{R}_L$.

Primitives Used: A canonical sig-com scheme SIG' with SIG and Com as the underlying signature and commitment schemes, $\mathcal{O}^{\text{SIG}}$ defined relative to SIG , and a $\mathcal{O}^{\text{SIG}}$ -aided universal argument (P_{UA}, V_{UA}) defined in Sec. 4.1.

Set Up: Run $(\text{pp}, O) \leftarrow \mathcal{O}^{\text{SIG}}(1^n)$, add pp to common input for P and V . Further, allow P oracle access to O .

Stage One (Trapdoor):

P_1 : Send $c_0 = \text{Com}(0^{2^n}, \tau_0)$ to V with uniform τ_0

V_1 : Send $r \xleftarrow{\$} \{0, 1\}^n$ to P

Stage Two (Encrypted OA-UA):

P_2 : Send $c_1 = \text{Com}(0^{2^n}, \tau_1)$ for uniformly selected τ_1

V_3 : Send r' , uniformly chosen random tape for V_{OA-UA}

P_3 : Send $c_2 = \text{Com}(0^k, \tau_2)$ for uniformly selected τ_2 , where k is the length of P_{OA-UA} 's second message.

Stage Three (Main Proof):

$P \Leftrightarrow V$: A WI-AOK (P_{WI}, V_{WI}) proving the OR of the following statements:

1. $\exists w \in \{0, 1\}^{\text{poly}(|x|)}$ s.t. $(x, w) \in \mathbf{R}_L$.
2. $\exists \langle p_1, p_2, \tau_1, \tau_2 \rangle$ s.t. $\langle (c_0, r, c_1, c_2, r', \text{pp}), (p_1, p_2, \tau_1, \tau_2) \rangle \in \mathbf{R}_{L_2}$ (defined in Fig. 2).

Figure 1: $\mathcal{O}^{\text{SIG}}$ -aided ZK argument of knowledge.

Relation 1: Let SIG' a sig-com scheme, with underlying signature scheme SIG and commitment scheme Com . Let ECC be a strong error-correcting code. We say that $\langle c_0, r, \text{pp} \rangle \in L_1$ if $\exists \langle \tau_0, d, l_\lambda, C, \{\tilde{\rho}_i\}_{i \in [2d]} \rangle$ such that

- $c_0 = \text{Com}((d, l_\lambda), \tau_0)$
- (d, l_λ) are the depth and root of a sig-com tree for C w.r.t. pp
- Each $\tilde{\rho}_i$ is a valid sig-com path for leaf i of this sig-com tree. That is, $\text{PATH}^{\text{SIG}'}(\tilde{\rho}_i, C_i, i, l_\lambda, \text{pp}) = 1$ for each i .
- $C = \text{ECC}(\Pi)$ for some circuit Π
- $\Pi(c_0) = r$.

We let $\mathbf{R}_{L_1}$ be the witness relation corresponding to L_1 .

Relation 2: Let L_1 be described as above, with respect to schemes SIG' and ECC . Let (P_{UA}, V_{UA}) be the $\mathcal{O}^{\text{SIG}}$ -aided universal argument constructed in Sec. 4.1. We say that $\langle c_0, r, c_1, c_2, r', \text{pp} \rangle \in L_2$ if $\exists \langle p_1, p_2, \tau_1, \tau_2 \rangle$ such that

- $c_1 = \text{Com}(p_1, \tau_1)$, $c_2 = \text{Com}(p_2, \tau_2)$
- (p_1, r', p_2) constitutes an accepting (P_{UA}, V_{UA}) transcript for $\langle c_0, r \rangle \in L_1$.

We let $\mathbf{R}_{L_2}$ be the witness relation corresponding to L_2 .

Figure 2: Relations used in $\mathcal{O}^{\text{SIG}}$ -aided ZK protocol.

THEOREM 9. *Let SIG' be a canonical sig-com scheme with SIG and Com being its underlying signature scheme and commitment scheme. Then there exists an $\mathcal{O}^{\text{SIG}}$ -aided constant-round resettably-sound argument of knowledge (P, V) for NP ; additionally,*

1. P does not make any queries to its oracle;
2. (P, V) is (SIG', ℓ) -oracle-aided zero-knowledge for some polynomial ℓ .

5. RS-ZK IN THE PLAIN MODEL

Let SIG' be a canonical sig-com scheme with SIG and Com being its underlying signature scheme and commitment scheme. Let (P, V) be a $\mathcal{O}^{\text{SIG}}$ -aided resettably sound argument of knowledge for the language L with witness relation R_L , where P does not make any queries to its oracle. Consider the protocol $(\tilde{P}, \tilde{V})$ that on common input x , and auxiliary prover input w proceeds as follows.

1. Init: $\tilde{V}$ runs $(\text{sk}, \text{vk}) \leftarrow \text{Gen}(1^n)$ and sends vk to $\tilde{P}$.
2. Signing Slot:
 - $\tilde{P}$ generates $c = \text{Com}(0^{2n}; \tau)$, where τ is uniformly sampled, and sends c to $\tilde{V}$.
 - $\tilde{V}$ replies with $\sigma = \text{Sign}_{\text{sk}}(c)$.
 - $\tilde{P}$ aborts if σ is not a valid signature of c .
3. Body: Invoke $(P(w), V)(x, \text{pp})$ with $\text{pp} = \text{vk}$.

LEMMA 10. *If (P, V) is $(\text{SIG}', 2n)$ -oracle-aided zero-knowledge for L with witness relation R_L , then $(\tilde{P}, \tilde{V})$ is a single-instance resettably-sound zero-knowledge argument of knowledge for L with witness relation R_L .*

Note that here we only obtain a *single-instance* resettably sound argument of knowledge (defined in Defn. 3), but this can be transformed into a “full-fledged” resettably sound one by using the transformation in Claim 2, which thus establishes our main Theorem 1. The proof of Lemma 10 is found in the full version. We provide a very brief sketch below.

PROOF. (sketch) Completeness of $(\tilde{P}, \tilde{V})$ follows directly from the completeness of (P, V) since by assumption, P never makes any oracle queries. Resettably-soundness and the argument of knowledge property, roughly speaking, follow by emulating all signature slot messages using the oracle; note that we here rely on the fact that the signature scheme is deterministic to ensure that “rewindings” of the signature slot can be emulated as oracle queries. The zero-knowledge simulator proceeds by first honestly emulating the signature slot for the malicious verifier V^* , and if V^* provides an accepting signature, we next run the oracle-aided simulator, and appropriately rewinding the malicious verifier during the signature slot to appropriately implement *some* valid sig-com oracle. The verifier may not always answer, but we can “keep rewinding” him, sending fresh commitments until he does. Roughly speaking, the key point is that if V^* did provide a valid signature during the first pass, then in expectation, by the hiding property of the commitment scheme, we only need a polynomial number of rewindings. This “almost” works: just as in [13], we need to take special care to deal with verifier’s that only provide valid signatures with very small probability. $\square$

Acknowledgements

We are very grateful to Ran Canetti for pointing out the connection to [9].

6. REFERENCES

- [1] B. Barak. How to go beyond the black-box simulation barrier. In *FOCS '01*, pages 106–115, 2001.
- [2] B. Barak and O. Goldreich. Universal arguments and their applications. In *Computational Complexity*, pages 162–171, 2002.
- [3] B. Barak, O. Goldreich, S. Goldwasser, and Y. Lindell. Resettably-sound zero-knowledge and its applications. In *FOCS'02*, pages 116–125, 2001.
- [4] B. Barak, O. Goldreich, R. Impagliazzo, S. Rudich, A. Sahai, S. P. Vadhan, and K. Yang. On the (im)possibility of obfuscating programs. *J. ACM*, 59(2):6, 2012.
- [5] N. Bitansky and O. Paneth. On the impossibility of approximate obfuscation and applications to resetttable cryptography. In *STOC*, 2011.
- [6] N. Bitansky and O. Paneth. From the impossibility of obfuscation to a new non-black-box simulation technique. In *FOCS*, 2012.
- [7] J. Camenisch and A. Lysyanskaya. An efficient system for non-transferable anonymous credentials with optional anonymity revocation. *Advances in Cryptology EUROCRYPT 2001*, pages 93–118, 2001.
- [8] R. Canetti, O. Goldreich, S. Goldwasser, and S. Micali. Resetttable zero-knowledge (extended abstract). In *STOC '00*, pages 235–244, 2000.
- [9] R. Canetti, O. Goldreich, and S. Halevi. On the random-oracle methodology as applied to length-restricted signature schemes. In *TCC*, pages 40–57, 2004.
- [10] R. Canetti, J. Kilian, E. Petrank, and A. Rosen. Black-box concurrent zero-knowledge requires $\tilde{\omega}(\log n)$ rounds. In *STOC '01*, pages 570–579, 2001.
- [11] Y. Deng, V. Goyal, and A. Sahai. Resolving the simultaneous resettability conjecture and a new non-black-box simulation strategy. In *Foundations of Computer Science, 2009. FOCS'09. 50th Annual IEEE Symposium on*, pages 251–260. IEEE, 2009.
- [12] O. Goldreich. *Foundations of Cryptography — Basic Tools*. Cambridge University Press, 2001.
- [13] O. Goldreich and A. Kahan. How to construct constant-round zero-knowledge proof systems for NP. *Journal of Cryptology*, 9(3):167–190, 1996.
- [14] O. Goldreich, S. Micali, and A. Wigderson. Proofs that yield nothing but their validity for all languages in NP have zero-knowledge proof systems. *J. ACM*, 38(3):691–729, 1991.
- [15] O. Goldreich and Y. Oren. Definitions and properties of zero-knowledge proof systems. *Journal of Cryptology*, 7:1–32, 1994.
- [16] S. Goldwasser, S. Micali, and C. Rackoff. The knowledge complexity of interactive proof systems. *SIAM Journal on Computing*, 18(1):186–208, 1989.
- [17] J. Håstad, R. Impagliazzo, L. Levin, and M. Luby. A pseudorandom generator from any one-way function. *SIAM Journal on Computing*, 28:12–24, 1999.
- [18] H. Lin and R. Pass. Constant-round non-malleable commitments from any one-way function. In *STOC*, pages 705–714, 2011.
- [19] R. Merkle. Digital signature system and method based on a conventional encryption function, Nov. 14 1989. US Patent 4,881,264.
- [20] S. Micali. Computationally sound proofs. *SIAM Journal on Computing*, 30(4):1253–1298, 2000.
- [21] M. Naor. Bit commitment using pseudorandomness. *Journal of Cryptology*, 4(2):151–158, 1991.
- [22] M. Naor and M. Yung. Universal one-way hash functions and their cryptographic applications. In *STOC '89*, pages 33–43, 1989.
- [23] R. Ostrovsky and A. Wigderson. One-way functions are essential for non-trivial zero-knowledge. In *Theory and Computing Systems, 1993*, pages 3–17, 1993.
- [24] R. Pass. Bounded-concurrent secure multi-party computation with a dishonest majority. In *STOC '04*, pages 232–241, 2004.
- [25] R. Pass and A. Rosen. New and improved constructions of non-malleable cryptographic protocols. In *STOC '05*, pages 533–542, 2005.

- [26] R. Pass, W.-L. D. Tseng, and M. Venkatasubramanian. Concurrent zero knowledge: Simplifications and generalizations. Manuscript, 2008. <http://hdl.handle.net/1813/10772>.
- [27] J. Rompel. One-way functions are necessary and sufficient for secure signatures, 1990.

Fast Two-Party Secure Computation with Minimal Assumptions*

ABSTRACT

Almost all existing protocols for secure two-party computation require a specific hardness assumption such as decisional Diffie-Hellman, discrete logarithm, or a random oracle even after assuming oracle access to the oblivious transfer functionality for their correctness and/or efficiency. We propose and implement a Yao-based protocol that is secure against malicious adversaries and enjoys the following benefits:

1. it requires the minimal hardness assumption, i.e., OTs;
2. it uses 10 rounds of communication plus OT rounds;
3. it has the optimal overhead complexity (for an approach that uses the circuit-level cut-and-choose technique); and
4. it is embarrassingly parallelizable in the sense that each circuit can be processed in a pipelined manner, and all circuits can be processed in parallel.

To achieve these properties, we solve the three main issues for achieving malicious security in a novel and efficient manner. In particular, we propose an efficient witness-indistinguishable proof for *the generator's output authenticity*; we suggest the use of an auxiliary circuit that computes a hash to ensure *the generator's input consistency*; and we advance the performance of the state-of-the-art approach defending *the selective failure attack*.

Not only does our protocol require weaker cryptographic assumptions, but our implementation of this protocol also demonstrates a several factor improvement over the best prior work, which relies on specific number-theoretic assumptions. Thus, we show that performance does not rely on specific assumptions.

*This work is supported by Defense Advanced Research Projects Agency (DARPA) and the Air Force Research Laboratory (AFRL) under contract FA8750-11-2-0211. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the Defense Advanced Research Projects Agency or the US government.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

Copyright 20XX ACM X-XXXXX-XX-X/XX/XX ...\$15.00.

Keywords

the Yao protocol, malicious model, cut-and-choose

1. INTRODUCTION

Secure two-party computation research aims to allow two parties to collaborate in a way that achieves maximal privacy of their inputs with simultaneous guarantees of correctness of outputs. The correctness property guarantees that when both parties follow the protocol honestly, the protocol output is indeed the output of the objective function. The privacy property ensures that during the protocol execution, neither party can learn more than that derivable from her own input and output. A trivial solution is have both parties hand their private inputs to this third party who performs the work and later distributes the computation result. However, the solution that cryptographers desire achieves the effect of a trusted party without one.

The first generic solution for secure two-party computation in the *honest-but-curious* model was proposed by Yao [24]. In this protocol, both parties agree on an objective function and its boolean circuit format (called the *objective circuit*) in advance¹. One party (called the *generator*) constructs a garbled version of this objective circuit, and the other party (called the *evaluator*) then obviously evaluates this garbled circuit and gets the output. By oblivious evaluation we mean that the evaluator does not learn any intermediate value of the computation. This protocol satisfies the two security properties if both participants follow the protocol instructions honestly.

This basic protocol must be hardened to handle the situation in which either party arbitrarily deviates from the protocol. A simply cheat is for a malicious generator to construct a faulty circuit that breaks the security properties, for example, one that purposely reveals the evaluator's private input. Since the circuit is garbled, the evaluator could not tell whether the circuit is faulty or not.

The *cut-and-choose* technique is one of the most efficient methods that enforce honest circuit garbling [9, 13, 14, 17, 21]. At a high level, this technique instructs the generator to prepare multiple copies of the garbled circuit, each with independent randomness, and instructs the evaluator then to randomly pick a fraction of the circuits whose randomness is later revealed. If any of the chosen circuits (called the *check circuits*) is not consistent with the revealed randomness, the evaluator aborts and the generator is caught cheating; other-

¹The equivalence between the objective function and the objective circuit is out of the scope of this paper.

wise, the evaluator starts to evaluate the remaining circuits (called the *evaluation circuits*) as instructed in the Yao protocol. Finally, the evaluator takes the majority of the evaluation outputs as the final output. As a result, a malicious generator constructs either too many faulty circuits and gets caught, or only a few and does not influence the final output at all.

Besides the threat of faulty circuits, there remain three other subtle but equally critical security issues that need to be addressed when dealing with malicious adversaries. The first two in fact result from the use of multiple garbled circuits (as instructed by the cut-and-choose technique): *two-output function handling* and *the generator's input consistency*. The third issue is also known as the *selective failure attack*. Prior work in this area that addresses these three concerns either requires specific hardness assumptions, or introduces large overheads in communication and computation.

1.1 Contributions

The main contribution of this work is to construct an *optimal* protocol in the circuit-level cut-and-choose-based category that (1) it requires minimal hardness assumptions, namely, an oblivious transfer (OT) protocol secure in the presence of malicious adversaries; (2) it introduces little computational and communicational overhead to solve the above three problems. In particular, its complexity is linear (in terms of the security parameter) to the original Yao protocol's, which is the best a circuit-level cut-and-choose-based solution could ever achieve. In other words, we show that *malicious security comes almost for free* both in terms of required hardness assumptions and various protocol performance metrics. Let n denote the input and output size, k denote the security parameter, σ denote the number of garbled circuits needed (typically, $\sigma = O(k)$). The contributions of this work are as follows:

1. We propose a novel witness-indistinguishable proof to ensure the generator's output authenticity that requires only standard primitives (any symmetric encryption scheme and commitment scheme to be precise) and incurs about the same amount of overhead as garbling and evaluating the generator's output gates. In other words, it requires only $O(\sigma n)$ symmetric cryptographic operations; prior approaches require $O(\sigma^2 n)$ symmetric operations [13, 20] or $O(\sigma n)$ symmetric operations plus $O(\sigma)$ algebraic operations [9].
2. We suggest the use of an auxiliary circuit to achieve the generator's input consistency. This auxiliary circuit only needs to compute the (universal) hash of the generator's input. By utilizing an XOR-homomorphic hash, we are able to evaluate the auxiliary circuit almost for free. Our solution is much more efficient than the prior works which need $O(\sigma^2 n)$ symmetric operations [13, 16] or $O(\sigma n)$ algebraic operations [21].
3. Most importantly, the above two techniques allow us to handle issues of two-output functions and the generator's input consistency while avoiding algebraic operations entirely. Lindell and Pinkas' [13] and Woodruff's [23] approaches are the only prior works, to the best of our knowledge, that do not rely on any number-theoretic assumptions. Nevertheless, our approach works in a more efficient manner as shown in Table 1.

4. Lindell and Pinkas suggested the use of a k -probe-resistant matrix (cf. Definition 5) to defend against the selective failure attack [13]. This solution has little overhead while combining with the free-XOR technique. However, it increases the number of OTs needed at the same time. While XOR gates can be computed almost for free, the OTs can not. While there exist extension techniques for OTs, not all variants of OTs can be efficiently extended. We therefore design a probabilistic algorithm based on Reed-Solomon code such that the number of OTs needed can be as low as 25% of that in the original Lindell and Pinkas' solution.

5. We propose an optimization technique that can save communication overhead by up to 60% (when 60% of all the garbled circuits are check circuits) with the price of a slight increase of computation overhead. We stress that our technique compares favorably with the Random Seed Checking technique [12]. In particular, our approach is compatible with the pipelining technique [6].

6. Based on an open-source system [12], we experimentally verify our theories by developing some of the above techniques. The integrated system can process 650,000+ gates per second on Stampede [22]. This is the fastest maliciously secure two-party system reported.

1.2 Related Work

While substantial efforts have been spent on converting the Yao protocol based on the cut-and-choose technique [9, 12–14, 16, 20, 21, 23] into maliciously secure protocols, several completely different approaches have been reported. Jarecki and Shmatikov suggested an approach that needs only a single copy of the garbled circuit, but this approach requires expensive zero-knowledge proof of correctness for every single gate [8] that also rely on specific RSA-based hardness assumptions. Nielson et al. reported a solution that uses efficient OTs to generate a pool of authenticated primitives [19]. With these primitives, both parties are able to securely evaluate a boolean circuit based on the generic Goldreich, Micali, and Wigderson protocol [3]. However, this protocol requires interactive communication for every AND gate, and therefore the number of rounds of communication depends on the circuit. While suitable for small circuits, large complicated circuits will require thousands of back-and-forth messages. Damgård et al. proposed a solution that uses somewhat homomorphic encryption to precompute a bunch of triples that are later used to securely compute an arithmetic circuit [1].

Our approach outperforms other approaches in the cut-and-choose-based category in terms of the number of symmetric or algebraic operations needed, as shown in Table 1. Note that by “in the cut-and-choose-based category,” we mean that the number of the garbled circuits needed is linear to the security parameter. So there is a hidden cost of $O(kC)$ of symmetric cryptographic operations in all these approaches, where C is circuit size. More details about Table 1 will be given in Section 3.

Our approach is also as competitive as any other in terms of computation complexity, round complexity, and the computation assumptions needed, as shown in Table 2. While Jarecki and Shmatikov's approach requires hundreds of expensive algebraic operations per gate, ours does not need any (given oracle access to OTs) [8]. Although Neilson et al.'s approach favorably compares to our approach in terms

	Gen. Symm. Op.	Input Consist. Alge. Op.	Gen. Output Symm. Op.	Auth. Alge. Op.	Assumptions (besides OT)
[13]	$O(k^2n)$		$O(k^2n)$		Standard (OWF)
[9]	$O(k^2n)$		$O(kn)$	$O(k)$	Discrete Log.
[14]	$O(kn)$	$O(kn)$	not mentioned		Decisional Diffie-Hellman
[21]	$O(kn)$	$O(kn)$	$O(kn)$	$O(kn)$	Discrete Log.
[12]	$O(kn)$	$O(kn)$	$O(kn)$	$O(k)$	Discrete Log.
This Work	$O(kn)$		$O(kn)$		Standard (OWF)

Table 1: Complexity of various circuit-level cut-and-choose-based approaches in terms of symmetric (or algebraic) operations.

of computation complexity, ours requires constant communication rounds while theirs needs rounds linear to the circuit depth [19]. At last, since Damgård et al.’s approach works with arithmetic circuits, it is incomparable to our work and thus omitted in the table [1]

	Symm. Op.	Alge. Op.	Rounds	Assumptions
[8]	$O(C)$	$O(C)$	$O(1)$	DCR + RSA
[18]	$O(\frac{k}{\lg C}C)$	$O(\frac{k}{\lg C}C)$	$O(1)$	Discrete Log
[19]	$O(\frac{k}{\lg C}C)$		$O(D)$	Random Oracle
This	$O(kC)$		$O(1)$	Standard (OWF)

Table 2: Overall complexity comparison with prior works, where C is the circuit size and D_f is the circuit depth.

We recently noticed an independent work by Mohassel and Riva that also proposes an optimal Yao-based protocol [17]. Their protocol indeed shares the same asymptotic complexity as ours and also relies on minimal assumptions. However, our approach favorably compares to theirs for two reasons:

1. The protocols for ensuring the generator’s output authenticity in both works are essentially the same. Both protocols capture the idea that the evaluator provides a unique random key corresponding to each of the generator’s output wires as the proof of authenticity. The only difference is that for each of the generator’s output wires in each of the garbled circuits, Mahassel and Riva’s protocol requires two possible random keys, which correspond to 0 or 1, to be encrypted and exchanged, whereas our protocol only needs the one that corresponds to the generator’s output value to be encrypted and exchanged. In other words, although both solutions need $O(\sigma n)$ symmetric cryptographic operations, ours has a smaller constant factor.
2. Their approach for checking GEN’s input consistency uses a different instance of circuit garbling, in which GEN’s and EVAL’s roles are reversed. In other words, while their solution introduces $O(n)$ instances of OTs, which is arguably the most expensive component of the Yao protocol, ours introduces only cryptographic symmetric operations.

Paper Organization: We first give background and notations in Section 2. We then show how the three attacks are handled by cryptographic primitives in Section 3. A detailed description of our main protocol and the argument of its security is presented in Section 4. The optimization technique that saves the communication overhead by at much as

60% is reported in Section 5. Finally, experimental results are reported in Section 6.

2. PRELIMINARIES

Since our solution is based on the Yao protocol, the two parties are referred to as GEN and EVAL for the rest of this paper. We denote by $f(x, y) \mapsto (f_1(x, y), f_2(x, y))$ a two-output objective function, where GEN with input x gets output $f_1(x, y)$ and EVAL with input y gets output $f_2(x, y)$. For simplicity, $f_1(x, y)$ and $f_2(x, y)$ are often noted as f_1 and f_2 , respectively. We use $f_1 = \perp$ or $f_2 = \perp$ to indicate that either GEN or EVAL gets no output. We denote by $\text{com}(x; r)$ the commitment to message x with randomness r . The randomness may be omitted for simplicity. We denote by $\text{enc}_e(x)$ the encryption of message x under encryption key e and by $\text{dec}_d(c)$ the decryption of ciphertext c under decryption key d . Additionally, we denote by $x||y$ or sometimes (x, y) the concatenation of x and y , and by $[n]$ the set $\{1, 2, \dots, n\}$ for some $n \in \mathbb{N}$. We also adopt the notation that $x^{(j)}$ ’s superscript implies that this variable is related to the j -th garbled circuit.

Furthermore, for the rest of this paper, we denote by k the security parameter, by σ the number of copies of the garbled circuit needed (also known as the statistical security parameter), and by n the size of a participant’s input and output..

3. MALICIOUS SECURITY

Our protocols achieve security against malicious adversaries according to the standard ideal-real paradigm for defining security. In the full version of this paper, we present the standard definition of ideal-real security and prove that our protocols achieve this notion. Since the proof techniques already highlighted in [13] and [21] suffice for our proof, in this short abstract, we omit the full proofs and focus on discussing our important contributions—namely, how we solve the three security issues faced by the protocol that transforms the Yao protocol into one that is secure in the malicious model via the cut-and-choose technique.

3.1 Two-Output Function Handling

For many real-world applications, both parties want to learn an output from the secure computation. Since EVAL always learns her output, the challenge is for GEN to learn hers securely. In particular, a solution needs to achieve GEN’s *output privacy* and *output authenticity*. The former requires that EVAL does not learn GEN’s output, and the latter requires that GEN gets either an authentic output or

no output at all, in which case EVAL is caught cheating. We stress that the two-output protocols we consider are not *fair*; that is, EVAL may learn her own output but refuse to send GEN's back—but if so, EVAL is caught cheating.

Goldreich suggested the use of an auxiliary circuit that encrypts GEN's output and computes the digital signature of the resulting ciphertext so that a malicious EVAL could neither learn GEN's output from the ciphertext nor forge an arbitrary signature [2]. Later, Lindell and Pinkas proposed an approach that uses one-time-pad encryption and one-time MAC circuits instead, which incurs $O(kn)$ extra gates per circuit [13]. Kiraz presented a two-party protocol in which a zero-knowledge proof of size $O(\sigma)$ is executed at the end [9]. shelat and Shen reported a signature-based solution that adds $O(n)$ gates to each circuit, and requires a WI proof of size $O(\sigma + n)$ [21] that requires specific complexity assumptions.

Our approach solves GEN's output privacy problem with a one-time-pad encryption circuit, which requires only $O(n)$ extra XOR-gates per circuit. The novel part of our approach is that we tackle the output authenticity problem in a way that needs no algebraic operations at all (hence no number-theoretic intractability assumption is needed). Our idea comes from the following three observations.

We first observe that the random keys retrieved from evaluating GEN's output gates can in fact serve as "message authentication codes" sufficient for EVAL to prove GEN's output authenticity. Recall that the Yao protocol ensures that EVAL learns exactly one of the two random keys assigned to each wire. So the knowledge of the retrieved random key corresponding to GEN's output wire is more than enough for EVAL to show the output authenticity. What remains is how EVAL demonstrates this knowledge without revealing the index of the garbled circuit from which EVAL retrieves the random key. This index has been shown to be exploitable in breaching EVAL's input privacy [9].

The second observation we have, which is also pointed out in shelat and Shen's work [21], is that a WI proof suffices the purposes here. Let us consider the case in which GEN (plays as the verifier) has private input $U = (u^{(1)}, u^{(2)}, \dots, u^{(s)})$ for some $s \in \mathbb{N}$, and EVAL (plays as the prover) knows $u^{(m)}$ for some $m \in [s]$. In the honest-but-curious model, a simple WI proof of EVAL's knowledge $u^{(m)}$ can be done as follows:

1. GEN picks random nonce r , and sends EVAL

$$(\text{enc}_{u^{(1)}}(r), \text{enc}_{u^{(2)}}(r), \dots, \text{enc}_{u^{(s)}}(r)).$$

2. EVAL receives $C = (c^{(1)}, c^{(2)}, \dots, c^{(s)})$ and returns $\text{dec}_{u^{(m)}}(C)$
3. GEN receives r' and accepts if $r' = r$, or aborts otherwise.

This proof is sound because an EVAL with no knowledge of any $u^{(m)} \in U$ can only guess r with negligible probability. However, this proof is not WI in the malicious model. In fact, a malicious GEN may pick distinct $r^{(j)}$ s and send EVAL

$$(\text{enc}_{u^{(1)}}(r^{(1)}), \text{enc}_{u^{(2)}}(r^{(2)}), \dots, \text{enc}_{u^{(s)}}(r^{(s)}))$$

so that $u^{(m)}$ can later be deduced by locating r' in $(r^{(1)}, r^{(2)}, \dots, r^{(s)})$.

To force GEN to behave honestly in Step 1, we suggest that GEN discloses (U, r) after receiving r' so that EVAL could check if C is constructed correctly. Our third observation is that this disclosure does not compromise GEN's input privacy. Indeed, EVAL should have already learned the majority

of U from the circuit evaluation, and r is a random nonce that has no information about GEN's input at all. So GEN's input is not leaked through (U, r) . Moreover, this disclosure does not compromise the soundness of the protocol since after GEN receives r' , EVAL has already delivered her proof of authenticity so that learning (U, r) afterwards will not change the proof retroactively. Nonetheless, we stress that GEN should not learn r' *before* EVAL finishes the check, and nor should EVAL be able to change r' after the check. This property suggests the use of a commitment scheme. Our idea is that EVAL commits to $\text{dec}_{u^{(m)}}(c^{(m)})$ instead of giving it away in clear. After GEN reveals (U, r) , EVAL checks if C is indeed correctly generated. If the check fails, EVAL aborts; otherwise, EVAL decommits to r' . Then GEN checks if $r' = r$ and responds as in the honest-but-curious protocol.

One more issue is that a malicious GEN could learn EVAL's private input $u^{(m)}$ with non-negligible probability by faking her private inputs from the beginning. More specifically, a malicious GEN could guess $u^{(m)}$ with probability $1/s$ and then pretend that her private input is $\bar{U} = (\bar{u}^{(1)}, \dots, \bar{u}^{(m-1)}, u^{(m)}, \bar{u}^{(m+1)}, \dots)$ instead of U , where $\bar{u}^{(j)}$ is randomly picked. With this attack, a malicious GEN is capable of providing checkable ciphertexts C when EVAL's private input is indeed $u^{(m)}$. Besides, the fact that EVAL can provide the correct nonce r confirms that her private input is $u^{(m)}$. A straightforward way to get around this issue is for the two parties to share the commitments to GEN's private inputs in the first place. By the binding property of the commitment scheme, a malicious GEN cannot change her private inputs at will. The correctness of these commitments will be guaranteed by the circuit-level cut-and-choose technique.

The complete description of our GEN's output authenticity protocol is presented in Figure 1, and the security of this protocol is stated in Lemma 1.

Common Input: security parameter 1^k , statistical security parameter 1^s , GEN's output bit b , and commitments to GEN's private input $\{(\text{com}(u_0^{(j)}), \text{com}(u_1^{(j)}))\}_{j \in [s]}$.

Private Input: GEN has $\{(u_0^{(j)}, u_1^{(j)})\}_{j \in [s]}$ and EVAL has random key v corresponding to GEN's output wire of value b in the m -th garbled circuit for some $m \in [s]$.

1. GEN picks a random nonce $r \in \{0, 1\}^k$ and sends EVAL $(\text{enc}_{u_b^{(1)}}(r), \text{enc}_{u_b^{(2)}}(r), \dots, \text{enc}_{u_b^{(s)}}(r))$.
2. After receiving $C = (c^{(1)}, c^{(2)}, \dots, c^{(s)})$, EVAL sends $\text{com}(\text{dec}_v(c^{(m)}))$ back to GEN.
3. After receiving $\text{com}(r')$, GEN decommits to $\{u_b^{(j)}\}_{j \in [s]}$.
4. EVAL checks the decommitted values $\{u_b^{(j)}\}_{j \in [s]}$ that
 - (a) if $\text{com}(u_b^{(j)})$ is correctly opened to $u_b^{(j)}$ for all j ?
 - (b) if $\text{dec}_{u^{(i)}}(c^{(i)}) \stackrel{?}{=} \text{dec}_{u^{(j)}}(c^{(j)})$ for all i, j ?
 EVAL aborts if any of the checks fails; otherwise, she decommits to r' .
5. GEN accepts the proof if $\text{com}(r')$ is correctly opened and $r' = r$; otherwise, she rejects.

Figure 1: A WI proof for GEN's output authenticity with malicious security (where EVAL plays the role of the prover)

LEMMA 1. Let $\{\text{com}(u_0^{(j)}), \text{com}(u_1^{(j)})\}_{j \in [s]}$ and bit b be common inputs. Suppose GEN has private input $\{(u_0^{(j)}, u_1^{(j)})\}_{j \in [s]}$, where $u_0^{(j)}, u_1^{(j)} \in \{0, 1\}^k$. The protocol presented in Figure 1 satisfies the following properties:

1. **(Completeness)** If EVAL knows $u_b^{(j)}$ for some $j \in [s]$, GEN always accepts.
2. **(Soundness)** If EVAL does not know any of $u_b^{(j)}$ s, GEN rejects with probability at least $1 - 2^{-k}$.
3. **(Witness-indistinguishability)** Let us denote by VIEW_v the view of GEN from running the protocol with EVAL using input v . If EVAL knows the majority V of $(u_b^{(1)}, u_b^{(2)}, \dots, u_b^{(s)})$, then for any $v_1, v_2 \in V$, $\{\text{VIEW}_{v_1}\}_{k \in \mathbb{N}}$ and $\{\text{VIEW}_{v_2}\}_{k \in \mathbb{N}}$ are computationally indistinguishable.

3.2 Generator's Input Consistency

In the cut-and-choose technique, multiple copies of the garbled circuit are constructed and then either checked or evaluated. It is conceivable that a malicious GEN may provide inconsistent inputs to different evaluation circuits. Lindell and Pinkas showed that for some functions, it is not difficult for a malicious GEN to use inconsistent inputs to extract information of EVAL's input [13]. For instance, suppose both parties agree upon the objective function

$$f((a_1, a_2, a_3), (b_1, b_2, b_3)) \mapsto (a_1 b_1 \oplus a_2 b_2 \oplus a_3 b_3, \perp),$$

where a_i and b_i is GEN's and EVAL's i -th input bit, respectively. Instead of providing (a_1, a_2, a_3) consistently, a malicious GEN may send $(1, 0, 0)$, $(0, 1, 0)$, and $(0, 0, 1)$ to different evaluation circuits. In the end, GEN learns the majority bit of EVAL's input, which is the extra information that EVAL did not agree to reveal.

Several approaches have been proposed to defend this attack. Mohassel and Franklin proposed the equality-checker technique, which requires $O(\sigma^2 n)$ commitments to be computed and exchanged [16]. Lindell and Pinkas developed an approach that also requires $O(\sigma^2 n)$ commitments [13]. Later, they realized that $O(\sigma^2 n)$ commitments are too much of the communication overhead, and then further suggested a pseudo-random synthesizer that relies on efficient zero-knowledge proofs under specific hardness assumptions and requires $O(\sigma n)$ algebraic operations [14]. shelat and Shen proposed the use of malleable claw-free collections, which also uses $O(\sigma n)$ algebraic operations, but they showed that the witness-indistinguishability is sufficient [21]. Our approach gets the best of the both worlds, i.e., it requires only $O(\sigma n)$ symmetric cryptographic operations.

We suggest to tackle this issue with an auxiliary circuit, in addition to the objective circuit, that computes a function of GEN's input. At a high level, the circuit-level cut-and-choose technique ensures the correctness of this auxiliary circuit, and its output is used to ensure GEN's input consistency. In particular, for this idea to work, we need to endow the output of this auxiliary circuit with *collision-free* and *hiding* properties. The former ensures that the consistency of the auxiliary outputs implies the consistency of GEN's inputs, and the latter ensures that the auxiliary outputs do not reveal any information of GEN's inputs.

A natural candidate for this auxiliary circuit is a commitment circuit. The binding and hiding properties of a commitment scheme satisfy the two security properties needed

here. This is a conceptually much simpler solution. We, however, failed to find a commitment circuit that introduces less overhead than the previous state-of-the-art solution does. Fortunately, we figured that a universal hash circuit is a sufficient and much more efficient alternative. We next give the definition of universal hash functions, and then we discuss how we achieve both collision-free and hiding properties with a universal hash circuit. Finally, we present an efficient instantiation with proper parameters.

DEFINITION 2 (UNIVERSAL HASH). A collection of hash functions $\mathcal{H} = \{h|h : A \rightarrow B\}$ is called universal if for any distinct $x, y \in A$, the probability that a uniformly chosen $h \in \mathcal{H}$ satisfies that $h(x) = h(y)$ is at most $1/|B|$.

3.2.1 Collision-Free Property

This property comes naturally with universal hash functions. Indeed, Definition 2 shows that for any distinct x, y , if they are *fixed* before h is *uniformly* chosen, their hashes are unlikely to collide. This suggests that the collision-free property can be achieved by letting GEN commit to (fix) her inputs before a hash function is jointly (uniformly) picked. Later, the consistency of GEN's inputs can be verified by checking the consistency of the auxiliary outputs (the hashes). Since both the objective circuit and the auxiliary circuit share the same input from GEN, GEN's input consistency to the auxiliary circuits implies the same to the objective circuits. Our protocol is outlined as follows:

1. GEN commits to her inputs $x^{(1)}, x^{(2)}, \dots, x^{(\sigma)}$, where $x^{(j)}$ denotes her input to the j -th garbled circuit.
2. GEN and EVAL jointly and uniformly pick $h \in \mathcal{H}$.
3. GEN constructs σ copies of the garbled circuit. Each circuit contains two parts: the objective circuit and the auxiliary circuit. While the first part computes the objective function, the j -th auxiliary circuit uses the input wires of the objective circuit to compute $h(x^{(j)})$.
4. EVAL asks to check the correctness of a random fraction of the garbled circuits. If the check fails, EVAL aborts; otherwise, EVAL asks GEN to decommit to her inputs for the remaining (unchecked) circuits.
5. EVAL first evaluates the remaining auxiliary circuits. If the evaluation outputs (the hashes) are not consistent, EVAL aborts; otherwise, EVAL proceeds to evaluating the remaining objective circuits.

Since the cut-and-choose technique instructs a majority operation at the end, the final evaluation output will not be influenced by a few inconsistent inputs introduced by a malicious GEN. We next argue, at a high level, that with high probability, this protocol enjoys the desired security property that GEN's inputs to the majority of the remaining circuits are consistent. Indeed, if a malicious GEN is able to pass the hash consistency check and provide inconsistent inputs to the majority of the remaining objective circuits, there are only three possibilities: (1) The (auxiliary) circuits are faulty: The cut-and-choose technique ensures that with high probability, this only happens to a minority of the remaining circuits. (2) GEN is really lucky to have found a collision: By Definition 2, this happens with probability at most $1/|B|$, which becomes negligible if B is properly chosen.

(3) GEN is able to break the binding property of the commitments: Note that since universal hash functions do not even provide pre-image resistance, given h and $h(x^{(i)})$, it can be easy to find $x^{(j)}$ such that $h(x^{(i)}) = h(x^{(j)})$. Thus, if GEN is able to open the commitment from Step 1 to some value computed after h is chosen in Step 2, she breaks the desired security property. However, this would imply that GEN is able to break the commitment scheme's binding property. This happens with negligible probability too.

REMARK 3.1. *Since the above protocol is not the final version of our solution, we only provide the intuitions for now. A simpler and more elaborate protocol will be given in Figure 2, but the same outline and security argument will apply.*

3.2.2 Hiding Property

A deterministic universal hash $h : A \rightarrow B$ provides the collision-free property we need, yet it is insufficient for the purposes here due to the lack of the hiding property. Indeed, if the size of A is small, EVAL could exhaustively compute the hash of all possibilities in A and then deduce $x^{(j)} \in A$ from $h(x^{(j)})$. As a result, the hash function has to be randomized. If the hashes are pseudo-random, they reveal little information about the input, which is the hiding property we desire. In particular, the celebrated left-over-hash lemma [7] (LHL for short, omitted here for space) states that the output of a uniformly picked universal hash function h is pseudo-random (even if h is made public) as long as the input has enough (min-)entropy. As a consequence, all we need to do is introduce entropy to the input of the auxiliary circuit. We suggest that objective function $f(x, y) \mapsto (f_1, f_2)$ is converted to $g(x||r, y) \mapsto (f_1, h(x||r)||f_2)$, where r is a proper randomness picked by GEN at the beginning and h is a universal hash function uniformly picked *after* GEN commits to her new input $x||r$.

We argue that the introduction of random input r does provide the hiding property even when h is public. Indeed, given $h(x||r)$ and h , as long as r is long enough (has enough entropy), for any x' , there must exist r' such that $h(x||r) = h(x'||r')$. This shows that fixing $h(x||r)$ does not rule out any possibilities of x .

Efficient Instantiation.

We suggest the use of the matrix universal hash family

$$\mathcal{M}_{k,m} = \{h_M \mid h_M(x) = M \cdot x \text{ for some } M \in \{0, 1\}^{k \times m}\},$$

for some $m \in \mathbb{N}$. A nice property of this hash family is that it is $\oplus$ -homomorphic, that is, for any $x, y \in \{0, 1\}^m$ and $h_M \in \mathcal{M}_{k,m}$, it holds that $h_M(x \oplus y) = h_M(x) \oplus h_M(y)$. This homomorphism allows a very efficient instantiation of the protocol outline presented in Section 3.2.1 with the following twists:

1. GEN commits not to $x^{(j)}$ directly but to $x^{(j)} \oplus \pi^{(j)}$ and $\pi^{(j)}$ instead, where $\pi^{(j)}$ is uniformly picked from $\{0, 1\}^m$.
2. GEN provides not a garbled circuit that lets EVAL compute $h(x^{(j)})$ obviously but rather $h(\pi^{(j)})$ that GEN computes locally.
3. Let S be the indices of the evaluation circuits.

- For $j \in [s] \setminus S$, EVAL checks not the correctness of the j -th garbled circuit but the correctness of $h(\pi^{(j)})$ (by asking GEN to decommit to $\pi^{(j)}$ first).

- For $j \in S$, EVAL learns $h(x^{(j)})$ not via evaluating the j -th garbled circuit but via asking GEN to decommit to $y^{(j)} = x^{(j)} \oplus \pi^{(j)}$ and then computing $h(y^{(j)}) \oplus h(\pi^{(j)})$.

The main goal of the above twist is to replace the garbled circuit that computes $h(x^{(j)})$ with $h(\pi^{(j)})$ that is locally computed while maintaining the main structure of the protocol, that is, letting GEN commit to her inputs before h is jointly picked and letting EVAL learn the hash of the input to the evaluation circuits. The complete protocol of our GEN's input consistency check is presented in Figure 2, and its security is stated in Lemma 3.

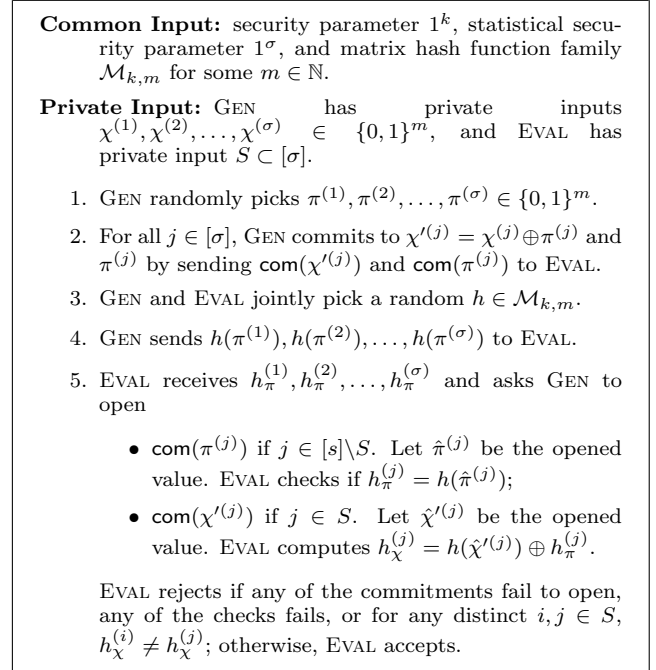


Figure 2: A proof for GEN's input consistency

LEMMA 3 (GEN'S INPUT CONSISTENCY CHECK). *Let $k, \sigma \in \mathbb{N}$ be common inputs. GEN has input $\{\chi^{(1)}, \chi^{(2)}, \dots, \chi^{(\sigma)}\}$, and EVAL has input $S \subset [\sigma]$. If $|S| = c \cdot \sigma$ for some $1 > c > 0$ and EVAL accepts the proof presented in Figure 2, the probability that no $\chi^{(j)}$ appears more than $|S|/2$ times in $\{\chi^{(j)}\}_{j \in S}$ is at most $2^{-O(k+\sigma)}$.*

REMARK 3.2. k in Lemma 3 is the security parameter for the commitment scheme used in the protocol presented in Figure 2.

REMARK 3.3. We chose to let EVAL have a total control over S , the set of circuits to evaluate. This saves the effort of jointly picking S , but still guarantees that whatever S that EVAL picks will not compromise GEN's privacy. Lemma 3 holds as long as S is a constant fraction of $[\sigma]$. However, EVAL is motivated to use the optimal $c = 2/5$ as suggested by shelat and Shen [21].

Finally, we suggest the parameters for security level 2^{-k} . By Definition 2, GEN cannot find a collision with probability better than $1/|B|$. So $|B|$ needs to be at least 2^k . As to the size of random input r , the LHL lemma shows that if the

min-entropy of $x||r$ is at least $k + 2k = 3k$ and the output hash is k -bits long, then the output is indistinguishable from a truly random k -bit string with probability at least $1 - 2^{-k}$. Since we make no assumptions about the input distribution of x , a simple approach is to uniformly pick r such that $|r| = 3k$. We can do better by exploiting specific properties of $\mathcal{M}_{k,m}$ and reach the same goal with $|r| = 2k + \lg(k)$ as shown in Lemma 4.

LEMMA 4. Let X_n denote $\{0,1\}^n$ for some $n \in \mathbb{N}$ and $\mathcal{M}_{k,m}$ be defined as above. For any $x \in X_n$ and any $t \geq 2k + \lg(k)$, distributions $\{(h, h(x||r))\}$ and $\{(h, y)\}$ are statistically indistinguishable with probability at least $1 - 2^{-k}$, where h, r , and y are uniformly chosen from $\mathcal{M}_{k,n+t}$, X_t , and X_k , respectively.

Proof omitted for space.

3.3 Selective Failure Attack

A subtle *selective failure attack* possible in the presence of malicious adversaries has been pointed out by Mohassel and Franklin [16] and independently by Kiraz and Schoenmakers [10]. This attack occurs when a malicious GEN assigns (K_0, K_1) to an EVAL's input wire in the garbled circuit while using $(K_0, K_1^* \neq K_1)$ instead in the corresponding OT. Consequently, if EVAL's input is 1, she learns K_1^* , gets stuck during the evaluation, and so GEN eventually learns that EVAL's input is 1.

Lindell and Pinkas [13] suggested that EVAL picks $M \in \{0,1\}^{n \times m}$ for some $m \in \mathbb{N}$ and computes her new input $\bar{y} \in \{0,1\}^m$ such that $M \cdot \bar{y} = y$. An auxiliary circuit will later convert $\bar{y}$ back to y to use as input to the original circuit. The insight is that selective failures allow GEN to probe some partial information of $\bar{y}$. For this approach to work, the security property needed here is given as follows:

DEFINITION 5. For some $m, n, k \in \mathbb{N}$, $M \in \{0,1\}^{n \times m}$ is called k -probe-resistant if for any non-empty $L \subset [n]$, the Hamming distance of $\bigoplus_{i \in L} M_i$ is at least k , where M_i is the i -th row of M .

As long as M is k -probe-resistant, a malicious GEN will have to successfully probe k bits of $\bar{y}$ in order to gain any partial information of y , the probability of which is negligible. [13] shows that when M is uniformly chosen from $\{0,1\}^{n \times m}$ where $m = \max(4n, 8k)$, the probability that M fails to be k -probe-resistant will be negligible. We stress that matrix M can even be made public so that the auxiliary circuit could consist of only XOR-gates, which requires no communication overhead and can be computed efficiently when combining with the free-XOR trick [11]. In short, not only does this approach elegantly reduce the selective failure attack problem to the classic error correcting code construction, it also has the potential to incur only little overhead.

Our idea to construct a k -probe-resistant matrix is to use a maximum distance separable code such as Reed-Solomon codes.² We will work on the Reed-Solomon code over $\mathbb{F}_{2^t}$ for some $t \in \mathbb{N}$ with codeword size N and message size K .

²We acknowledge that this direction has indeed been mentioned in the original work by Lindell and Pinkas: "an explicit construction can be achieved using any explicit linear code. [13]" However, we believe an explicit solution that enjoys the optimal performance—optimal asymptotic complexity with a constant factor of one—is worth-reporting.

By optimizing parameters, we prove the following in the full version of this paper:

LEMMA 6. Suppose $K \geq (\lg(n) + n + k)/t$. Let $P_1, P_2, \dots, P_n$ be distinct, non-zero polynomials with degree at least $K - 1$ uniformly picked from $\mathbb{F}_{2^t}[x]$. The probability that there exist some $i \in [n]$ and some $L \subset [n] \setminus \{i\}$ such that $P_i = \sum_{j \in L} P_j$ is at most 2^{-k} .

Finally, we describe and implement an algorithm to find a k -probe-resistant M with high probability and prove the correctness of this algorithm as follows:

THEOREM 7. With probability at least $1 - 2^{-k}$, the algorithm presented in Figure 5 outputs a k -probe-resistant $M \in \{0,1\}^{n \times m}$ such that $m \in \mathbb{N}$ and $m \leq \lg(n) + n + k + k \cdot \max(\lg(4n), \lg(4k))$.

Proof omitted.

4. THE MAIN PROTOCOL

Our presentation uses the permuted garbled truth table technique [15] (also known as the *point-and-permute* technique in literature). Briefly, this technique suggests to assign each wire w_i an extra random permutation bit π_i . The circuit garbling and evaluating is then slightly modified: for GEN, each garbled truth table is constructed in the way that its entries are permuted according to its input wires' permutation bits; and for EVAL, each random key now comes with a *locator* that helps EVAL identify the right entry in the garbled truth table while evaluating it.

For each wire, EVAL learns exactly one key-locator pair out of the two assigned to that wire from the circuit evaluation, and the learned locator is in fact the evaluation result of that wire one-time padded with the permutation bit. This property ensures that EVAL is oblivious to the intermediate result of the circuit evaluation and allows EVAL to learn the output by revealing the permutation bits assigned to circuit-output wires.

Common Input: security parameter 1^k , statistical security parameter 1^σ , symmetric cipher (**enc, dec**) with semantic security, (perfectly-hiding) commitment scheme **com**, and objective function $f : (x, y) \mapsto (f_1, f_2)$.

Private Input: GEN has input x and EVAL has input y .

Private Output: GEN receives f_1 and EVAL receives f_2 .

1. (**New Inputs**) GEN uniformly picks $e \in \{0,1\}^{|f_1|}$ (as the one-time pad for f_1) and $r \in \{0,1\}^{2k + \lg(k)}$ (as her random input for computing the universal hash). EVAL samples a k -probe-resistant matrix M and computes $\bar{y}$ such that $M \cdot \bar{y} = y$. From now on, GEN's input refers to $\bar{x} = x||e||r = \bar{x}_1\bar{x}_2 \dots \bar{x}_{m_1}$ and EVAL's input refers to $\bar{y} = \bar{y}_1\bar{y}_2 \dots \bar{y}_{m_2}$. Let $\bar{x}_i$ and $\bar{y}_i$ denote the i -th bit of $\bar{x}$ and $\bar{y}$, respectively.

REMARK 4.1. By definition, k -probe-resistant matrix M has to have full (row) rank. So for any y , there must exist some $\bar{y}$ such that $M \cdot \bar{y} = y$. Moreover, given y and M , $\bar{y}$ can be efficiently computed by Gaussian elimination.

2. (**Pick the randomness**) For all $j \in [\sigma]$, GEN picks randomness $\rho^{(j)}$ for the j -th garbled circuit.

REMARK 4.2. Randomness $\rho^{(j)}$ can be considered as either a pool of truly random bits or a truly random seed to a pseudo-random number generator. It has internal states that keep track of used and fresh random bits, and it always returns fresh random bits when it is used. We stress that when later requested, GEN needs to reveal $\rho^{(j)}$'s initial state so that EVAL will be able to regenerate the random bits that GEN used to construct the j -th garbled circuit.

3. **(Fix GEN's input)** For all $j \in [\sigma]$ and $i \in [m_1]$, GEN uses $\rho^{(j)}$ to compute $(K_{i,0}^{(j)}, K_{i,1}^{(j)}, \pi_i^{(j)}) \in \{0, 1\}^{2k+1}$. Let $W_{i,b}^{(j)}$ denote the key-locator pair $(K_{i,b}^{(j)}, b \oplus \pi_i^{(j)})$. $W_{i,b}^{(j)}$ is called the *label* corresponding to wire w_i of value b in the j -th garbled circuit hereafter. GEN commits to her input by sending $\{\Gamma^{(j)}\}_{j \in [\sigma]}$ to EVAL, where $\Gamma^{(j)} = \{\text{com}(W_{i,\bar{x}_i}^{(j)}; \gamma_i^{(j)})\}_{i \in [m_1]}$. Moreover, GEN uses $\rho^{(j)}$ to commit to both labels assigned to $\{w_i\}_{i \in [m_1]}$ by sending $\{\Theta^{(j)}\}_{j \in [\sigma]}$ to EVAL, where
- $$\Theta^{(j)} = \{\text{com}(W_{i,0 \oplus \pi_i^{(j)}}^{(j)}; \theta_i^{(j)}), \text{com}(W_{i,1 \oplus \pi_i^{(j)}}^{(j)}; \theta_i^{(j)})\}_{i \in [m_1]}.$$

REMARK 4.3. It is crucial that commitments $\Gamma^{(j)}$ cannot use the randomness from $\rho^{(j)}$ like commitments $\Theta^{(j)}$ do. If they do, EVAL will learn GEN's inputs to all check circuits.

4. **(Determine the objective circuit)** EVAL first reveals M , and then both parties jointly run a coin flipping protocol to uniformly pick $H \in \{0, 1\}^{k \times m_1}$. Both parties now have determined the objective circuit C that computes $g : (\bar{x}, \bar{y}) \mapsto (\perp, (c, g_2))$, where $\bar{x} = x||e||r$, $y = M \cdot \bar{y}$, $g_1 = f_1(x, y)$, $c = g_1 \oplus e$, and $g_2 = f_2(x, y)$.
5. **(Commit to input/output label pairs)** Let $\{w_i\}_{i \in [m_1]}$ be the wires corresponding to GEN's input, $\{w_{m_1+i}\}_{i \in [m_2]}$ be those corresponding to EVAL's input, and $\{w_i\}_{i \in O_1}$ be those corresponding to the first part of EVAL's output (output c in particular). GEN uses randomness $\rho^{(j)}$ to

- (a) compute $(K_{i,0}^{(j)}, K_{i,1}^{(j)}, \pi_i^{(j)}) \in \{0, 1\}^{2k+1}$ for all wires but those corresponding to GEN's input³ and
- (b) commit to the label pairs assigned to $\{w_i\}_{i \in [m_2] \cup O_1}$ by sending $\{\Omega^{(j)}, \Phi^{(j)}\}_{j \in [\sigma]}$ to EVAL, where

$$\Omega^{(j)} = \{\text{com}(W_{m_1+i,0}^{(j)}; \omega_i^{(j)}), \text{com}(W_{m_1+i,1}^{(j)}; \omega_i^{(j)})\}_{i \in [m_2]},$$

$$\Phi^{(j)} = \{\text{com}(W_{i,0}^{(j)}), \text{com}(W_{i,1}^{(j)})\}_{i \in O_1}.$$

REMARK 4.4. GEN commits to the label pairs assigned to GEN's input wires (in Step 3) and EVAL's input wires so that when EVAL later receives proper decommitments, she will know that the decommitted labels are valid. GEN also commits to the label pairs assigned to GEN's output wires, and these commitments are for GEN's output authenticity proof.

³The random keys assigned to the wires corresponding to GEN's input were already computed in Step 3.

Moreover, the commitment pairs corresponding to GEN's input wires need to be randomly swapped so that each label's semantics is independent of its location. In other words, the location of a successfully decommitted label will not disclose GEN's input to EVAL. This random swap is done by reusing the permutation bit $\pi_i^{(j)}$ that is assigned to each wire and used to permute entries in garbled truth tables. In contrast, the commitment pairs corresponding to EVAL's input wires, need to follow a known order so that EVAL can know the semantics of her input labels and can verify that the successfully decommitted labels actually match her input. As a result, the commitment pairs in $\Theta^{(j)}$ are randomly swapped so that the commitment to b -label of the i -th wire is the $(2 \cdot i + b \oplus \pi_i^{(j)})$ -th entry, whereas in $\Omega^{(j)}$, the commitment to b -label of the i -th wire is the $(2 \cdot i + b)$ -th.

The order of commitments in $\Phi^{(j)}$ simply follows GEN's output authenticity proof presented in Figure 1.

6. Both parties jointly execute $(m_2 + \sigma)$ instances of $\binom{2}{1}$ -OTs. In particular,

- (a) **(Eval's Input OTs)** For each $i \in [m_2]$, both parties run a $\binom{2}{1}$ -OT in which GEN's input equals

$$\left(\{(W_{m_1+i,0}^{(j)}, \omega_i^{(j)})\}_{j \in [\sigma]}, \{(W_{m_1+i,1}^{(j)}, \omega_i^{(j)})\}_{j \in [\sigma]} \right)$$

and EVAL's input equals $\bar{y}_i$. Let $Y^{(j)}$ denote the set of decommitments EVAL received for the j -th garbled circuit, that is, $Y^{(j)} = \{(W_{m_1+i,\bar{y}_i}^{(j)}, \omega_i^{(j)})\}_{i \in [m_2]}$.

- (b) **(Circuit OTs)** EVAL randomly picks $S \subset [\sigma]$ such that $|S| = 2\sigma/5$. Let $s \in \{0, 1\}^\sigma$ such that $s_j = 1$ if $j \in S$; or $s_j = 0$ otherwise. If $s_j = 0$, EVAL will learn the randomness and check the j -th circuit; otherwise, EVAL will retrieve GEN's input labels and evaluate the j -th circuit. More specifically, for each $j \in [\sigma]$, both parties run a $\binom{2}{1}$ -OT in which EVAL's input equals s_j and GEN's input equals $(\rho^{(j)}, (X_1^{(j)}, X_2^{(j)}, h_\pi^{(j)}))$, where
- $$X_1^{(j)} = \{(W_{i,\bar{x}_i}^{(j)}, \gamma_i^{(j)})\}_{i \in [m_1]}, X_2^{(j)} = \{(W_{i,\bar{x}_i}^{(j)}, \theta_i^{(j)})\}_{i \in [m_1]},$$
- and $h_\pi^{(j)} = H \cdot (\pi_1^{(j)} || \pi_2^{(j)} || \dots || \pi_{m_1}^{(j)})$.

Either party aborts if any $\binom{2}{1}$ -OT fails.

REMARK 4.5. The above $\binom{2}{1}$ -OTs could run in parallel if they provide security for parallel execution.

REMARK 4.6. We chose to let EVAL have a total control over S , the set of circuits to evaluate. On one hand, both parties save the efforts to jointly pick S . On the other hand, our solution has the property that whatever S that EVAL picks will not compromise GEN's privacy.

REMARK 4.7. Decommitments $X_1^{(j)}$ are for EVAL to retrieve GEN's input labels from commitments $\Gamma^{(j)}$, which EVAL received in Step 3; decommitments $X_2^{(j)}$ are for EVAL to retrieve GEN's input labels from commitments $\Theta^{(j)}$, which EVAL received in Step 3; and decommitments $Y^{(j)}$ are for EVAL to retrieve EVAL's input labels from commitments $\Omega^{(j)}$, which EVAL received in Step 5b.

7. **(Circuit Garbling)** For each gate $g : \{0, 1\} \times \{0, 1\} \mapsto \{0, 1\}$ with input wires w_a and w_b and output wire w_c , GEN computes its garbled truth table

$$G(g)^{(j)} = (\langle \pi_a^{(j)}, \pi_b^{(j)} \rangle, \langle \pi_a^{(j)}, 1 \oplus \pi_b^{(j)} \rangle, \langle 1 \oplus \pi_a^{(j)}, \pi_b^{(j)} \rangle, \langle 1 \oplus \pi_a^{(j)}, 1 \oplus \pi_b^{(j)} \rangle),$$

where $\langle b_1, b_2 \rangle = \text{enc}_{K_{a,b_1}}^{(j)}(\text{enc}_{K_{b,b_2}}^{(j)}(W_{c,g(b_1,b_2)}^{(j)}))$.

REMARK 4.8. Note that once $(K_{i,0}^{(j)}, K_{i,1}^{(j)}, \pi_i^{(j)})$ are chosen for every wire w_i , no more randomness is needed for generating the j -th garbled circuit. So $\rho^{(j)}$ is not used here.

8. **(Circuit Checking)** Let $\{w_i\}_{i \in O}$ be the circuit-output wires. GEN then sends $\{G(C)^{(j)}\}_{j \in [\sigma]}$ to EVAL, where

$$G(C)^{(j)} = \left(\{G(g)^{(j)}\}_{g \in C}, \{\pi_i^{(j)}\}_{i \in O} \right).$$

- **(Check Circuits)** For each $j \in [\sigma] \setminus S$, EVAL checks:
 - (a) if $\rho^{(j)}$ received in Step 6b can regenerate commitments $\{\Theta^{(j)}, \Omega^{(j)}, \Phi^{(j)}\}$ received in Step 5b and reconstruct garbled circuit $G(C)^{(j)}$?
 - (b) if $h_\pi^{(j)}$ indeed equals $H \cdot (\pi_1^{(j)} \parallel \pi_2^{(j)} \parallel \dots \parallel \pi_{m_1}^{(j)})$?
- **(Evaluation Circuits)** For each $j \in S$, EVAL checks:
 - (a) if $X_1^{(j)}$ and $X_2^{(j)}$ both received in Step 6b successfully opens $\Gamma^{(j)}$ and half of $\Theta^{(j)}$ both received in Step 3, respectively? and if the decommitted labels match? In particular,
 - i. if the i -th entry of $X_1^{(j)}$ successfully opens the i -th entry of $\Gamma^{(j)}$?
 - ii. if the i -th entry of $X_2^{(j)}$ successfully opens the $(2 \cdot i + \bar{x}_i \oplus \pi_i^{(j)})$ -th entry of $\Theta^{(j)}$?
 - iii. if the i -th decommitted labels from the above two steps coincide?
 - (b) if $Y^{(j)}$ received in Step 6a successfully opens half of the commitments in $\Omega^{(j)}$? In particular, if the i -th entry of $Y^{(j)}$ successfully opens the $(2 \cdot i + \bar{y}_i)$ -th entry of $\Omega^{(j)}$?

EVAL aborts immediately as long as a failure occurs.

REMARK 4.9. For evaluation circuits, the fact that decommitments $X_2^{(j)}$ (resp. $Y^{(j)}$) successfully open $\Theta^{(j)}$ (resp. $\Omega^{(j)}$) shows that the decommitted labels corresponding to GEN's (resp. EVAL's) input to the j -th garbled circuit are valid. Furthermore, the fact that the decommitted labels from $\Gamma^{(j)}$ coincide with those from $\Theta^{(j)}$ shows that GEN indeed commits to her inputs before H is chosen, which is the necessary condition for our 2-universal hash idea to work.

9. **(Circuit Evaluating)** EVAL has now obtained garbled circuit $G(C)^{(j)}$ and $(m_1 + m_2)$ labels corresponding to the circuit-input wires of C . EVAL then evaluates the circuit:

(a) For each gate g with retrieved input labels $W_a^{(j)} = (K_a^{(j)}, \delta_a^{(j)})$ and $W_b^{(j)} = (K_b^{(j)}, \delta_b^{(j)})$, EVAL picks the $(2 \cdot \delta_a^{(j)} + \delta_b^{(j)})$ -th entry E in $G(g)^{(j)}$ and computes output label $W_c^{(j)} = (K_c^{(j)}, \delta_c^{(j)}) = \text{dec}_{K_b^{(j)}}(\text{dec}_{K_a^{(j)}}(E))$.

(b) For each circuit-output wire w_i with corresponding label $W_i^{(j)} = (K_i^{(j)}, \delta_i^{(j)})$, EVAL computes the wire value $b_i^{(j)} = \delta_i^{(j)} \oplus \pi_i^{(j)}$.

EVAL interprets $\{b_i^{(j)}\}$ as $(c^{(j)}, g_2^{(j)})$. Let $Z^{(j)}$ denote the labels that came with $c^{(j)}$.

10. **(Gen's Input Consistency Check)** Let $\{w_i\}_{i \in [m_1]}$ be the wires corresponding to GEN's input and $W_i^{(j)} = (\delta_i^{(j)}, K_i^{(j)})$ be the decommitted label corresponding to w_i in the j -th garbled circuit. EVAL computes the hash of GEN's input

$$h_x^{(j)} = h_\pi^{(j)} \oplus H \cdot (\delta_1^{(j)} \parallel \delta_2^{(j)} \parallel \dots \parallel \delta_{m_1}^{(j)}).$$

EVAL verifies GEN's input consistency by checking if for any $i, j \in S$, $h_x^{(i)} = h_x^{(j)}$. EVAL aborts if any of the checks fails.

REMARK 4.10. The proof for GEN's input consistency presented in Figure 2 is in fact embedded in our main protocol. In particular, Step 3, Step 4, Step 6b, Step 8, and this step constitute that proof. It is worth-mentioning that $\Gamma^{(j)}$ and $\Theta^{(j)}$ here are equivalent to the commitments to $\chi^{(j)} \oplus \pi^{(j)}$ and $\pi^{(j)}$, respectively, in the protocol presented in Figure 2.

11. **(Majority Operation)** Let (c, g_2) be the most common tuple in $\Pi = \{(c^{(j)}, g_2^{(j)})\}_{j \in S}$. EVAL aborts if (c, g_2) is not the majority in Π , that is, (c, g_2) does not appear more than $\frac{|\Pi|}{2} = \frac{\sigma}{5}$ times in Π ; otherwise, EVAL outputs g_2 .
12. **(Gen's Output Authenticity Proof)** The two parties conduct the protocol presented in Figure 1. In particular, the common inputs include security parameter 1^k , statistical security parameter $1^{|S|}$, commitments to label pairs assigned to GEN's output wires $\{\Phi^{(j)}\}_{j \in S}$, and GEN's alleged output c . Also, GEN's input equals $\{W_{i,0}^{(j)}, W_{i,1}^{(j)}\}_{i \in O, j \in S}$ and EVAL's input equals $Z^{(j)}$ for some $j \in S$ such that $c^{(j)} = c$. If the proof fails, GEN aborts; otherwise, GEN outputs $c \oplus e$.

REMARK 4.11. Due to the well-known selective decommitment attack, the commitment scheme needs to be perfectly-hiding so that the unopened commitments remain hiding [5].

THEOREM 8. Assume that the $\binom{2}{1}$ -OT protocol is secure in the presence of malicious adversaries, and there exist a perfectly-hiding commitment scheme, a family of pseudo-random functions, the main protocol (GEN, EVAL) presented above securely computes $f : (x, y) \mapsto (f_1, f_2)$ in the presence of malicious adversaries.

The proof sketch is provided in Appendix B.

5. REDUCING COMMUNICATION COST

In this section, we provide an effective technique that reduces the communication overhead of the cut-and-choose-based protocol by up to 60%. We stress that this is not the first technique that enjoys such an improvement [4, 12], but this is the first one that is compatible with the pipeline technique that vastly increases the scalability of secure two-party computation [6].

Our trick starts from the idea of random seed checking [12]. This idea suggests that GEN sends not the whole check circuit but its hash to EVAL. Since EVAL will learn each check circuit's randomness, she can regenerate the circuit and verify its correctness by comparing its hash with the one received from GEN. As a consequence, only a fixed amount of data needs to be exchanged for a check circuit regardless of the circuit size.

Another useful technique is the pipeline technique. This technique suggests that circuit garbling and evaluating is done in a pipeline manner. The garbled circuits are sent in batches of gates. The i -th batch consists of the i -th garbled gate from all circuits. GEN sends out the current batch immediately after it is generated. While EVAL is working with the current batch, GEN could start generating the next. The advantages include that the idle time is greatly reduced and that minimal memory is needed (since at any moment, only a batch of gates reside in memory).

Our technique nicely combines the advantages of the above two techniques, that is, each check circuit incurs only a small amount of communication overhead, while at the same time, circuit garbling and evaluating can be pipelined. The challenge is that the random seed checking technique requires GEN to handle a check circuit and an evaluation circuit differently, while at the same time, she does not get to know if a circuit is checked or evaluated. The problem can thus be formulated as follows:

GEN wants to send σ numbers $\{g^{(1)}, g^{(2)}, \dots, g^{(\sigma)}\}$ to EVAL, while EVAL actually knows μ of them. How do they reduce the communication overhead while GEN remains oblivious to which numbers EVAL knows?

An interesting trick is that they could treat $\{g^{(j)}\}$ as coefficients of a polynomial, that is, $P(x) = g^{(1)} + g^{(2)}x + \dots + g^{(\sigma)}x^{\sigma-1}$. Although GEN does not know which μ numbers out of $\{g^{(j)}\}$ that EVAL knows, she does know that EVAL only needs $(\sigma - \mu)$ points to fully recover the polynomial. We therefore suggest that GEN sends $P(1), P(2), \dots, P(\sigma - \mu)$ to EVAL. With the μ coefficients she already knew and the $(\sigma - \mu)$ points from GEN, EVAL can efficiently recover all $\{g^{(j)}\}$ with simple polynomial interpolation.

Since our protocol suggests 60%-40% check circuit and evaluation circuit ratio, with a slight increase of the computation overhead due to the polynomial interpolations, this trick reduces the communication overhead by 60%, and more importantly, this trick is compatible with the pipeline technique.

6. EXPERIMENTAL RESULTS

In this section, we report empirical evidence of our performance advantages over prior work. We implemented our work on top of the open-source project—KSS [12]. We ran our experiments on the grid Stampede hosted in Texas Ad-

vanced Computing Center. Each instance of the experiments invokes 32 computing nodes; each node has 32GB memory and two 2 Intel Xeon E5-2680 2.7G processors; and each processor has 8 cores.

Performance of our Proposed Technique:

We show the performance of our GEN's input consistency check and k -probe-resistant matrix generating algorithm compared with the prior state-of-the-art in Figure 3.

We first compare the performance of the KSS system with that of the KSS system integrated with our GEN's input consistency check. This experiment is conducted by using both systems to evaluate circuits of various input sizes. These circuits compute 2^n blocks of AES128 encryption, where $n = 1, 2, \dots, 10$ in which GEN provides inputs for 2^n blocks of AES128 (and thus has a 2^{n+7} -bit input), EVAL provides a 128-bit encryption key, and EVAL receives the 2^{n+7} -bit ciphertext. Figure 3a shows that when the input size increases only to a moderate level (2^{17}), the performance gap due to GEN's input consistency check has almost dominated the whole protocol execution. Specifically, the wall-clock running of the improved protocol is 48.8 seconds versus 92.1 seconds for KSS (which was the fastest published protocol whose results we could replicate on our setup).

Next, we show in Figure 3b that as GEN's input size increases, the ratio between the width and height of the k -probe-resistant matrices generated by our algorithm indeed approximates 1, while the ratio of those generated by Lindell and Pinkas's approach remains constant 4. This comparison suggests that for a circuit that has OTs as the dominant component, the overall protocol execution time could be reduced to 25% simply by replacing the k -probe-resistant matrix generated by the original work with the one generated by our algorithm.

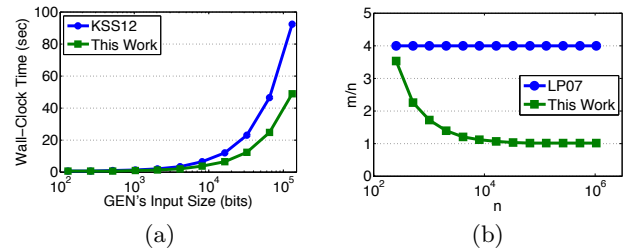


Figure 3: Performance comparison with prior works

Performance of the Main Protocol:

We show in Figure 4 the overall execution time of our system securely evaluating circuits EDT-4095⁴, RSA-256⁵, and 1024-AES128. Overall, our system is able to handle 650,000+ (or $\sim 200,000$ non-XOR) gates per second. We also observe that for all three circuits that we evaluated, more than 60% of the execution time is spent on communicating the huge amount of data, the garbled circuits. If we consider only the circuit garbling, the rate that our system actually achieves could be as high as 1,600,000+ (or 500,000+ non-XOR) gates per second, with the help of var-

⁴This circuit computes the edit distance of two 4,095-bit inputs.

⁵This circuit computes a 256-bit modular exponentiation.

		Gen (sec)		Eval (sec)	Comm (MB)
OT	comp comm	0.4±0.09%	1%	0.3±0.6%	6
cut-& chk	comp comm	—	—	—	9
Inp. Chk	comp comm	0.8± 0.3±	1% 1%	0.3±0.2% 0.9± 1%	2,008
Evl.	comp comm	11.4± 9.2±	0.6% 1%	28.0±0.4% 30.3±0.8%	72,271
Total	comp comm	12.6± 9.6±	0.3% 1%	28.0±0.2% 31.5±0.4%	74,294

Table 3: The 95% two-sided confidence intervals of the computation and communication time for each stage in the 1024-AES128 experiment $(x, y) \mapsto (\perp, 1024\text{-AES128}_y(x))$.

ious optimization techniques, including SSE2 and AESNI instruction sets, and the free-XOR technique.

circuit	gates	(non-XOR)	time (sec)	comm.
EDT-4095	5.9B	(2.4B)	9,042	18 TB
RSA-256	0.93B	(0.33B)	1,437	3 TB
1024-AES128	32M	(9.3M)	49	74 GB

Figure 4: The performance of our main protocol with $k = 80$ and $\sigma = 256$. All numbers in “time” column come from an average of 30 data points and have the 95% confidence interval $< 1\%$.

It is also worth-mentioning that our system has not reached its full potential yet. The communication overhead is expected to drop to 40% of the reported once the technique presented in Section 5 is integrated. An interesting future task is to explore the computational price paid, due to the polynomial interpolation, for those 60% savings.

7. REFERENCES

- [1] I. Damgård, V. Pastro, N. Smart, and S. Zakarias. Multiparty Computation from Somewhat Homomorphic Encryption. CRYPTO '12, 2012. <http://eprint.iacr.org/2011/535>.
- [2] O. Goldreich. *Foundations of Cryptography: Volume 2, Basic Applications*. Cambridge University Press, 2004.
- [3] O. Goldreich, S. Micali, and A. Wigderson. How to Play any Mental Game. STOC '87, pp. 218–229.
- [4] V. Goyal, P. Mohassel, and A. Smith. Efficient Two-Party and Multiparty Computation against Covert Adversaries. EUROCRYPT'08, pp. 289–306. Springer-Verlag.
- [5] D. Hofheinz. Possibility and Impossibility Results for Selective Decommitments. *J. Cryptol.*, 24(3):470–516, July 2011.
- [6] Y. Huang, D. Evans, J. Katz, and L. Malka. Faster Secure Two-Party Computation using Garbled Circuits. USENIX SEC'11, pp. 35–35.
- [7] R. Impagliazzo and D. Zuckerman. How to Recycle Random Bits. SFCS '89.
- [8] S. Jarecki and V. Shmatikov. Efficient Two-Party Secure Computation on Committed Inputs. EUROCRYPT '07, pp. 97–114.
- [9] M. Kiraz. *Secure and Fair Two-Party Computation*. PhD thesis, Technische Universiteit Eindhoven, 2008.
- [10] M. Kiraz and B. Schoenmakers. A Protocol Issue for The Malicious Case of Yao's Garbled Circuit Construction. In

- 27th Symposium on Information Theory in the Benelux, 2006.
- [11] V. Kolesnikov and T. Schneider. Improved Garbled Circuit: Free XOR Gates and Applications. ICALP '08, pp. 486–498.
- [12] B. Kreuter, a. shelat, and C. Shen. Billion-Gate Secure Computation with Malicious Adversaries. USENIX SEC'12, 2012.
- [13] Y. Lindell and B. Pinkas. An Efficient Protocol for Secure Two-Party Computation in the Presence of Malicious Adversaries. EUROCRYPT '07.
- [14] Y. Lindell and B. Pinkas. Secure Two-Party Computation via Cut-and-Choose Oblivious Transfer. TCC'11, pp. 329–346.
- [15] D. Malkhi, N. Nisan, B. Pinkas, and Y. Sella. Fairplay: A Secure Two-Party Computation System. USENIX SEC'04, volume 13, pp. 287–302.
- [16] P. Mohassel and M. Franklin. Efficiency Tradeoffs for Malicious Two-Party Computation. PKC'06, pp. 458–473.
- [17] P. Mohassel and B. Riva. Garbled Circuits Checking Garbled Circuits: More Efficient and Secure Two-Party Computation, 2013. <http://eprint.iacr.org/2013/051>.
- [18] J. Nielsen and C. Orlandi. LEGO for Two-Party Secure Computation. TCC'09, volume 5444 of LNCS, pages 368–386.
- [19] J. B. Nielsen, P. S. Nordholt, C. Orlandi, and S. S. Burra. A New Approach to Practical Active-Secure Two-Party Computation. CRYPTO '12, 2012.
- [20] B. Pinkas, T. Schneider, N. P. Smart, and S. C. Williams. Secure Two-Party Computation is Practical. ASIACRYPT '09, pp. 250–267.
- [21] a. shelat and C.-H. Shen. Two-Output Secure Computation with Malicious Adversaries. EUROCRYPT'11, pp 386–405.
- [22] Stampede. Tacc stampede. <http://www.tacc.utexas.edu/user-services/user-guides/stampede-user-guide>.
- [23] D. Woodruff. Revisiting the Efficiency of Malicious Two-Party Computation. EUROCRYPT'07, volume 4515 of LNCS, pp. 79–96.
- [24] A. C. Yao. Protocols for Secure Computations. SFCS '82, pp. 160–164.

APPENDIX

A. ALGORITHMS AND PROTOCOLS

Input: EVAL's input size n and a security parameter 1^k
Output: A k -probe-resistant $M \in \{0, 1\}^{n \times m}$ for some $m \in \mathbb{N}$

```

1 begin
2    $t \leftarrow \lceil \max(\lg(4n), \lg(4k)) \rceil$ 
3   while  $2^{t-1} > k + (\lg(n) + n + k)/(t - 1)$  do
4      $t \leftarrow t - 1$ ;
5   end
6    $K \leftarrow \lceil (\lg(n) + n + k)/t \rceil$ ;
7    $N \leftarrow K + k - 1$ ;
8   for  $i \leftarrow 1$  to  $n$  do
9     Pick  $P(x) = \sum_{i=0}^{K-1} a_i x^i$ , where  $a_i \leftarrow_R \mathbb{F}_{2^t}$ ;
10     $M_i \leftarrow [P(1)_2 || P(2)_2 || \dots || P(N)_2]$ 
11  end
12  return  $M$ ; //  $M \in \{0, 1\}^{n \times m}$ , where  $m = Nt$ 
13 end

```

REMARK A.1. Line 2-5 is to find the minimum t such that $2^t \geq k + (\lg(n) + n + k)/t$, and $P(i)_2$ denotes a $t \times 1$ row vector over $\{0, 1\}$.

Figure 5: A probabilistic algorithm to generate a k -probe-resistant matrix $M \in \{0, 1\}^{n \times m}$ for some $m \in \mathbb{N}$.

B. MAIN THEOREM PROOF SKETCH

B.1 Malicious Generator P_1^*

S_1 first randomly picks a k -probe-resistant M and a fake input y' . Then S_1 finds a pre-image $\bar{y}'$ such that $M \cdot \bar{y}' = y'$. This $\bar{y}'$ is used as input to EVAL's input OTs. The OT's receiver security ensures that P_1^* (as the OT sender) cannot distinguish the OT receiver's input being $\bar{y}'$ provided by S_1 in the ideal model from being $\bar{y}$ provided by P_2 in the real model.

Next, for the circuit OTs, S_1 invokes the OT's simulator (the existence of which is guaranteed by OT's security) to extract both inputs from P_1^* , including randomness $\rho^{(j)}$, decommitments $(X_1^{(j)}, X_2^{(j)})$ to GEN's input keys, and hash $h_\pi^{(j)}$. A garbled circuit is *bad* if the retrieved randomness cannot be used to regenerate the provided commitments and garbled circuit. S_1 aborts if more than $\sigma/5$ of the garbled circuits are bad. This step is indistinguishable from the real model due to the cut-and-choose technique. In particular, if more than $\sigma/5$ circuits are bad, P_2 in the real model would abort with high probability too since the probability that none of the bad circuits is checked is negligible.

If S_1 does not abort, it learns the randomness of at least $4\sigma/5$ good garbled circuits. Note that after P_1^* passes the circuit checking, S_1 also learns the decommitments $X^{(j)}$ of the $2\sigma/5$ evaluation circuits. In other words, S_1 learns the randomness of at least $\sigma/5$ evaluation circuits, which are good circuits too. The binding property of the commitments ensures that S_1 learns the private inputs that P_1^* provided to those good evaluation circuits. S_1 aborts if these private inputs are inconsistent. This step is indistinguishable from the real model due to GEN's input consistency check. In particular, the input consistency check ensures that the probability of good evaluation circuits having inconsistent inputs is negligible.

Let P_1^* 's private input extracted from above be $\bar{x}' = x' || r' || e'$. S_1 sends x' to the external trusted party and gets $f_1(x', y)$ in return, where $y = M \cdot \bar{y}$. If S_1 in the ideal model (resp. P_2 in the real model) has come to this far, with high probability, P_1^* must have provided majority good evaluation circuits with valid input labels corresponding to P_1^* 's input $\bar{x}'$ and S_2 's input $\bar{y}'$ (resp. P_2 's input $\bar{y}$). So the majority of P_2 's evaluation outputs are exactly $f_1(x', y) \oplus e'$. This implies that P_2^* cannot distinguish S_1 on input $\bar{y}'$ but providing $f_1(x', y) \oplus e'$ (from the external party) versus P_1 on input $\bar{y}$ and providing the evaluation output $f_1(x', y) \oplus e'$. Moreover, the k -probe-resistant matrix also helps to support the indistinguishability between S_1 on protocol input y' and P_1 on protocol input y . In particular, the k -probe-resistant matrix ensures that the difference between the probability that S_1 on fake input y' aborts (due to selective failure) and the probability that P_2 on real input y aborts is negligible.

Finally, since S_1 knows the randomness of many circuits, it also knows the random keys corresponding to P_1^* 's encrypted output $f_1(x', y) \oplus e'$. S_1 is able to complete the GEN's output authenticity proof as P_1 does.

B.2 Malicious Evaluator P_2^*

Simulator S_2 honestly follows the main protocol all the way until the step of OTs except that S_2 picks a random $\bar{x}' \in \{0, 1\}^{m_1}$ at the beginning and uses this fake input as input. In particular, S_2 commits to fake input $\bar{x}'$ in Step 3 by

committing to the corresponding labels. The commitments are denoted by $\{\Gamma^{(j)}\}_{j \in [\sigma]}$. In the step of OTs, S_2 invokes OT's simulator S_2^{OT} (whose existence is guaranteed by OT's security) to extract P_2^* 's input to OTs, that is, her private input $\bar{y}'$ to EVAL's input OTs and the choice string s' to the circuit OTs. Recall that s' determines the check circuits and evaluation circuits. Next, S_2 externally invokes the trusted third party with input $y' = M \cdot \bar{y}'$ and gets $f_2(x, y')$ in return. Note that M is the k -probe-resistant matrix received from P_2^* before OTs. After this, if $s'_j = 0$, the j -th garbled circuit is a check circuit and needs to be honestly constructed according to objective circuit C ; otherwise, the j -th garbled circuit is an evaluation circuit and is constructed in a way that it always outputs $(h', c', f_2(x, y'))$, where h' and c' are randomly picked by S_2 . From now on, S_2 follows the Main protocol faithfully. Finally, if S_2 accepts in the step of GEN's Output Authenticity Proof, it sends 1 to the external oracle so that S_1 gets $f_1(x, y')$; otherwise, S_2 sends 0 to the external oracle so that S_1 gets $\perp$.

Here we argue at a high level that P_2^* cannot distinguish S_2 using fake input $\bar{x}'$ in the ideal model versus P_1 using real input $\bar{x}$ in the real model.

For check circuits, the only difference between S_2 in the ideal model and P_1 in the real model is the committed message in $\Gamma^{(j)}$. Note that this commitment is never opened for check circuits. Therefore, if P_1^* is able to distinguish S_2 committing to fake input $\bar{x}'$ from P_1 committing to real input $\bar{x}$ in any check circuit, P_1^* is able to break the hiding property of the commitment scheme.

For evaluation circuits, the information P_1^* learned related the other party's input is the location of the decommitted labels within each pair of commitments and the evaluation output:

1. Recall that the pairs of commitments to the labels assigned to GEN's input wires are randomly swapped by permutation bits $\{\pi_i^{(j)}\}_{i \in [m_1], j \in [\sigma]}$. This random swapping implies that the location learned in the ideal model is $\bar{x}' \oplus \pi'^{(j)}$, where $\pi'^{(j)} = \pi_1'^{(j)} || \pi_2'^{(j)} || \dots || \pi_{m_1}'^{(j)}$ while that learned in the real model is $\bar{x} \oplus \pi^{(j)}$, where $\pi^{(j)} = \pi_1^{(j)} || \pi_2^{(j)} || \dots || \pi_{m_1}^{(j)}$. Since $\pi'^{(j)}$ and $\pi^{(j)}$ are independently chosen from the uniform distribution, the location information is statistically indistinguishable.
2. The other message that might give S_2 away is the evaluation output. First, we claim that P_2^* always gets consistent output among different garbled circuits. Indeed, in the real model, since P_1 is assumed to be honest, the outputs from the evaluation circuits are identical, and similarly, in the ideal model, S_2 also generates evaluation circuits that have a fixed output. So it remains to argue that (h, c, f_2) in the real model is indistinguishable from (h', c', f_2') in the ideal model. Intuitively, h and h' are indistinguishable due to the hiding property of our 2-universal hash scheme, c and c' are indistinguishable due to the perfect secrecy of the one-time pad encryption, and f_2 and f_2' are indistinguishable due to the simulation security of OTs.

Common Input: security parameter 1^k , symmetric encryption scheme (enc, dec) with semantic security, and boolean circuit C that computes $f(x, y)$.

Private Input: GEN has private input $x = x_1x_2 \cdots x_{m_1}$ and EVAL has private input $y = y_1y_2 \cdots y_{m_2}$, where x_i and y_i denote the i -th bit of x and y , respectively.

Output: Both GEN and EVAL receive $f(x, y)$ at the end.

1. **(Circuit Garbling)** GEN garbles C as follows:

- (a) GEN picks $(K_{i,0}, K_{i,1}, \pi_i) \in \{0, 1\}^{2k+1}$ at random for each wire w_i . Let $W_{i,b}$ denote the key-locator pair $(K_{i,b}, b \oplus \pi_i)$. $W_{i,b}$ is called the *label* corresponding to wire w_i of value b hereafter.
- (b) For each gate $g : \{0, 1\} \times \{0, 1\} \rightarrow \{0, 1\}$ with input wires w_a and w_b and output wire w_c , GEN computes its garbled truth table

$$G(g) = (\langle \pi_a, \pi_b \rangle, \langle \pi_a, 1 \oplus \pi_b \rangle, \langle 1 \oplus \pi_a, \pi_b \rangle, \langle 1 \oplus \pi_a, 1 \oplus \pi_b \rangle),$$

where $\langle b_1, b_2 \rangle = \text{enc}_{K_{a,b_1}}(\text{enc}_{K_{b,b_2}}(W_{c,g(b_1,b_2)}))$.

Let $\{w_i\}_{i \in O}$ be the circuit-output wires. GEN sends $G(C) = (\{G(g)\}_{g \in C}, \{\pi_i\}_{i \in O})$ to EVAL.

2. **(Input Labels Retrieving)** Let $\{w_i\}_{i \in [m_1]}$ be the wires corresponding to GEN's input, and let $\{w_{m_1+i}\}_{i \in [m_2]}$ be those corresponding to EVAL's input.

- (a) **(Gen's Input Labels)** GEN sends EVAL the labels corresponding to her input $\{W_{i,x_i}\}_{i \in [m_1]}$.
- (b) **(Eval's Input Labels)** For each $i \in [m_2]$, GEN and EVAL execute a $\binom{2}{1}$ -OT in which GEN's input equals $(W_{m_1+i,0}, W_{m_1+i,1})$ and EVAL's input equals y_i .

The above $\binom{2}{1}$ -OTs could all be run in parallel.

3. **(Circuit Evaluating)** EVAL has now obtained garbled circuit $G(C)$ and $(m_1 + m_2)$ labels corresponding to the $(m_1 + m_2)$ circuit-input wires. EVAL evaluates the circuit as follows:

- (a) For each gate g with retrieved input labels $W_a = (K_a, \delta_a)$ and $W_b = (K_b, \delta_b)$, EVAL picks the $(2 \cdot \delta_a + \delta_b)$ -th entry E in $G(g)$ and computes the output label $W_c = (K_c, \delta_c) = \text{dec}_{K_b}(\text{dec}_{K_a}(E))$.
- (b) For each circuit-output wire w_i with corresponding label $W_i = (K_i, \delta_i)$, EVAL computes the wire value $b_i = \delta_i \oplus \pi_i$. Recall that π_i for wire w_i comes with $G(C)$.

Finally, EVAL interprets $\{b_i\}$ as $f(x, y)$ and sends $f(x, y)$ to GEN. Both parties output $f(x, y)$.

Figure 6: The Yao protocol using the point-and-permute trick.

Full Domain Hash from (Leveled) Multilinear Maps and Identity-Based Aggregate Signatures

Susan Hohenberger ^{*}
Johns Hopkins University

Amit Sahai [†]
UCLA

Brent Waters [‡]
University of Texas at Austin

July 9, 2013

Abstract

In this work, we explore building constructions with full domain hash structure, but with standard model proofs that do not employ the random oracle heuristic. The launching point for our results will be the utilization of a “leveled” *multilinear map* setting for which Garg, Gentry, and Halevi (GGH) recently gave an approximate candidate. Our first step is the creation of a standard model signature scheme that exhibits the structure of the Boneh, Lynn and Shacham signatures. In particular, this gives us a signature that admits unrestricted aggregation.

We build on this result to offer the first *identity-based* aggregate signature scheme that admits unrestricted aggregation. In our construction, an arbitrary-sized set of signatures on identity/message pairs can be aggregated into a single group element, which authenticates the entire set. The identity-based setting has important advantages over regular aggregate signatures in that it eliminates the considerable burden of having to store, retrieve or verify a set of verification keys, and minimizes the total cryptographic overhead that must be attached to a set of signer/message pairs. While identity-based signatures are trivial to achieve, their aggregate counterparts are not. To the best of our knowledge, no prior candidate for realizing unrestricted identity-based aggregate signatures exists in either the standard or random oracle models.

A key technical idea underlying these results is the realization of a hash function with a Naor-Reingold-type structure that is publicly computable using repeated application of the multilinear map. We present our results in a generic “leveled” multilinear map setting and then show how they can be translated to the GGH graded algebras analogue of multilinear maps.

^{*}Supported by the National Science Foundation (NSF) CNS-1154035, CNS-1228443; the Defense Advanced Research Projects Agency (DARPA) and the Air Force Research Laboratory (AFRL) under contract FA8750-11-2-0211, DARPA N11AP20006, the Office of Naval Research under contract N00014-11-1-0470, and a Microsoft Faculty Fellowship.

[†]Research supported in part from a DARPA/ONR PROCEED award, NSF grants 1228984, 1136174, 1118096, 1065276, 0916574 and 0830803, a Xerox Faculty Research Award, a Google Faculty Research Award, an equipment grant from Intel, and an Okawa Foundation Research Grant. This material is based upon work supported by the Defense Advanced Research Projects Agency through the U.S. Office of Naval Research under Contract N00014-11-1-0389. The views expressed are those of the author and do not reflect the official policy or position of the Department of Defense, the National Science Foundation, or the U.S. Government.

[‡]Supported by NSF CNS-0915361, CNS-0952692, CNS-1228599; DARPA through the U.S. Office of Naval Research under Contract N00014-11-1-0382, DARPA N11AP20006, a Google Faculty Research Award, an Alfred P. Sloan Fellowship, a Microsoft Faculty Fellowship, and a Packard Foundation Fellowship. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the Department of Defense or the U.S. Government.

1 Introduction

Applying a full domain hash is a common technique in cryptography where a hash function, modeled as a random oracle, is used to hash a string into a set. Originally, the concept referred to a signature scheme where one hashed into the range of a trapdoor permutation [4]. Subsequently, full domain hash has been treated as a more general concept and applied in bilinear map cryptography where typically a hash function $H : \{0, 1\}^* \rightarrow \mathbb{G}$ is used to hash a string into a bilinear group. (We note that multiple early works [11, 13, 12] employ this terminology.) Pairing-based applications of Full Domain Hash include: the original Boneh-Franklin [11], short and aggregate signatures [13, 12], Hierarchical Identity-Based Encryption [25], and decentralized Attribute-Based Encryption [30]. Typically, proofs of such schemes will use the random oracle heuristic to “program” the output of the hash function in a certain way for which there is no known standard model equivalent (see [28]).

Given that there are well-known issues with random oracle instantiability in general [16] and problems with Full Domain Hash in particular [20, 19], there has been a push to find standard model realizations of these applications. These endeavors have been successful in several applications such as signatures [9, 40] and (Hierarchical) Identity-Based Encryption [17, 7, 8, 40, 23, 41]. Despite this progress, the current state is not entirely satisfactory on two fronts. First, each of the standard model examples given above created new cryptographic constructions with fundamentally different structure than the original Full Domain Hash construction. While creating a new structure is a completely valid and novel approach, that path does not necessarily lend insight or further understanding of the original constructions.

Second, there are important applications of the Full Domain Hash method where implementing such a hash using a random oracle introduces significant limitations in the applicability of the Full Domain Hash method. A prominent example concerns aggregate signature schemes and their identity-based counterparts:

An aggregate signature system is one in which a signature σ' on verification key/message pair (VK', M') can be combined with a signature $\tilde{\sigma}$ on $(\tilde{VK}, \tilde{M})$ producing a new signature σ on the set $S = \{(VK', M'), (\tilde{VK}, \tilde{M})\}$. This process can be repeated indefinitely to aggregate an arbitrary number of signatures together. Crucially, the size of σ should be independent of the number of signatures aggregated, although the description of the set S will grow. The ultimate goal, however, is to minimize the entire transmission size [35].

The need for a public-key infrastructure for verification keys is a major drawback of traditional public-key cryptography, and for this reason identity-based cryptography has flourished [39, 11]: In an *identity-based* aggregate signature scheme, verification keys like VK would be replaced with simple identity strings like $\mathcal{I} = \text{“harrypotter@hogwarts.edu”}$. This offers a very meaningful savings for protocols such as BGPsec, which require routers to store, retrieve and verify certificates for over 36,000 public keys [18, 15]. We note that while identity-based signatures follow trivially from standard signatures, identity-based aggregate signatures are nontrivial (more on this below).

A decade ago, the Boneh, Gentry, Lynn and Shacham (BGLS) [12] aggregate signature scheme was built using the Full Domain Hash methodology. In the original vision of BGLS, aggregation could be performed by any third party on any number of signatures. The authors showed how the Boneh, Lynn and Shacham (BLS) [13] signatures (which are in turn comprised of Boneh-Franklin [11] private IBE keys) can be aggregated in this manner. The BLS construction uses a full domain hash and its security proof is in the random oracle model. However, even though the BGLS scheme was built upon the key mechanism for Boneh-Franklin Identity-Based Encryption, BGLS does *not* support identity-based aggregation. The Full Domain Hash in BGLS is realized using

a random oracle, which destroys the structure that would be needed for identity-based aggregate signatures. To the best of our knowledge, no prior solution to identity-based aggregate signatures in either the standard or random oracle models exists. Prior work considered ID-based aggregates restricted to a common nonce [24] (e.g., where signatures can only be aggregated if they were created with the same nonce or time period) or sequential additions [6] (e.g., where a group of signers sequentially form an aggregate by each adding their own signature to the aggregate-so-far).

Our results in a nutshell. In this work, we give a new method for implementing the Full Domain Hash method using leveled multilinear maps, including the ones recently proposed by Garg, Gentry, and Halevi (GGH) [21]. We show how to use this method to implement aggregate signatures in the standard model in a way that naturally extends to give the first full solution to the problem of identity-based aggregate signatures (also in the standard model).

Prior work on standard model aggregate signatures. All previous work on achieving standard model aggregate signatures did so by departing fundamentally from the Full Domain Hash methodology.

Subsequently to BGLS [12], different standard model solutions were proposed, but with different restrictions on aggregation. These include: constructions [31] where the signatures must be sequentially added in by the signers, multisignatures [31] where aggregation can occur only for the same message M , or where aggregation is limited to signatures associated with the same nonce or time period [1].¹ These restrictions limit their practical applicability.

In 2009, Rückert and Schröder [38] gave an intriguing vision on how multilinear maps might enable standard model constructions of aggregate signatures, also departing from the Full Domain Hash methodology. They did not discuss or achieve identity-based aggregate signatures. Their proposal came before the Garg, Gentry and Halevi [21] candidate and used the earlier Boneh-Silverberg [14] view of multilinear maps, where a k -linear map would allow the *simultaneous* multiplication of k source group elements into one target group element. The GGH candidate in contrast allows for encodings to exist on multiple levels and a pairing between an encoding on level i and one on level j gives an encoding on level $i + j$ as long as $i + j$ is less than or equal to some k . One drawback of the Rückert and Schröder construction is that the security proof requires access to an interactive (or oracle-type) assumption in order to answer the signature queries where the structure of the oracle output is essentially identical to the signatures required. This property seems to be tightly coupled with the modeling of a multilinear map as a one time multiplication. In contrast, we will exploit the leveling of the GGH abstraction to actually replace the hash function in a BLS-type structure and obtain proofs from non-interactive assumptions.

1.1 Overview of our Aggregate Signature Constructions

We now overview the constructions and their security claims. To simplify the description of the main ideas, we describe the constructions here in terms of leveled multilinear maps. Later on, we explicitly give translations of both constructions to the GGH framework.

¹We remark that these restrictions were considered in other works such as [37, 36, 33, 5, 32] prior to the standard model constructions cited above.

The Base Construction. A trusted setup algorithm will take as input security parameter λ and message bit-length ℓ and run a group generator $\mathcal{G}(1^\lambda, k = \ell + 1)$ and outputs a sequence of groups $\vec{\mathbb{G}} = (\mathbb{G}_1, \dots, \mathbb{G}_k)$ of prime order p .² The group sequence will have canonical generators $g = g_1, g_2, \dots, g_k$ along with a pairing operation that computes $e(g_i^a, g_j^b) = g_{i+j}^{ab}$ for any $a, b \in \mathbb{Z}_p$ and $i + j \leq k$. The setup algorithm will also choose $\vec{A} = (A_{1,0} = g^{a_{1,0}}, A_{1,1} = g^{a_{1,1}}), \dots, (A_{\ell,0} = g^{a_{\ell,0}}, A_{\ell,1} = g^{a_{\ell,1}}) \in \mathbb{G}_1^2$. We define $H : \{0, 1\}^\ell \rightarrow \mathbb{G}_{k-1}$ as $H(M) = g_{k-1}^{\prod_{i \in [1, \ell]} a_i m_i}$, where m_i are the bits of message M . The hash function hashes a message into the group $\mathbb{G}_{k-1}$. It exhibits a Naor-Reingold [34]-type structure and is publicly computable using repeated application of a multilinear map. Since a group element in $\mathbb{G}_{k-1}$ has one pairing left, it intuitively reflects the bilinear map setting. In our scheme a private key contains a random exponent $\alpha \in \mathbb{Z}_p$ and the corresponding verification key VK contains g^α . A signature on a message M is computed as $\sigma = H(M)^\alpha$ and verified by testing $e(\sigma, g) \stackrel{?}{=} e(H(M), g^\alpha)$.

Stepping back, the structure of our scheme very closely resembles BLS signatures. For this reason it is possible to aggregate them in the BGLS fashion by simply multiplying two together. The size of an aggregate signature depends on the security parameter plus message length ℓ (assuming the group representation size increases with $k = \ell + 1$), but is independent of the number of times aggregation is applied. Aggregation is unrestricted and can be done by any third party.

The Rückert and Schröder construction [38] also insightfully uses a Naor-Reingold type function for aggregation. A key distinction is that in the RS method there is a *unique* NR function for each signer and it is privately computed by each signer per each message/input. In our construction the Naor-Reingold function is computed as a *public* hash using the levels of the multilinear map. A signer simply multiplies in his secret exponent after computing the hash. Thus, this mimicks the BLS structure much more closely. One advantage of our structure is that the hash function can be derived from a single common reference string and then public keys are just a single group element. In addition, we will see that our structure is amenable to proofs under non-interactive assumptions and allow us to extend to the identity-based setting. In the aggregation setting, where bandwidth is at a premium, our smaller public keys and the ability to go identity-based is important.

Proofs of Security. We view our aggregate signatures as signatures on a multiset of message/verification key pairs for full generality. We prove security in a modular way as a two step process. First, we define a weaker “distinct message” variant of security that only considers an attacker successful if the aggregate forgery no two signers sign the same message. We then show how to transform any distinct message secure scheme into one with standard security. The transformation captures the BGLS idea (formalized by Bellare, Namprempre and Neven [2]) of hashing the public key plus message together. Using the transformation we can focus on designing proofs in the distinct message game. We first prove selective security under a natural analog of the CDH assumption we call the k -Multilinear Computational Diffie-Hellman (k -MCDH) assumption. We next show full (a.k.a., adaptive) security using a subexponentially secure version of the assumption. Finally, we show full security with only polynomial factors in the reduction using a non-interactive, but parameterized assumption.

Realizing Identity-Based Aggregation. The authority will run a setup algorithm that takes the message bit-length ℓ and identity bit-length n . It runs a group generator $\mathcal{G}(1^\lambda, k = \ell + n)$

²In practice one will perform a CRHF of an arbitrary length message to ℓ bits.

and outputs a sequence of groups $\vec{\mathbb{G}} = (\mathbb{G}_1, \dots, \mathbb{G}_k)$ of prime order p . It creates the parameters $\vec{A}$ as in the prior scheme and $\vec{B} = (B_{1,0} = g^{b_{1,0}}, B_{1,1} = g^{b_{1,1}}), \dots, (B_{n,0} = g^{b_{n,0}}, B_{n,1} = g^{b_{n,1}}) \in \mathbb{G}_1^2$. We define $H : \{0,1\}^n \times \{0,1\}^\ell \rightarrow \mathbb{G}_{k-1}$ as $H(\mathcal{I}, M) = g_k^{(\prod_{i \in [1,n]} b_{i, \text{id}_i}) (\prod_{i \in [1,\ell]} a_{i, m_i})}$, where m_i are the bits of message M and id_i the bits of $\mathcal{I}$. The hash function is publicly computable from the multilinear map. A secret key for identity $\mathcal{I}$ is computed as $\text{SK}_{\mathcal{I}} = g_{n-1}^{\prod_{i \in [1,n]} b_{i, \text{id}_i}} \in G_{n-1}$. This can be used to produce a signature on message M by computing $(g_{k-1})^{(\prod_{i \in [1,n]} b_{i, \text{id}_i}) (\prod_{i \in [1,\ell]} a_{i, m_i})}$ using the multilinear map. Finally, a signature can be verified by checking $e(\sigma, g) \stackrel{?}{=} H(\mathcal{I}, M)$. The signatures will aggregate in the same manner by multiplying together.

The distinct message translation is not required in the identity-based setting, because there is no rogue key problem. We first prove selective security under the k -MCDH assumption, and then show full security using a subexponentially secure version of the assumption. We provide these proofs in both the generic multilinear and the GGH framework.

Further Applications. Taken altogether we show that multilinear forms provide an opportunity for revisiting cryptographic structures that were strongly associated with the random oracle heuristic. It remains to be seen how widely this direction will apply. One interesting example of an application that currently requires the full domain hash is the decentralized Attribute-Based Encryption system of Lewko and Waters [30]. There is no standard model candidate that has comparable expressiveness. Here performing an analogous transformation to our aggregate signatures hash function gives a candidate construction that we do not immediately see how to break. However, it is less easy to see how our proof techniques would extend to the variant of the Lewko-Waters [30] decentralized ABE scheme.

2 Leveled Multilinear Maps and the GGH Graded Encoding

We give a description of generic, leveled multilinear maps. The assumptions used in this setting are defined inline with their respective security proofs. More details of the GGH graded algebras analogue of multilinear maps are included in Appendix A, and for further details, please refer to [21].

For generic, leveled multilinear maps, we assume the existence of a group generator $\mathcal{G}$, which takes as input a security parameter 1^λ and a positive integer k to indicate the number of allowed pairing operations. $\mathcal{G}(1^\lambda, k)$ outputs a sequence of groups $\vec{G} = (\mathbb{G}_1, \dots, \mathbb{G}_k)$ each of large prime order $p > 2^\lambda$. In addition, we let g_i be a canonical generator of $\mathbb{G}_i$ (and is known from the group's description). We let $g = g_1$.

We assume the existence of a set of bilinear maps $\{e_{i,j} : G_i \times G_j \rightarrow G_{i+j} \mid i, j \geq 1; i + j \leq k\}$. The map $e_{i,j}$ satisfies the following relation:

$$e_{i,j}(g_i^a, g_j^b) = g_{i+j}^{ab} : \forall a, b \in \mathbb{Z}_p$$

We observe that one consequence of this is that $e_{i,j}(g_i, g_j) = g_{i+j}$ for each valid i, j .

When the context is obvious, we will sometimes abuse notation and drop the subscripts i, j . For example, we may simply write $e(g_i^a, g_j^b) = g_{i+j}^{ab}$.

Algorithmic components of GGH encodings. While we assume familiarity with the basics of GGH encodings [21], we now review the algorithmic components of the GGH encodings that we will use in our constructions and proofs. The setup algorithm $\text{InstGen}(1^\lambda, 1^k)$ takes as input a security parameter 1^λ and the level of multilinearity 1^k , and outputs the public parameters params needed for using the remaining GGH algorithms, along with a special parameter $\mathbf{p}_{zt}$ to be used for zero testing. The sampling algorithm $\text{samp}(\text{params})$ outputs a level-0 encoding of a randomly chosen element. The canonicalizing encoding $\text{cenc}_e(\text{params}, i, \alpha)$ algorithm takes as input an encoding α of some element a , and outputs a level- i encoding of a , with re-randomization parameter e . This canonicalizing encoding algorithm can re-randomize an encoding for a fixed constant number of re-randomization parameters e . Finally, the zero-testing algorithm $\text{isZero}(\mathbf{p}_{zt}, \alpha)$ takes as input a level- k encoding α , and accepts iff α is an encoding of 0. A more elaborate review of these algorithms can be found in Appendix A.

3 Definitions for Aggregate and ID-based Aggregate Signatures

We now give our definitions for aggregate signatures. In our setting, each aggregate signature is associated with a *multiset* S over verification key/message pairs (or identity/message pairs in the ID-based setting). A set S is of the form $\{(\text{VK}_1, M_1), \dots, (\text{VK}_{|S|}, M_{|S|})\}$. Since S is a multiset it is possible to have $(\text{VK}_i, M_i) = (\text{VK}_j, M_j)$ for $i \neq j$. All signatures, including those that come out of the sign algorithm, are considered to be aggregate signatures. The aggregation algorithm is general in that it can take any two aggregate signatures and combine them into a new aggregate signature.

Our definition allows for an initial trusted setup that will generate a set of common public parameters PP . This will define a bit length of all messages (and identities). In practice one could set these fixed lengths to be the output length ℓ of a collision resistant hash function and allow arbitrary-length messages/identities by first hashing them down to ℓ bits. In the ID-based setting, the authority also produces a master secret key used later to run the key generation algorithm.

We emphasize a few features of our setting. First, aggregation is very general in that it allows for the combination of any two aggregate signatures into a single one. Some prior definitions required an aggregate signature to be combined with a single message signature. This is a limitation for applications where an aggregator comes across two aggregate signatures that it wishes to combine. The aggregation operation does not require any secret keys. The multiset structure allows one to combine two aggregate signatures which both include the same message from the same signer.

We begin formally with the ID-based definition, because it is novel to this work, and then discuss its simpler counterpart.

Authority-Setup $(1^\lambda, \ell, n)$ The trusted setup algorithm takes as input the security parameter as well the bit-length ℓ of messages and bit-length n of the identities. It outputs a common set of public parameters PP and master secret key MSK .

KeyGen $(\text{MSK}, \mathcal{I} \in \{0, 1\}^n)$ The key generation algorithm is run by the authority. It takes as input the system master secret key and an identity $\mathcal{I}$, and outputs a secret signing key $\text{SK}_{\mathcal{I}}$.

Sign $(\text{PP}, \text{SK}_{\mathcal{I}}, \mathcal{I} \in \{0, 1\}^n, M \in \{0, 1\}^\ell)$ The signing algorithm takes as input a secret signing key and corresponding identity $\mathcal{I} \in \{0, 1\}^n$, the common public parameters as well as a message

$M \in \{0, 1\}^\ell$. It outputs a signature σ for identity $\mathcal{I}$. We emphasize that a single signature that is output by this algorithm is considered to also be an aggregate signature.

Aggregate(PP, $\tilde{S}$, S' , $\tilde{\sigma}$, σ'). The aggregation algorithm takes as input two multisets $\tilde{S}$ and S' and purported signatures $\tilde{\sigma}$ and σ' . The elements of $\tilde{S}$ consist of identity/message pairs $\{(\tilde{\mathcal{I}}_1, \tilde{M}_1), \dots, (\tilde{\mathcal{I}}_{|\tilde{S}|}, \tilde{M}_{|\tilde{S}|})\}$ and the elements of S' consist of $\{(\mathcal{I}'_1, M'_1), \dots, (\mathcal{I}'_{|S'|}, M'_{|S'|})\}$. The process produces a signature σ on the multiset $S = \tilde{S} \cup S'$, where $\cup$ is a multiset union.

Verify(PP, S , σ). The verification algorithm takes as input the public parameters, a multiset S of identity and message pairs and an aggregate signature σ . It outputs true or false to indicate whether verification succeeded.

Correctness The correctness property states that all valid aggregate signatures will pass the verification algorithm, where a valid aggregate is defined recursively as an aggregate signature derived by an application of the aggregation algorithm on two valid inputs or the signing algorithm. More formally, for all integers $\lambda, \ell, n, k \geq 1$, all $\text{PP} \in \text{Authority-Setup}(1^\lambda, \ell, n)$, all $\mathcal{I}_1, \dots, \mathcal{I}_k \in \{0, 1\}^n$, all $\text{SK}_{\mathcal{I}_i} \in \text{KeyGen}(\text{PP}, \mathcal{I}_i)$, $\text{Verify}(\text{PP}, S, \sigma) = 1$, if σ is a *valid* aggregate for multiset S under PP. We say that an aggregate signature σ is *valid* for multiset S if: (1) $S = \{(\mathcal{I}_i, M)\}$ for some $i \in [1, k]$, $M \in \{0, 1\}^\ell$ and $\sigma \in \text{Sign}(\text{PP}, \text{SK}_{\mathcal{I}_i}, \mathcal{I}_i, M)$; or (2) there exists multisets $S', \tilde{S}$ where $S = S' \cup \tilde{S}$ and valid aggregate signatures $\sigma', \tilde{\sigma}$ on them respectively such that $\sigma \in \text{Aggregate}(\text{PP}, \tilde{S}, S', \tilde{\sigma}, \sigma')$.

Security Model for Aggregate Signatures. Adapting aggregation [12, 2] to the identity-based setting takes some care in considering how keys are handled and which query requests the adversary should be allowed to make. Informally, in the unforgeability game, it should be computationally infeasible for any adversary to produce a forgery implicating an honest identity, even when the adversary can control all other identities involved in the aggregate and can mount a chosen-message attack on the honest identity. This is defined using a game between a challenger and an adversary $\mathcal{A}$ with respect to scheme $\Pi = (\text{Authority-Setup}, \text{KeyGen}, \text{Sign}, \text{Aggregate}, \text{Verify})$.

– **ID-Unforg**($\Pi, \mathcal{A}, \lambda, \ell, n$):

Setup. The challenger runs $\text{Authority-Setup}(1^\lambda, \ell, n)$ to obtain PP. It sends PP to $\mathcal{A}$.

Queries. Proceeding adaptively, $\mathcal{A}$ can make three types of requests:

1. **Create New Key:** The challenger begins with an index $i = 1$ and an empty sequence of index/identity/private key triples T . On input an identity $\mathcal{I} \in \{0, 1\}^n$, the challenger runs $\text{KeyGen}(\text{MSK}, \mathcal{I})$ to obtain $\text{SK}_{\mathcal{I}}$. It adds the triple $(i, \mathcal{I}, \text{SK}_{\mathcal{I}})$ to T and then increments i for the next call. Nothing is returned to the adversary. We note that the adversary can query this oracle multiple times for the same identity. This will capture security for applications that might release more than one secret key per identity.
2. **Corrupt User:** On input an index $i \in [1, |T|]$, the challenger returns to the adversary the triple $(i, \mathcal{I}_i, \text{SK}_{\mathcal{I}_i}) \in T$. It returns an error if T is empty or i is out of range.
3. **Sign:** On input an index $i \in [1, |T|]$ and a message $M \in \{0, 1\}^\ell$, the challenger obtains the triple $(i, \mathcal{I}_i, \text{SK}_{\mathcal{I}_i}) \in T$ (returning an error if it does not exist) and returns the signature resulting from $\text{Sign}(\text{PP}, \text{SK}_{\mathcal{I}_i}, \mathcal{I}_i, M)$ to $\mathcal{A}$.

Response. Finally, $\mathcal{A}$ outputs a multiset S^* of identity/message pairs and a purported aggregate signature σ^* .

We say the adversary “wins” or that the output of this experiment is 1 if: (1) $\text{Verify}(\text{PP}, S^*, \sigma^*) = 1$ and (2) there exists an element $(\mathcal{I}^*, M^*) \in S^*$ such that M^* was not queried for a signature by the adversary on any index corresponding to $\mathcal{I}^*$; i.e., any index i such that $(i, \mathcal{I}^*, \cdot) \in T$. Otherwise, the output is 0. Define $\text{ID-Forge}_{\mathcal{A}}$ as the probability that $\text{Unforg}(\Pi, \mathcal{A}, \lambda, \ell, n) = 1$, where the probability is over the coin tosses of the Authority-Setup, KeyGen, and Sign algorithms and of $\mathcal{A}$.

Definition 3.1 (Adaptive Unforgeability) *An ID-based aggregate signature scheme Π is existentially unforgeable with respect to adaptive chosen-message attacks if for all probabilistic polynomial-time adversaries $\mathcal{A}$, the function $\text{ID-Forge}_{\mathcal{A}}$ is negligible in λ .*

Selective Security. We consider a selective variant to **ID-Unforg** (selective in both the identity and the message) where there is an Init phase before the Setup phase, wherein $\mathcal{A}$ gives to the challenger a forgery identity/message pair $(\mathcal{I}^* \in \{0, 1\}^n, M^* \in \{0, 1\}^\ell)$. The adversary cannot request a signing key for $\mathcal{I}^*$. (It may request that the challenger create one or more keys for this identity, but it cannot corrupt any user index i associated with $\mathcal{I}^*$.) Moreover, the adversary only “wins” causing the experiment output to be 1 if the normal checks hold (i.e., its signature verifies and it did not request that $\mathcal{I}^*$ sign M^*) and additionally $(\mathcal{I}^*, M^*)$ appears in S^* .

Non-ID-Based Aggregates and the Distinct Message Variant. We provide security definitions for the non-ID-based setting in Appendix B that follow from [12, 2]. We provide adaptive and selective variants. We also identify a weaker “distinct message” security game that is easier to work with. In Appendix C, we describe and prove secure a simple transformation from distinct message security to standard aggregate signature security. The transformation captures the idea of hashing the public key and message together [12, 2] in a modular way. Focusing on distinct message security allows one to avoid the “rogue key” attack (see Section 4.3). We do not consider distinct message security in the ID-based setting, because there are no verification keys.

4 Our Base Aggregate Signature Construction

4.1 Generic Multilinear Construction

Setup($1^\lambda, \ell$) The trusted setup algorithm takes as input the security parameter as well as the length ℓ of messages. It first runs $\mathcal{G}(1^\lambda, k = \ell + 1)$ and outputs a sequence of groups $\vec{\mathbb{G}} = (\mathbb{G}_1, \dots, \mathbb{G}_k)$ of prime order p , with canonical generators $g_1, \dots, g_k$, where we let $g = g_1$.

Next, it outputs random group elements $(A_{1,0}, A_{1,1}), \dots, (A_{\ell,0}, A_{\ell,1}) \in \mathbb{G}_1^2$. These will be used to compute a function $H(M) : \{0, 1\}^\ell \rightarrow \mathbb{G}_{k-1}$, which serves as the analog of the full domain hash function of the BGLS [12] construction. Let $m_1, \dots, m_\ell$ be the bits of message M . It is computed iteratively as $H_1(M) = A_{1,m_1}$ and for $i \in [2, \ell]$, $H_i(M) = e(H_{i-1}(M), A_{i,m_i})$. We define $H(M) = H_\ell(M)$. The public parameters, PP, consist of the group descriptions plus $(A_{1,0}, A_{1,1}), \dots, (A_{\ell,0}, A_{\ell,1})$.

KeyGen(PP) The key generation algorithm first chooses random $\alpha \in \mathbb{Z}_p$. It outputs the public verification key as $\text{VK} = g^\alpha$. The secret key SK is $\alpha \in \mathbb{Z}_p$.

Sign(PP, SK, $M \in \{0, 1\}^\ell$) The signing algorithm computes the signature as $\sigma = H(M)^\alpha \in G_{k-1}$. This serves as an aggregate signature for the (single element) multiset $S = (\text{VK}, M)$.

Aggregate(PP, $\tilde{S}, S', \tilde{\sigma}, \sigma'$). The aggregation algorithm simply computes the output signature σ as $\sigma = \tilde{\sigma} \cdot \sigma'$. This serves as a signature on the multiset $S = \tilde{S} \cup S'$, where $\cup$ is a *multiset union*.

Verify(PP, S, σ). The verification algorithm parses S as $\{(\text{VK}_1, M_1), \dots, (\text{VK}_{|S|}, M_{|S|})\}$. It then checks that $e(\sigma, g) \stackrel{?}{=} \prod_{i=1, \dots, |S|} e(H(M_i), \text{VK}_i)$ and accepts if and only if it holds.

Correctness To see correctness, an aggregate σ on $S = \{(\text{VK}_1, M_1), \dots, (\text{VK}_{|S|}, M_{|S|})\}$ will be the product of individual signatures; i.e., $\sigma = \prod_{i=1}^{|S|} H(M_i)^{\alpha_i}$ where $\text{VK}_i = g^{\alpha_i}$, and thus will pass the verification equation as $e(\sigma, g) = e(\prod_{i=1}^{|S|} H(M_i)^{\alpha_i}, g) = \prod_{i=1}^{|S|} e(H(M_i)^{\alpha_i}, g) = \prod_{i=1}^{|S|} e(H(M_i), g)^{\alpha_i} = \prod_{i=1}^{|S|} e(H(M_i), g^{\alpha_i}) = \prod_{i=1}^{|S|} e(H(M_i), \text{VK}_i)$.

Efficiency and Tradeoffs An aggregate signature is one group element in $\mathbb{G}_{k-1}$ independent of the number of messages aggregated. In a multilinear setting, the space to represent a group element might grow with k (which is $\ell+1$). Indeed, this happens in the GGH [21] graded algebra translation. One way to mitigate this is to differ the message alphabet size in a tradeoff of computation versus storage. The above construction uses a binary message alphabet. If it used an alphabet of 2^d symbols, then the aggregate signature could reside in the group $G_{\ell/d}$ with $\ell/d - 1$ pairings required to compute it, at the cost of the public parameters requiring $2^d \ell$ group elements in $\mathbb{G}$.

4.2 Construction in the GGH Framework

We give a translation of the above construction to the GGH [21] framework in Appendix E.

4.3 Security Analysis

Assumption 4.1 (Multilinear Computational Diffie-Hellman: k -MCDH) *The k -Multilinear Computational Diffie-Hellman (k -MCDH) problem states the following: A challenger runs $\mathcal{G}(1^\lambda, k)$ to generate groups and generators of order p . Then it picks random $c_1, \dots, c_k \in \mathbb{Z}_p$. The assumption then states that given $g = g_1, g^{c_1}, \dots, g^{c_k}$ it is hard for any poly-time algorithm to compute $g_{k-1}^{\prod_{j \in [1, k]} c_j}$ with better than negligible advantage (in security parameter λ).*

We say that the k -MCDH assumption holds against subexponential advantage if there exists a universal constant $\epsilon_0 > 0$ such that no polynomial-time algorithm can succeed in the experiment above with probability greater than $2^{-\lambda^{\epsilon_0}}$. In Section 5.3, we will give a variant of the k -MCDH assumption in the approximate multilinear maps setting of GGH [21] that we will call the GGH k -MCDH assumption. We note that the best cryptanalysis available of the GGH framework [21] suggests that the GGH k -MCDH assumption holds against subexponential advantage.

In Appendix D, we show that the basic aggregate signature scheme for message length ℓ in the distinct message unforgeability game is:

- (Theorem D.1) selectively secure under the $(\ell + 1)$ -Multilinear Computational Diffie-Hellman (MCDH) assumption.

- (Corollary D.2) fully secure under the $(\ell + 1)$ -MCDH assumption against subexponential advantage.
- (Theorem D.4) fully secure under a non-interactive, parameterized assumption which depends on ℓ , the number of adversarial signing queries and the number of messages in the adversary's forgery.

By applying the simple transformation of Appendix C, the distinct message requirement can be removed. Without this transformation, there is a simple attack where the attacker sets some $VK' = VK^{-1}$ and submits the identity element in $\mathbb{G}_{k-1}$ as an aggregate forgery for $S = \{(VK, M), (VK', M)\}$ for any message M of its choosing.

5 Our ID-Based Aggregate Signature Construction

5.1 Generic Multilinear Construction

Authority-Setup $(1^\lambda, \ell, n)$ The trusted setup algorithm is run by the master authority of the ID-based system. It takes as input the security parameter as well the bit-length ℓ of messages and bit-length n of identities. It first runs $\mathcal{G}(1^\lambda, k = \ell + n)$ and outputs a sequence of groups $\vec{\mathbb{G}} = (\mathbb{G}_1, \dots, \mathbb{G}_k)$ of prime order p , with canonical generators $g_1, \dots, g_k$, where we let $g = g_1$.

Next, it chooses random group elements $(A_{1,0} = g^{a_{1,0}}, A_{1,1} = g^{a_{1,1}}), \dots, (A_{\ell,0} = g^{a_{\ell,0}}, A_{\ell,1} = g^{a_{\ell,1}}) \in \mathbb{G}_1^2$. It also chooses random exponents $(b_{1,0}, b_{1,1}), \dots, (b_{n,0}, b_{n,1}) \in \mathbb{Z}_p^2$ and sets $B_{i,\beta} = g^{b_{i,\beta}}$ for $i \in [1, n]$ and $\beta \in \{0, 1\}$.

These will be used to define a function $H(\mathcal{I}, M) : \{0, 1\}^n \times \{0, 1\}^\ell \rightarrow \mathbb{G}_k$. Let $m_1, \dots, m_\ell$ be the bits of message M and $\text{id}_1, \dots, \text{id}_n$ as the bits of $\mathcal{I}$. It is computed iteratively as

$$\begin{aligned} H_1(\mathcal{I}, M) &= B_{1,\text{id}_1} \quad \text{for } i \in [2, n] \quad H_i(\mathcal{I}, M) = e(H_{i-1}(\mathcal{I}, M), B_{i,\text{id}_i}) \\ &\quad \text{for } i \in [n+1, n+\ell = k] \quad H_i(\mathcal{I}, M) = e(H_{i-1}(\mathcal{I}, M), A_{i-n, m_{i-n}}). \end{aligned}$$

We define $H(\mathcal{I}, M) = H_{k=\ell+n}(\mathcal{I}, M)$.

The public parameters, PP, consist of the group sequence description plus:

$$(A_{1,0}, A_{1,1}), \dots, (A_{\ell,0}, A_{\ell,1}), (B_{1,0}, B_{1,1}), \dots, (B_{n,0}, B_{n,1})$$

The master secret key MSK includes PP together with the values $(b_{1,0}, b_{1,1}), \dots, (b_{n,0}, b_{n,1})$.

KeyGen(MSK, $\mathcal{I} \in \{0, 1\}^n$) The signing key for identity $\mathcal{I}$ is $\text{SK}_{\mathcal{I}} = g_{n-1}^{\prod_{i \in [1, n]} b_{i, \text{id}_i}} \in G_{n-1}$.

Sign(PP, $\text{SK}_{\mathcal{I}}$, $\mathcal{I} \in \{0, 1\}^n$, $M \in \{0, 1\}^\ell$) The signing algorithm lets temporary variable $D_0 = \text{SK}_{\mathcal{I}}$. Then for $i = 1$ to ℓ it computes $D_i = e(D_{i-1}, A_{i, m_i}) \in G_{n-1+i}$. The output signature is

$$\sigma = D_\ell = (g_{k-1})^{(\prod_{i \in [1, n]} b_{i, \text{id}_i})(\prod_{i \in [1, \ell]} a_{i, m_i})}.$$

This serves as an ID-based aggregate signature for the (single element) multiset $S = (\mathcal{I}, M)$.

Aggregate(PP, $\tilde{S}, S', \tilde{\sigma}, \sigma'$). The aggregation algorithm simply computes the output signature σ as $\sigma = \tilde{\sigma} \cdot \sigma'$. The serves as a signature on the multiset $S = \tilde{S} \cup S'$, where $\cup$ is a multiset union.

Verify(PP, S , σ). It parses S as $\{(\mathcal{I}_1, M_1), \dots, (\mathcal{I}_{|S|}, M_{|S|})\}$. It then accepts if and only if

$$e(\sigma, g) \stackrel{?}{=} \prod_{i=1, \dots, |S|} H(\mathcal{I}_i, M_i).$$

Correctness and Security To see correctness, an aggregate σ on $S = \{(\mathcal{I}_1, M_1), \dots, (\mathcal{I}_{|S|}, M_{|S|})\}$ will be the product of individual signatures; i.e., σ_i where $e(\sigma_i, g) = H(\mathcal{I}_i, M_i)$, and thus we have $\prod_{i=1}^{|S|} e(\sigma_i, g) = e(\prod_{i=1}^{|S|} \sigma_i, g) = e(\sigma, g) = \prod_{i=1}^{|S|} H(\mathcal{I}_i, M_i)$. We show that this scheme is selectively secure under the k -MCDH assumption in Appendix F, where the proof is very similar to that of the GGH translation which we provide shortly in Section 5.3.

5.2 ID-Based Construction in the GGH Framework

We show how to modify our ID-based construction to use the GGH [21] graded algebras analogue of multilinear maps. Please note that we use the same notation developed in [21], with some minor changes: Firstly, we use the canonical encoding function **cenc** provided by the GGH framework more than once at each level of the encoding, but only a globally fixed constant number of times per level. This is compatible with the GGH encoding [21], and allows for a simpler exposition of our scheme and proof. Also, **for ease of notation on the reader, we suppress repeated params arguments that are provided to every algorithm.** Thus, for instance, we will write $\alpha \leftarrow \text{samp}()$ instead of $\alpha \leftarrow \text{samp}(\text{params})$. Note that in our scheme, there will only ever be a single uniquely chosen value for **params** throughout the scheme, so there is no cause for confusion. Finally, we use the variant of the GGH framework with “strong” zero-testing, where the zero test statistically guarantees that a vector is a valid encoding of zero if it passes the zero test. For further details on the GGH framework, please refer to [21]. See also the summary of [22] as included in Appendix A.

Authority-Setup($1^\lambda, \ell, n$) The trusted setup algorithm is run by the master authority of the ID-based system. It takes as input the security parameter as well the bit-length ℓ of messages and bit-length n of identities. It then runs $(\text{params}, \mathbf{p}_{\text{zt}}) \leftarrow \text{InstGen}(1^\lambda, 1^{k=\ell+n})$. Recall that **params** will be implicitly given as input to all GGH-related algorithms below.

Next, it chooses random encodings $a_{i,\beta} = \text{samp}()$ for $i \in [1, \ell]$ and $\beta \in \{0, 1\}$; and random encodings $b_{i,\beta} = \text{samp}()$ for $i \in [1, n]$ and $\beta \in \{0, 1\}$. Then it assigns $A_{i,\beta} = \text{cenc}_1(1, a_{i,\beta})$ for $i \in [1, \ell]$ and $\beta \in \{0, 1\}$; and it assigns $B_{i,\beta} = \text{cenc}_1(1, b_{i,\beta})$ for $i \in [1, n]$ and $\beta \in \{0, 1\}$.

These will be used to compute a function H mapping $\ell + n$ bit strings to level $k - 1$ encodings. Let $m_1, \dots, m_\ell$ be the bits of M and $\text{id}_1, \dots, \text{id}_n$ be the bits of $\mathcal{I}$. It is computed iteratively as

$$\begin{aligned} H_1(\mathcal{I}, M) &= B_{1, \text{id}_1} \quad \text{for } i \in [2, n] \quad H_i(\mathcal{I}, M) = H_{i-1}(\mathcal{I}, M) \cdot B_{i, \text{id}_i} \\ \text{for } i \in [n+1, n+\ell=k] \quad H_i(\mathcal{I}, M) &= H_{i-1}(\mathcal{I}, M) \cdot A_{i-n, m_{i-n}}. \end{aligned}$$

We define $H(\mathcal{I}, M) = \text{cenc}_2(k, H_{k=\ell+n}(\mathcal{I}, M))$.

The public parameters, PP, consist of the **params**, $\mathbf{p}_{\text{zt}}$ plus:

$$(A_{1,0}, A_{1,1}), \dots, (A_{\ell,0}, A_{\ell,1}), (B_{1,0}, B_{1,1}), \dots, (B_{n,0}, B_{n,1})$$

Note that **params** includes a level 1 encoding of 1, which we denote as g .

The master secret key MSK includes PP together with the encodings $(b_{1,0}, b_{1,1}), \dots, (b_{n,0}, b_{n,1})$.

KeyGen(MSK, $\mathcal{I} \in \{0, 1\}^n$) The signing key for identity $\mathcal{I}$ is $\text{SK}_{\mathcal{I}} = \text{cenc}_2(n - 1, \prod_{i \in [1, n]} b_{i, \text{id}_i})$.

Sign(PP, $\text{SK}_{\mathcal{I}}$, $\mathcal{I} \in \{0, 1\}^n$, $M \in \{0, 1\}^\ell$) The signing algorithm lets temporary variable $D_0 = \text{SK}_{\mathcal{I}}$. Then for $i = 1$ to ℓ it computes $D_i = D_{i-1} \cdot A_{i, m_i}$. The output signature is

$$\sigma = \text{cenc}_3(k - 1, D_\ell).$$

This serves as an ID-based aggregate signature for the (single element) multiset $S = (\mathcal{I}, M)$.

Aggregate(PP, $\tilde{S}, S', \tilde{\sigma}, \sigma'$). The aggregation algorithm simply computes the output signature σ as $\sigma = \tilde{\sigma} + \sigma'$. The serves as a signature on the multiset $S = \tilde{S} \cup S'$, where $\cup$ is a multiset union.

Verify(PP, S, σ). The verification algorithm parses S as $\{(\mathcal{I}_1, M_1), \dots, (\mathcal{I}_{|S|}, M_{|S|})\}$. It rejects if the multiplicity of any identity/message pair is greater than 2^λ .

The algorithm then proceeds to check the signature by setting $\tau = \text{cenc}_2(1, g)$, and testing: :

$$\text{isZero} \left(\mathbf{p}_{zt}, \tau \cdot \sigma - \sum_{i=1, \dots, |S|} H(\mathcal{I}_i, M_i) \right)$$

and accepts if and only if the zero testing procedure outputs true. Recall that g above is a canonical level 1 encoding of 1 that is included in **params**, part of the public parameters.

Correctness. Correctness follows from the same argument as for the ID-based aggregate signature scheme in the generic multilinear setting.

5.3 Proof of Security for ID-based Aggregate Signatures in the GGH framework

We now describe how to modify our proof of security for our ID-based construction to use the GGH [21] graded algebras analogue of multilinear maps. As before, for ease of notation on the reader, we suppress repeated **params** arguments that are provided to every algorithm. For further details on the GGH framework, please refer to Appendix A or [21].

We begin by describing the GGH analogue of the k -MCDH assumption that we will employ:

Assumption 5.1 (GGH analogue of k -MCDH: GGH k -MCDH) *The GGH k -Multilinear Computational Diffie-Hellman (GGH k -MCDH) problem states the following: A challenger runs $\text{InstGen}(1^\lambda, 1^k)$ to obtain $(\text{params}, \mathbf{p}_{zt})$. Note that **params** includes a level 1 encoding of 1, which we denote as g . Then it picks random $c_1, \dots, c_k$ each equal to the result of a fresh call to $\text{samp}()$.*

The assumption then states that given $\text{params}, \mathbf{p}_{zt}, \text{cenc}_1(1, c_1), \dots, \text{cenc}_1(1, c_k)$ it is hard for any poly-time algorithm to compute an integer $t \in [1, 2^\lambda]$ and an encoding z such that

$$zTst \left(\mathbf{p}_{zt}, \text{cenc}_2(1, g) \cdot z - \text{cenc}_1(k, t \cdot \prod_{j \in [1, k]} c_j) \right)$$

outputs true.

We say the GGH k -MCDH assumption holds against subexponential advantage if there exists a universal constant $\epsilon_0 > 0$ such that no polynomial-time algorithm can succeed in the experiment above with probability greater than $2^{\lambda^{\epsilon_0}}$. The best cryptanalysis available of the GGH framework [21] suggests that the GGH k -MCDH assumption holds against subexponential advantage.

We establish full security of our ID-based aggregate signature scheme conditioned on the k -MCDH assumption holding against subexponential advantage. This follows immediately from the following theorem and a standard complexity leveraging argument:

Theorem 5.2 (Selective Security of GGH ID-Based Construction) *The ID-based aggregate signature scheme for message length ℓ and identity length n in Section 5.2 is selectively secure in the unforgeability game in Section 3 under the GGH $(\ell + n)$ -MCDH assumption.*

Corollary 5.3 *The ID-based aggregate signature scheme for message length ℓ in Section 5.2 is fully secure in the distinct message unforgeability game under the GGH $(\ell + n)$ -MCDH assumption against subexponential advantage.*

Proof. This follows immediately from a complexity leveraging argument: the security parameter λ is chosen to ensure that $2^{\lambda^{\epsilon_0}} \gg 2^\ell$, where $2^{-\lambda^{\epsilon_0}}$ is the maximum probability of success allowed in the k -MCDH assumption against subexponential advantage. Now, to establish full security, the simulator performs exactly as in the selective security proof, but first it simply guesses the message that will be forged (instead of expecting the adversary to produce this message). Because this guess will be correct with probability at least $2^{-\ell}$, and the security parameter λ is chosen carefully, full security with polynomial advantage (or even appropriately defined subexponential advantage) implies an attacker on the GGH k -MCDH assumption with subexponential advantage. $\square$

Proof. (of Theorem 5.2) We show that if there exists a PPT adversary $\mathcal{A}$ that can break the selective security of the ID-based aggregate signature scheme in the unforgeability game with probability ϵ for message length ℓ , identity length n and security parameter λ , then there exists a PPT simulator that can break the GGH $(\ell + n)$ -MCDH assumption for security parameter λ with probability ϵ . The simulator takes as input a GGH MCDH instance $\text{params}, \mathbf{p}_{zt}, C_1 = \text{cenc}_1(1, c_1), \dots, C_k = \text{cenc}_1(1, c_k)$ where $k = \ell + n$. Let m_i denote the i th bit of M and id_i denote the i th bit of $\mathcal{I}$. The simulator plays the role of the challenger in the game as follows.

Init. Let $\mathcal{I}^* \in \{0, 1\}^n$ and $M^* \in \{0, 1\}^\ell$ be the forgery identity/message pair output by $\mathcal{A}$.

Setup. The simulator chooses random $x_1, \dots, x_\ell, y_1, \dots, y_n$ with fresh calls to $\text{samp}()$. For $i = 1$ to ℓ , let $A_{i, m_i^*} = C_{i+n}$ and $A_{i, \bar{m}_i^*} = \text{cenc}_1(1, x_i)$. For $i = 1$ to n , let $B_{i, \text{id}_i^*} = C_i$ and $B_{i, \bar{\text{id}}_i^*} = \text{cenc}_1(1, y_i)$. We remark that the parameters are distributed independently and uniformly at random as in the real scheme.

Queries. Conceptually, the simulator will be able to create keys or signatures for the adversary, because his requests will differ from the challenge identity or message in at least one bit. More specifically,

1. Create New Key: The simulator begins with an index $i = 1$ and an empty sequence of index/identity/private key triples T . On input an identity $\mathcal{I} \in \{0, 1\}^n$, if $\mathcal{I} = \mathcal{I}^*$, the simulator records $(i, \mathcal{I}^*, \perp)$ in T . Otherwise, the simulator computes the secret key as follows. Let β be the first index such that $\text{id}_i \neq \text{id}_i^*$. Compute $s = \prod_{i=1, \dots, n \wedge i \neq \beta} B_{i, \text{id}_i}$.

Then compute $\text{SK}_{\mathcal{I}} = \text{cenc}_2(n - 1, s \cdot y_\beta)$. Record $(i, \mathcal{I}, \text{SK}_{\mathcal{I}})$ in T . Secret keys are well-formed and, due to the rerandomization in the cenc_2 algorithm, are distributed in a manner statistically exponentially close to the keys generated in the real game.

2. Corrupt User: On input an index $i \in [1, |T|]$, the simulator returns to the adversary the triple $(i, \mathcal{I}_i, \text{SK}_{\mathcal{I}_i}) \in T$. It returns an error if T is empty or i is out of range. Recall that i cannot be associated with $\mathcal{I}^*$ in this game.
3. Sign: On input an index $i \in [1, |T|]$ and a message $M \in \{0, 1\}^\ell$, the simulator obtains the triple $(i, \mathcal{I}_i, \text{SK}_{\mathcal{I}_i}) \in T$ or returns an error if it does not exist. If $\mathcal{I}_i \neq \mathcal{I}^*$, then the simulator signs M with $\text{SK}_{\mathcal{I}_i}$ in the usual way.

If $\mathcal{I}_i = \mathcal{I}^*$, then we know $M \neq M^*$. Let β be the first index such that $m_\beta \neq m_\beta^*$. First compute $\sigma' = \prod_{i=1, \dots, \ell \wedge i \neq \beta} A_{i, m_i}$. Next, compute $\sigma'' = \sigma' \cdot x_i$. Also compute $\gamma = \prod_{i=1, \dots, n} B_{i, \text{id}_i}$. Finally, compute $\sigma = \text{cenc}_3(k - 1, \gamma \cdot \sigma'')$. Return σ to $\mathcal{A}$. Signatures are well-formed and, due to the rerandomization in the cenc_3 algorithm, are distributed in a manner statistically exponentially close to the keys generated in the real game.

Response. Eventually, $\mathcal{A}$ outputs an aggregate signature σ^* on multiset S^* where $(\mathcal{I}^*, M^*) \in S^*$. The simulator will extract from this a solution to the MCDH problem. This works by iteratively computing all the other signatures in S^* and then subtracting them out of the aggregate until only one or more signatures on $(\mathcal{I}^*, M^*)$ remain. That is, the simulator takes an aggregate for S^* and computes an aggregate signature for S' where S' has one less verification key/message pair than S at each step. These signatures will be computed as in the query phase.

Eventually, we have an aggregate σ' on $t \geq 1$ instances of $(\mathcal{I}^*, M^*)$. However recall that $H(\mathcal{I}^*, M^*)$ is a level k encoding of $(\prod_{i \in [1, n]} b_{i, \text{id}_i^*})(\prod_{i \in [1, \ell]} a_{i, m_i^*}) = \prod_{i \in [k]} c_i$. Thus verification of the signature σ' implies that (t, σ') is a solution to the GGH k -MCDH problem, and so the simulator returns (t, σ') to break the GGH k -MCDH assumption.

As remarked above, the responses of the challenger are distributed statistically exponentially closely to the real unforgeability game. The simulator succeeds whenever $\mathcal{A}$ does. $\square$

References

- [1] Jae Hyun Ahn, Matthew Green, and Susan Hohenberger. Synchronized aggregate signatures: new definitions, constructions and applications. In *ACM Conference on Computer and Communications Security*, pages 473–484, 2010.
- [2] Mihir Bellare, Chanathip Namprempre, and Gregory Neven. Unrestricted aggregate signatures. In *ICALP*, pages 411–422, 2007.
- [3] Mihir Bellare and Thomas Ristenpart. Simulation without the artificial abort: Simplified proof and improved concrete security for waters’ ibe scheme. In *EUROCRYPT*, pages 407–424, 2009.
- [4] Mihir Bellare and Phillip Rogaway. Random oracles are practical: A paradigm for designing efficient protocols. In *ACM Conference on Computer and Communications Security*, pages 62–73, 1993.

- [5] Alexandra Boldyreva. Threshold signatures, multisignatures and blind signatures based on the gap-diffie-hellman-group signature scheme. In *Public Key Cryptography*, pages 31–46, 2003.
- [6] Alexandra Boldyreva, Craig Gentry, Adam O’Neill, and Dae Hyun Yum. Ordered multisignatures and identity-based sequential aggregate signatures, with applications to secure routing. In *ACM Conference on Computer and Communications Security*, pages 276–285, 2007.
- [7] Dan Boneh and Xavier Boyen. Efficient selective-id secure identity-based encryption without random oracles. In *EUROCRYPT*, pages 223–238, 2004.
- [8] Dan Boneh and Xavier Boyen. Secure identity based encryption without random oracles. In *CRYPTO*, pages 443–459, 2004.
- [9] Dan Boneh and Xavier Boyen. Short signatures without random oracles. In *EUROCRYPT*, pages 56–73, 2004.
- [10] Dan Boneh, Xavier Boyen, and Eu-Jin Goh. Hierarchical identity based encryption with constant size ciphertext. In *EUROCRYPT*, pages 440–456, 2005.
- [11] Dan Boneh and Matthew K. Franklin. Identity-based encryption from the weil pairing. *SIAM J. Comput.*, 32(3):586–615, 2003. extended abstract in Crypto 2001.
- [12] Dan Boneh, Craig Gentry, Ben Lynn, and Hovav Shacham. Aggregate and verifiably encrypted signatures from bilinear maps. In *EUROCRYPT*, pages 416–432, 2003.
- [13] Dan Boneh, Ben Lynn, and Hovav Shacham. Short signatures from the weil pairing. In *ASIACRYPT*, pages 514–532, 2001.
- [14] Dan Boneh and Alice Silverberg. Applications of multilinear forms to cryptography. *IACR Cryptology ePrint Archive*, 2002:80, 2002.
- [15] Kyle Brogle, Sharon Goldberg, and Leonid Reyzin. Sequential aggregate signatures with lazy verification from trapdoor permutations - (extended abstract). In *ASIACRYPT*, pages 644–662, 2012.
- [16] Ran Canetti, Oded Goldreich, and Shai Halevi. The random oracle methodology, revisited. *J. ACM*, 51(4):557–594, 2004.
- [17] Ran Canetti, Shai Halevi, and Jonathan Katz. A forward-secure public-key encryption scheme. In *EUROCRYPT*, pages 255–271, 2003.
- [18] Ying-Ju Chi, Ricardo Oliveira, and Lixia Zhang. Cyclops: The Internet AS-level Observatory. In *ACM SIGCOMM CCR*, 2008.
- [19] Yevgeniy Dodis, Iftach Haitner, and Aris Tentes. On the instantiability of hash-and-sign rsa signatures. In *TCC*, pages 112–132, 2012.
- [20] Yevgeniy Dodis, Roberto Oliveira, and Krzysztof Pietrzak. On the generic insecurity of the full domain hash. In *CRYPTO*, pages 449–466, 2005.
- [21] Sanjam Garg, Craig Gentry, and Shai Halevi. Candidate multilinear maps from ideal lattices and applications. In *EUROCRYPT*, 2013.

- [22] Sanjam Garg, Craig Gentry, Shai Halevi, Amit Sahai, and Brent Waters. Attribute-based encryption for circuits from multilinear maps. In *CRYPTO*, 2013.
- [23] Craig Gentry. Practical identity-based encryption without random oracles. In *EUROCRYPT*, pages 445–464, 2006.
- [24] Craig Gentry and Zulfikar Ramzan. Identity-based aggregate signatures. In *Public Key Cryptography*, pages 257–273, 2006.
- [25] Craig Gentry and Alice Silverberg. Hierarchical id-based cryptography. In *ASIACRYPT*, pages 548–566, 2002.
- [26] Jeffrey Hoffstein, Jill Pipher, and Joseph H. Silverman. Ntru: A ring-based public key cryptosystem. In *ANTS*, pages 267–288, 1998.
- [27] Dennis Hofheinz, Tibor Jager, and Edward Knapp. Waters signatures with optimal security reduction. In *Public Key Cryptography*, pages 66–83, 2012.
- [28] Dennis Hofheinz and Eike Kiltz. Programmable hash functions and their applications. *J. Cryptology*, 25(3):484–527, 2012.
- [29] Susan Hohenberger and Brent Waters. Constructing verifiable random functions with large input spaces. In *EUROCRYPT*, pages 656–672, 2010.
- [30] Allison B. Lewko and Brent Waters. Decentralizing attribute-based encryption. In *EUROCRYPT*, pages 568–588, 2011.
- [31] Steve Lu, Rafail Ostrovsky, Amit Sahai, Hovav Shacham, and Brent Waters. Sequential aggregate signatures and multisignatures without random oracles. In *EUROCRYPT*, pages 465–485, 2006.
- [32] Anna Lysyanskaya, Silvio Micali, Leonid Reyzin, and Hovav Shacham. Sequential aggregate signatures from trapdoor permutations. In *EUROCRYPT*, pages 74–90, 2004.
- [33] Silvio Micali, Kazuo Ohta, and Leonid Reyzin. Accountable-subgroup multisignatures: extended abstract. In *ACM Conference on Computer and Communications Security*, pages 245–254, 2001.
- [34] Moni Naor and Omer Reingold. Constructing pseudo-random permutations with a prescribed structure. *J. Cryptology*, 15(2):97–102, 2002.
- [35] Gregory Neven. Efficient sequential aggregate signed data. *IEEE Transactions on Information Theory*, 57(3):1803–1815, 2011.
- [36] Kazuo Ohta and Tatsuaki Okamoto. A digital multisignature scheme based on the fiat-shamir scheme. In *ASIACRYPT*, pages 139–148, 1991.
- [37] Tatsuaki Okamoto. A digital multisignature schema using bijective public-key cryptosystems. *ACM Trans. Comput. Syst.*, 6(4):432–441, 1988.
- [38] Markus Rückert and Dominique Schröder. Aggregate and verifiably encrypted signatures from multilinear maps without random oracles. In *ISA*, pages 750–759, 2009.

- [39] Adi Shamir. Identity-based cryptosystems and signature schemes. In *CRYPTO*, pages 47–53, 1984.
- [40] Brent Waters. Efficient identity-based encryption without random oracles. In *EUROCRYPT*, pages 114–127, 2005.
- [41] Brent Waters. Dual system encryption: Realizing fully secure ibe and hibe under simple assumptions. In *CRYPTO*, pages 619–636, 2009.

A Background on GGH

In this section, we provide some background on the GGH framework. We use the GGH framework in a manner very similar to the way it was used in the recent work of Garg, Gentry, Halevi, Sahai, and Waters on constructing Attribute-Based Encryption for Circuits [22]. For consistency, the following text is taken verbatim from [22]:

A.1 Graded Encoding Systems: Definition

Garg, Gentry and Halevi (GGH) [21] defined an “approximate” version of a multilinear group family, which they call a *graded encoding system*. As a starting point, they view g_i^α in a multilinear group family as simply an *encoding* of α at “level- i ”. This encoding permits basic functionalities, such as equality testing (it is easy to check that two level- i encodings encode the same exponent), additive homomorphism (via the group operation in $\mathbb{G}_i$), and bounded multiplicative homomorphism (via the multilinear map e). They retain the notion of a somewhat homomorphic encoding with equality testing, but they use probabilistic encodings, and replace the multilinear group family with “less structured” sets of encodings related to lattices.

Abstractly, their n -graded encoding system for a ring R includes a system of sets $\mathcal{S} = \{S_i^{(\alpha)} \subset \{0, 1\}^* : i \in [0, n], \alpha \in R\}$ such that, for every fixed $i \in [0, n]$, the sets $\{S_i^{(\alpha)} : \alpha \in R\}$ are disjoint (and thus form a partition of $S_i := \bigcup_{\alpha} S_i^{(\alpha)}$). The set $S_i^{(\alpha)}$ consists of the “level- i encodings of α ”. Moreover, the system comes equipped with efficient procedures, as follows:³

Instance Generation. The randomized $\text{InstGen}(1^\lambda, 1^n)$ takes as input the security parameter λ and integer n . The procedure outputs $(\text{params}, \mathbf{p}_{zt})$, where params is a description of an n -graded encoding system as above, and $\mathbf{p}_{zt}$ is a level- n “zero-test parameter”.

Ring Sampler. The randomized $\text{samp}(\text{params})$ outputs a “level-zero encoding” $a \in S_0$, such that the induced distribution on α such that $a \in S_0^{(\alpha)}$ is statistically uniform.

Encoding. The (possibly randomized) $\text{enc}(\text{params}, i, a)$ takes $i \in [n]$ and a level-zero encoding $a \in S_0^{(\alpha)}$ for some $\alpha \in R$, and outputs a level- i encoding $u \in S_i^{(\alpha)}$ for the same α .

Re-Randomization. The randomized $\text{reRand}(\text{params}, i, u)$ re-randomizes encodings to the same level, as long as the initial encoding is under a given noise bound. Specifically, for a level $i \in [n]$ and encoding $u \in S_i^{(\alpha)}$, it outputs another encoding $u' \in S_i^{(\alpha)}$. Moreover for any two

³Since GGH’s realization of a graded encoding system uses “noisy” encodings over ideal lattices, the procedures incorporate information about the magnitude of the noise.

encodings $u_1, u_2 \in S_i^{(\alpha)}$ whose noise bound is at most some b , the output distributions of $\text{reRand}(\text{params}, i, u_1)$ and $\text{reRand}(\text{params}, i, u_2)$ are statistically the same.

Addition and negation. Given params and two encodings at the same level, $u_1 \in S_i^{(\alpha_1)}$ and $u_2 \in S_i^{(\alpha_2)}$, we have $\text{add}(\text{params}, u_1, u_2) \in S_i^{(\alpha_1 + \alpha_2)}$, and $\text{neg}(\text{params}, u_1) \in S_i^{(-\alpha_1)}$, subject to bounds on the noise.

Multiplication. For $u_1 \in S_{i_1}^{(\alpha_1)}$, $u_2 \in S_{i_2}^{(\alpha_2)}$, we have $\text{mult}(\text{params}, u_1, u_2) \in S_{i_1 + i_2}^{(\alpha_1 \cdot \alpha_2)}$.

Zero-test. The procedure $\text{isZero}(\text{params}, \mathbf{p}_{zt}, u)$ outputs 1 if $u \in S_n^{(0)}$ and 0 otherwise. Note that in conjunction with the procedure for subtracting encodings, this gives us an equality test.

Extraction. This procedure extracts a “canonical” and “random” representation of ring elements from their level- n encoding. Namely $\text{ext}(\text{params}, \mathbf{p}_{zt}, u)$ outputs (say) $K \in \{0, 1\}^\lambda$, such that:

- (a) With overwhelming probability over the choice of $\alpha \in R$, for any two $u_1, u_2 \in S_n^{(\alpha)}$, $\text{ext}(\text{params}, \mathbf{p}_{zt}, u_1) = \text{ext}(\text{params}, \mathbf{p}_{zt}, u_2)$,
- (b) The distribution $\{\text{ext}(\text{params}, \mathbf{p}_{zt}, u) : \alpha \in R, u \in S_n^{(\alpha)}\}$ is statistically uniform over $\{0, 1\}^\lambda$.

We can extend add and mult to handle more than two encodings as inputs, by applying the binary versions of add and mult iteratively. Also, it is convenient to define a canonicalized encoding algorithm $\text{cenc}(\text{params}, i, a)$ which takes as input encoding of a and generates another encoding according to a “nice” distribution. The parameter ℓ denotes that the noise introduced with ℓ successive invocations of the reRand operation. This parameter was implicit in [21] encoding scheme and we make it explicit. This parameter essentially captures the noise present in the encodings. In our scheme we will need to re-randomize at most a constant number of times and hence the maximum value ℓ takes will be a small constant.

A.2 Graded Encoding Systems: Realization

Concretely, GGH’s n -graded encoding system works as follows. (This is a whirlwind overview; see [21] for details.) The system uses three rings. First, it uses the ring of integers $\mathcal{O}$ of the m -th cyclotomic field. This ring is typically represented as the ring of polynomials $\mathcal{O} = \mathbb{Z}[x]/(\Phi_m(x))$, where $\Phi_m(x)$ is the m -th cyclotomic polynomial, which has degree $N = \phi(m)$. Second, for some suitable integer modulus q , it uses the quotient ring $\mathcal{O}/(q) = \mathbb{Z}_q[x]/(\Phi_m(x))$, similar to the NTRU encryption scheme [26]. The encodings live in $\mathcal{O}/(q)$. Finally, it uses the quotient ring $R = \mathcal{O}/\mathcal{I}$, where $\mathcal{I} = \langle g \rangle$ is a principal ideal of $\mathcal{O}$ that is generated by g and where $|\mathcal{O}/\mathcal{I}|$ is a large prime. This is the ring “ R ” referred to above; elements of R are what is encoded.

What does a GGH encoding look like? For a fixed random $z \in \mathcal{O}/(q)$, an element of $S_i^{(\alpha)}$ – that is, a level- i encoding of $\alpha \in R$ – has the form $e/z^i \in \mathcal{O}/(q)$, where $e \in \mathcal{O}$ is a “small” representative of the coset $\alpha + \mathcal{I}$ (it has coefficients that are very small compared to q). To add encodings $e_1/z^i \in S_i^{(\alpha_1)}$ and $e_2/z^i \in S_i^{(\alpha_2)}$, just add them in $\mathcal{O}/(q)$ to obtain $(e_1 + e_2)/z^i$, which is in $S_i^{(\alpha_1 + \alpha_2)}$ if $e_1 + e_2$ is “small”. To mult encodings $e_1/z^{i_1} \in S_{i_1}^{(\alpha_1)}$ and $e_2/z^{i_2} \in S_{i_2}^{(\alpha_2)}$, just multiply them in $\mathcal{O}/(q)$ to obtain $e_1 \cdot e_2/z^{i_1 + i_2}$, which is in $S_{i_1 + i_2}^{(\alpha_1 \cdot \alpha_2)}$ if $e_1 \cdot e_2$ is “small”. This smallness condition limits the GGH encoding system to degree polynomial in the security parameter. Intuitively, dividing encodings does not “work”, since the resulting denominator has a nontrivial term that is not z .

The GGH **params** allow everyone to generate encodings of random (known) values. The **params** include a level-1 encoding of 1 (from which one can generate encodings of 1 at other levels), and (for each $i \in [n]$) a sufficient number of level- i encodings of 0 to enable re-randomization. To encode (say at level-1), run **samp(params)** to sample a small element a from $\mathcal{O}$, e.g. according to a discrete Gaussian distribution. For a Gaussian with appropriate deviation, this will induce a statistically uniform distribution over the cosets of $\mathcal{I}$. Then, multiply a with the level-1 encoding of 1 to get a level-1 encoding u of $a \in R$. Finally, run **reRand(params, 1, u)**, which involves adding a random Gaussian linear combination of the level-1 encodings of 0, whose noisiness (i.e., numerator size) “drowns out” the initial encoding. The parameters for the GGH scheme can be instantiated such that the re-randomization procedure can be used for any pre-specified polynomial number of times.

To permit testing of whether a level- n encoding $u = e/z^n \in S_n$ encodes 0, GGH publishes a level- n zero-test parameter $\mathbf{p}_{zt} = hz^n/g$, where h is “somewhat small”⁴ and g is the generator of $\mathcal{I}$. The procedure **isZero(params, $\mathbf{p}_{zt}$, u)** simply computes $\mathbf{p}_{zt} \cdot u$ and tests whether its coefficients are small modulo q . If u encodes 0, then $e \in \mathcal{I}$ and equals $g \cdot c$ for some (small) c , and thus $\mathbf{p}_{zt} \cdot u = h \cdot c$ has no denominator and is small modulo q . If u encodes something nonzero, $\mathbf{p}_{zt} \cdot u$ has g in the denominator and is not small modulo q . The **ext(params, $\mathbf{p}_{zt}$, u)** procedure works by applying a strong extractor to the most significant bits of $\mathbf{p}_{zt} \cdot u$. For any two $u_1, u_2 \in S_n^{(\alpha)}$, we have (subject to noise issues) $u_1 - u_2 \in S_n^{(0)}$, which implies $\mathbf{p}_{zt}(u_1 - u_2)$ is small, and hence $\mathbf{p}_{zt} \cdot u_1$ and $\mathbf{p}_{zt} \cdot u_2$ have the same most significant bits (for an overwhelming fraction of α ’s).

Garg et al. provide an extensive cryptanalysis of the encoding system, which we will not review here. We remark that the underlying assumptions are stronger, but related to, the hardness assumption underlying the NTRU encryption scheme: that it is hard to distinguish a uniformly random element from $\mathcal{O}/(q)$ from a ratio of “small” elements – i.e., an element $u/v \in \mathcal{O}/(q)$ where $u, v \in \mathcal{O}/(q)$ both have coefficients that are on the order of (say) q^ϵ for small constant ϵ .

B Definitions for Aggregate Signatures

We introduced our general definitional setting in Section 3. Now, for our regular aggregate security definition, we define adaptive and selective variants. We also identify a slightly weaker “distinct message” security game that is easier to work with. In Appendix C, we describe and prove secure a simple transformation from distinct message security to standard aggregate signature security. The transformation captures the idea of hashing the public key and message together [12, 2] in a modular way.

An aggregate signature scheme is comprised of the following algorithms:

Setup($1^\lambda, \ell$) The trusted setup algorithm takes as input the security parameter as well the bit-length ℓ of messages. It outputs a common set of public parameters PP.

KeyGen(PP) The key generation algorithm takes as input the system public parameters and outputs a signature verification key and secret key pair (VK, SK).

⁴Its coefficients are on the order of (say) $q^{2/3}$, while other terms – such as a numerator e or the principal ideal generator g – are much, much smaller.

Sign(PP, SK, $M \in \{0, 1\}^\ell$) The signing algorithm takes as input a secret signing key, the common public parameters as well as a message $M \in \{0, 1\}^\ell$. It outputs a signature σ . *We emphasize that a single signature that is output by this algorithm is considered to also be an aggregate signature.*

Aggregate(PP, $\tilde{S}, S', \tilde{\sigma}, \sigma'$). The aggregation algorithm takes as input two multisets $\tilde{S}$ and S' , two purported signatures on these multisets and the public parameters. The elements of $\tilde{S}$ consist of verification key/message pairs $\{(\tilde{VK}_1, \tilde{M}_1), \dots, (\tilde{VK}_{|\tilde{S}|}, \tilde{M}_{|\tilde{S}|})\}$ and the elements of S' consist of $\{(VK'_1, M'_1), \dots, (VK'_{|S'|}, M'_{|S'|})\}$. The process produces a signature σ on the multiset $S = \tilde{S} \cup S'$, where $\cup$ is a multiset union.

Verify(PP, S, σ). The verification algorithm takes as input the public parameters, a multiset S of verification key/message pairs and a purported aggregate signature σ . It outputs true or false to indicate whether verification succeeded.

Correctness The correctness property states that all valid aggregate signatures will pass the verification algorithm, where a valid aggregate is defined recursively as an aggregate signature derived by an application of the aggregation algorithm on two valid inputs or the signing algorithm. More formally, for all integers $\lambda, \ell, k \geq 1$, all $PP \in \text{Setup}(1^\lambda, \ell)$, all $(VK_i, SK_i) \in \text{KeyGen}(PP)$ for $i = 1$ to k , $\text{Verify}(PP, S, \sigma) = 1$, if σ is a *valid* aggregate for multiset S under PP. We say that an aggregate signature σ is *valid* for multiset S if: (1) $S = \{(VK_i, M)\}$ for some $i \in [1, k]$, $M \in \{0, 1\}^\ell$ and $\sigma \in \text{Sign}(PP, SK_i, M)$; or (2) there exists multisets $S', \tilde{S}$ where $S = S' \cup \tilde{S}$ and valid aggregate signatures $\sigma', \tilde{\sigma}$ on them respectively such that $\sigma \in \text{Aggregate}(PP, \tilde{S}, S', \tilde{\sigma}, \sigma')$.

B.1 Security Model for Aggregate Signatures

We define the adaptive security game as in [12, 2]. Informally, it should be computationally infeasible for any adversary to produce a forgery implicating an honest signer, even when the adversary can control all other keys involved in the aggregate and can mount a chosen-message attack on the honest signer. This is defined using a game between a challenger and an adversary $\mathcal{A}$ with respect to scheme $\Pi = (\text{Setup}, \text{KeyGen}, \text{Sign}, \text{Aggregate}, \text{Verify})$.

– **Unforg**($\Pi, \mathcal{A}, \lambda, \ell$):

Setup. The challenger runs $\text{Setup}(1^\lambda, \ell)$ to obtain PP and $\text{KeyGen}(PP)$ to obtain (VK, SK) . It sends (PP, VK) to $\mathcal{A}$.

Queries. Proceeding adaptively, $\mathcal{A}$ requests signatures under VK on ℓ -bit messages of his choice.

Response. Finally, $\mathcal{A}$ outputs a multiset S^* of verification key/message pairs and a purported aggregate signature σ^* .

We say the adversary “wins” or that the output of this experiment is 1 if: (1) $\text{Verify}(PP, S^*, \sigma^*) = 1$ and (2) there exists an element $(VK, M^*) \in S^*$ such that M^* was not queried for a signature by the adversary. Otherwise, the output is 0. Define $\mathbf{Forg}_{\mathcal{A}}$ as the probability that $\mathbf{Unforg}(\Pi, \mathcal{A}, \lambda, \ell) = 1$, where the probability is over the coin tosses of the Setup, KeyGen, and Sign algorithms and of $\mathcal{A}$.

Definition B.1 (Adaptive Unforgeability) *An aggregate signature scheme Π is existentially unforgeable with respect to adaptive chosen-message attacks if for all probabilistic polynomial-time adversaries $\mathcal{A}$, the function $\mathbf{Forg}_{\mathcal{A}}$ is negligible in λ .*

Selective Security. We consider a selective variant to **Unforg** where there is an Init phase before the Setup phase, wherein $\mathcal{A}$ gives to the challenger the forgery message $M^* \in \{0, 1\}^\ell$. This message M^* cannot be queried for a signature and yet (VK, M^*) must appear in S^* .

Distinct Message Security. We consider a distinct message variant to **Unforg**, where the game is the same as above, but we change how we define the experiment output. The output of the experiment is 1 if and only if: (1) it was 1 in the **Unforg** game, and (2) the message M^* was not associated with any other signer. That is, for all $(VK_i, M_i) \in S^*$, if $VK_i \neq VK$, then $M_i \neq M^*$. (The forgery message M^* could be associated with the key VK multiple times. This is allowed.)

C Transforming Distinct Message Security into Standard Security

In this section, we show how to transform any aggregate signature scheme proved in the distinct message security game into one which is secure in the standard security game. This will apply to both the selective and full security games. We remind the reader that distinct message security is not used in the ID-based setting, so we consider only regular signatures here.

The transformation essentially captures and modularizes idea of Boneh, Gentry, Lynn and Shacham [12], which was formally captured by Bellare, Namprempre, and Neven [2], of hashing the public key and message to get an output that is plugged in as the message for the core scheme. To execute this transformation the message length, ℓ , of the core scheme must be as large as the output of a collision-resistant hash function. We give the construction.

Let $\Pi = (\text{Setup}, \text{KeyGen}, \text{Sign}, \text{Aggregate}, \text{Verify})$ be an aggregate signature scheme for message length ℓ . Let $H : \{0, 1\}^* \rightarrow \{0, 1\}^\ell$ be a collision resistant hash function. We build a second aggregate signature scheme Π' derived from Π as follows. The public parameters now include the description of H . Keys are generated as before. To sign a message $M \in \{0, 1\}^*$ for the user associated with verification key VK , first compute $M' = H(VK, M)$ and then run the regular signing algorithm on M' . Aggregation works the same as before. The verification algorithm recomputes $M'_i = H(VK_i, M_i)$ for each $(VK_i, M_i) \in S$ and treats these as the messages in the regular verification algorithm.

Lemma C.1 (Distinct Message to Standard Transformation) *If Π is an adaptively (resp., selectively), distinct message secure aggregate signature scheme for message length ℓ and $H : \{0, 1\}^* \rightarrow \{0, 1\}^\ell$ is a collision-resistant hash function, then Π' as defined above is an adaptively (resp., selectively) secure aggregate signature scheme.*

Proof. We argue that any PPT adversary $\mathcal{A}'$ against Π' can be turned into a PPT adversary $\mathcal{A}$ that breaks either Π in the distinct message game or finds a collision in H . Let σ^* be the forgery on S^* submitted by the adversary at the end of the standard security game. Let $M'_1, \dots, M'_{|S|}$ be derived as $M'_i = H(VK_i, M_i)$ for each entry (VK_i, M_i) in S^* . We consider two cases.

First, suppose there exists some M'_i, M'_j such that $M'_i = M'_j$ and yet $(VK_i, M_i) \neq (VK_j, M_j)$. Then, the simulator can use the adversary to find a collision in the hash function since $H(VK_i, M_i) = H(VK_j, M_j)$ and the inputs are not equal.

Otherwise, it must be the case that if $M'_i = M'_j$, then $(VK_i, M_i) = (VK_j, M_j)$. Thus, the adversary is not violating the distinct message property since there cannot be $VK_i \neq VK_j$ where $M'_i = M'_j$. The simulator can reduce to the security of the underlying distinct message secure scheme. $\square$

There are alternatives to proving security in the distinct message setting and then applying the above transformation, which have been explored in prior works. One possibility is to require public keys to be registered with some authority, where registration is contingent upon proving knowledge of the secret key to the authority. Verification only proceeds if the public key includes a registration certificate from the authority. Alternatively, one could include a non-interactive zero knowledge proof of knowledge of the private key as part of the private key. Verification only proceeds after the verifier checks the NIZKs. While we choose the distinct message plus transformation route, we expect these other alternatives would be viable with only minor technical modifications.

D Security of the Base Construction

We provide three claims on the security of the generic, base construction from Section 4.1. The proofs for the translation to the GGH framework follow along the same lines from these and the proof of the ID-Based construction in the GGH framework.

D.1 Security against Selective or Subexponential Advantage Attacks

The k -Multilinear Computational Diffie-Hellman (k -MDDH) assumption was defined in Section 4.3. We establish full security of our basic aggregate signature scheme conditioned on the k -MCDH assumption holding against subexponential advantage. This follows immediately from the following theorem and a standard complexity leveraging argument:

Theorem D.1 (Selective Security of Base Construction) *The aggregate signature scheme for message length ℓ in Section 4.1 is selectively secure in the distinct message unforgeability game under the $(\ell + 1)$ -MCDH assumption.*

Corollary D.2 *The aggregate signature scheme for message length ℓ in Section 4.1 is fully secure in the distinct message unforgeability game under the $(\ell + 1)$ -MCDH assumption against subexponential advantage.*

Proof. This follows immediately from a complexity leveraging argument: the security parameter λ is chosen to ensure that $2^{\lambda^{\epsilon_0}} \gg 2^\ell$, where $2^{-\lambda^{\epsilon_0}}$ is the maximum probability of success allowed in the k -MCDH assumption against subexponential advantage. Now, to establish full security, the simulator performs exactly as in the selective security proof, but first it simply guesses the message that will be forged (instead of expecting the adversary to produce this message). Because this guess will be correct with probability at least $2^{-\ell}$, and the security parameter λ is chosen carefully, full security with polynomial advantage (or even appropriately defined subexponential advantage) implies an attacker on the k -MCDH assumption with subexponential advantage. $\square$

Proof. (of Theorem D.1) We show that if there exists a PPT adversary $\mathcal{A}$ that can break the selective security of the aggregate signature scheme in the distinct message unforgeability game with probability ϵ for message length ℓ and security parameter λ , then there exists a PPT simulator that can break the $(\ell + 1)$ -MCDH assumption for security parameter λ with probability ϵ . The simulator takes as input a MCDH instance $(g, g^{c_1}, \dots, g^{c_k})$ together with the group descriptions where $k = \ell + 1$. The simulator plays the role of the challenger in the game as follows.

Init. Let $M^* \in \{0, 1\}^\ell$ be the forgery message output by $\mathcal{A}$.

Setup. The simulator chooses random $x_1, \dots, x_\ell \in \mathbb{Z}_p$. Let $A_{i,m_i^*} = g^{c_i}$ and $A_{i,\bar{m}_i^*} = g^{x_i}$. Set the challenge verification key as $VK^* = g^{c_k}$. We remark that the parameters are distributed independently and uniformly at random as in the real scheme.

Queries. The simulator can sign any of $\mathcal{A}$'s message requests using the multilinear map. Let $M \in \{0,1\}^\ell$ be the request. The key point is that since $M \neq M^*$ there will be at least one i where x_i is used (and x_i is known to the simulator). So at most ℓ of the $k = \ell + 1$ parameters associated with c_i will need to be accumulated to make σ which is doable. That is, letting β be the first index such that $m_\beta \neq m_\beta^*$, compute the pairing of all $\ell - 1$ elements A_{i,m_i} where $i \neq \beta$ together with VK^* and denote this σ' . This requires $\ell - 1$ pairings resulting in an element in $\mathbb{G}_{k-1}$. Next compute $\sigma = \sigma'^{x_\beta}$ and return σ . Signatures are unique and perfectly distributed as in the real game.

Response. Eventually, $\mathcal{A}$ outputs an aggregate signature σ^* on multiset S^* where $(VK^*, M^*) \in S^*$. The simulator will extract from this a solution to the MCDH problem. This works by iteratively computing all the other signatures in S^* and then dividing them out of the aggregate until only one or more signatures on (VK^*, M^*) remain. That is, the simulator takes an aggregate for S^* and computes an aggregate signature for S' where S' has one less verification key/message pair than S at each step. These signatures will be peeled off in one of two ways. If the signature is under key VK^* , then it can be computed as in the query phase. If the signature is under key $VK \neq VK^*$, then due to the distinct message restriction, we know $M \neq M^*$. Thus, there is some β where $m_\beta \neq m_\beta^*$. The signature can be computed similarly to the query phase by pairing VK with all A_{i,m_i} where $i \neq \beta$ and then raising the result to x_β . To help see why this works, recall that these signatures are unique.

Eventually, we have an aggregate σ' on $t \geq 1$ instances of (VK^*, M^*) . We have that $e(\sigma', g) = e(H(M^*), VK^*)^t = g_k^{t \prod_{i=1}^k c_i}$ and thus $\sigma' = g_{k-1}^{t \prod_{i=1}^k c_i}$. The simulator computes $\sigma'^{1/t}$ (recall that t is not 0 mod p) which gives $g_{k-1}^{\prod_{i=1}^k c_i}$ and this is given as the solution to the MCDH problem.

As remarked in the Setup and Query phase, the responses of the challenger are distributed identically to the real unforgeability game. The simulator succeeds whenever $\mathcal{A}$ does. $\square$

D.2 Security against Adaptive Attacks

We now give an adaptive proof of security under a *polynomial* assumption (as opposed to the subexponential advantage assumption necessary to achieve full security given previously). We will employ a parameterized assumption family, where the choice of assumption depends on the adversary's behavior. It can be viewed as a modification/adaptation of the computation Bilinear-Diffie Hellman Assumption (introduced by Boneh, Boyen, and Goh [10]) to the multilinear map setting.

Assumption D.3 ((n, k)-Modified Multilinear Computational Diffie-Hellman Exponent)

The (n, k)-Modified Multilinear Computational Diffie-Hellman Exponent ((n, k)-MMCDHE) problem states the following: A challenger runs $\mathcal{G}(1^\lambda, k)$ to generate groups and generators of order p . Then it picks random $a, b, c \in \mathbb{Z}_p$.

The assumption then states that given

$$g = g_1, g^b, \forall i \in [1, n] \ g^{a^i c}, \forall i \neq n \in [1, 2n] \ (g_{k-2})^{a^i c^{k-1}}$$

it is hard to compute $(g_{k-1})^{a^n c^{k-1} b}$ with better than negligible advantage (in security parameter λ).

The above assumption is only defined for $k \geq 3$ due to the reference of the g_{k-2} generator.

Intuitively, our proof will follow in the Waters [40] framework. Waters gave a technique for partitioning approximately a (hidden) $1/Q$ fraction of messages to be useful as challenge forgeries and the other $1 - 1/Q$ to be messages a reduction algorithm could create signatures on.⁵ Typically, one sets Q to be the maximum number of queries made by the adversary, although in this case, we must also add in the messages involved in the adversary's forgery aggregate signature. We will use a multiplicative analog of the technique which is closer to the VRF analysis of Hohenberger and Waters [29].

Theorem D.4 (Adaptive Security of Base Construction) *The aggregate signature scheme for message length ℓ in Section 4.1 is adaptively secure in the distinct message unforgeability game under the $(4Q(\ell + 2), \ell + 1)$ -MMCDHE assumption, where Q is the number of signing queries made by the adversary plus one minus the number of distinct messages in the forgery aggregate.*

Proof. We show that if there exists a PPT adversary $\mathcal{A}$ that can break the adaptive security of the aggregate signature scheme in the distinct message unforgeability game with probability ϵ for message length ℓ , security parameter λ , making at most Q signing queries and Q' distinct messages in the forgery aggregate, then there exists a PPT simulator that can break the (n, k) -MMCDHE assumption for security parameter λ with probability $\geq \frac{3\epsilon}{64Q(\ell+1)}$.

The simulator takes as input an MMCDHE instance

$$(g, g^b, \forall i \in [1, n] \ g^{a^i c}, \forall i \neq n \in [1, 2n] \ (g_{k-2})^{a^i c^{k-1}})$$

together with the group descriptions where $n = 4Q(\ell + 2)$ and $k = \ell + 1$. The simulator's challenge is to compute $(g_{k-1})^{a^n c^{(k-1)} b} = (g_{k-1})^{a^n c^\ell b}$. The simulator plays the role of the challenger in the game as follows.

Setup. The simulator first sets an integer $z = 4Q$ and chooses an integer t uniformly at random between 0 and ℓ . Recall that Q is the number of queries made by the adversary plus one minus the number of distinct messages in S^* that forms the forgery aggregate and ℓ is the message bit-length. It then chooses random integers $r_{1,0}, r_{1,1}, \dots, r_{\ell,0}, r_{\ell,1}, r'$ between 0 and $z - 1$. Additionally, the simulator chooses random values $s_{1,0}, s_{1,1}, \dots, s_{\ell,0}, s_{\ell,1} \in \mathbb{Z}_p^*$. These values are all kept internal to the simulator. Intuitively, the r values will be used to embed the challenge, while the s values will be used as blinding factors to present the proper distribution to the adversary.

Let m_i denote the i th bit of M . For $M \in \{0, 1\}^\ell$, define the functions:

$$C(M) = zt + r' + \sum_{i=1}^{\ell} r_{i,m_i} \quad , \quad J(M) = \prod_{i=1}^{\ell} s_{i,m_i}$$

⁵There exists some variants of this technique [3, 28, 27] with different loss tradeoffs. We believe these tradeoffs are applicable to our setting, but we choose to stick closest to the original analysis.

For $M \in \{0, 1\}^\ell$, define the binary function:

$$K(M) = \begin{cases} 0 & \text{if } r' + \sum_{i=1}^\ell r_{i,m_i} \equiv 0 \pmod{z}; \\ 1 & \text{otherwise.} \end{cases}$$

If the function K outputs 1 on a given message, then we know that the simulator will be able to correctly produce a signature on this message. If the function outputs 0, then the simulator may or may not be able to do it. This function will be used in later analysis.

The simulator sets the public parameters as $A_{1,0} = (g^{a^{(zt+r'+r_{1,0})c}})^{s_{1,0}}$, $A_{1,1} = (g^{a^{(zt+r'+r_{1,1})c}})^{s_{1,1}}$, and $A_{i,0} = (g^{a^{r_{i,0}c}})^{s_{i,0}}$ and $A_{i,1} = (g^{a^{r_{i,1}c}})^{s_{i,1}}$ for $i = 2$ to ℓ . The simulator can compute these values from the challenge input since all powers of a in the exponent are at most $zt + 2(z - 1) = 4Q(\ell + 2) - 2 < n$ for any possible choice of r', r_i, t . It sets the challenge verification key as $VK^* = g^b$. It passes the public information to the adversary. We remark that the parameters are distributed independently and uniformly at random as in the real scheme.

Queries. The adversary will ask for signatures under the challenge verification key. On message input M , the simulator first checks if $C(M) = n$ and aborts if this is true. Otherwise, it outputs the signature as

$$\sigma = e(g^b, g_{k-2}^{a^{C(M)c^{k-1}}})^{J(M)} = g_{k-1}^{ba^{C(M)c^\ell J(M)}}.$$

Given the above settings, we can verify that for any value of $M \in \{0, 1\}^\ell$, the maximum value of $C(M)$ is $z\ell + (\ell + 1)(z - 1) < 2z(\ell + 1) = 2n$. Thus, if $C(M) \neq n$, then the simulator can correctly compute the signature.

Response. Eventually, the adversary will output a multiset S^* of verification key/message pairs and a purported aggregate signature σ^* such that:

1. $\text{Verify}(\text{PP}, S^*, \sigma^*) = 1$, and
2. there exists an element $(VK, M^*) \in S^*$ such that M^* was not one of the adversary's signature query inputs, and
3. the message M^* is not signed by any other signer. (This is the distinct message requirement.)

If $C(M^*) \neq n$, then the simulator will abort. The goal is that the forgery message will fall into the “hole” of the assumption and that all other messages will not.

If $C(M^*) = n$, then the simulator will now work to extract a forgery on M^* from the aggregate by calculating the other signatures and then removing them from the aggregate. These can come both from the challenge signer and other signers. The simulator does this as follows:

Signatures from other signers. For (VK', M) where $VK' = g^{b'} \neq VK$, if $C(M) = n$, then the simulator aborts. Otherwise, it computes the signature as

$$\sigma = e(VK', g_{k-2}^{a^{C(M)c^{k-1}}})^{J(M)} = g_{k-1}^{b'a^{C(M)c^\ell J(M)}}.$$

Signatures from challenge signer. For (VK, M) where $M \neq M^*$, if $C(M) = n$, then the simulator aborts. Otherwise, it computes the signature as

$$\sigma = e(g^b, g_{k-2}^{a^{C(M)}c^{k-1}})^{J(M)} = g_{k-1}^{ba^{C(M)}c^\ell J(M)}.$$

Extracting the response. Once these signatures are calculated they can be removed from the aggregate by division, resulting in an aggregate on $w \geq 1$ (non-multiple of p) signatures by the challenge signer on M^* . The uniqueness of this scheme dictates that this aggregate is:

$$\sigma' = ((g_{k-1})^{ba^{C(M^*)}c^\ell})^{J(M^*)w} = ((g_{k-1})^{ba^nc^{k-1}})^{J(M^*)w}$$

and raising σ' to $1/(J(M^*)w)$ results in $(g_{k-1})^{ba^nc^{k-1}}$, the solution to the MMCDHE instance. Recall that w is not a multiple of the group order p . $J(M^*)$ is a product of elements from $\mathbb{Z}_p^*$ where p is prime and therefore will also have an inverse modulo p .

A Series of Games Analysis. We now argue that any successful adversary $\mathcal{A}$ against our scheme will have success in the game presented by the simulator. To do this, we first define a sequence of games, where the first game models the real security game and the final game is exactly the view of the adversary when interacting with the simulator. We then show via a series of claims that if $\mathcal{A}$ is successful in Game j , then it will also be successful in Game $j + 1$.

Game 1: This game is defined to be the same as the distinct message unforgeability game.

Game 2: The same as Game 1, with the exception that we keep a record of each signing query made by $\mathcal{A}$ concatenated together with each *distinct* message in the forgery multiset S^* minus M^* . We'll denote $\vec{M} = (M_0, M_1, M_2, \dots, M_Q)$. Without loss of generality, let $M_0 = M^*$. At the end of the game, we set $z = 4Q$ and choose random integers $\vec{r} = (r_{1,0}, r_{1,1}, \dots, r_{\ell,0}, r_{\ell,1}, r')$ between 0 and $z - 1$ and a random integer t between 0 and ℓ . We define the regular abort function:

$$\text{regabort}(\vec{M}, \vec{r}, t) = \begin{cases} 1 & \text{if } C(M^*) \neq n \vee_{i=1}^Q K(M_i) = 0; \\ 0 & \text{otherwise.} \end{cases}$$

This function evaluates to 0 if the queries and forgery messages will not cause a regular abort by the simulator for the given choice of simulation values. Consider the probability over all simulation values for the given set of queries and forgery messages as $\zeta(\vec{M}) = \Pr_{\vec{r}, t}[\text{regabort}(\vec{M}, \vec{r}, t) = 0]$.

As in [40], the simulator estimates $\zeta(\vec{M})$ as ζ' by evaluating $\tau(\vec{M}, \vec{r}, t)$ with fresh random $\vec{r}, t$ values a total of $O(\epsilon^{-2} \ln(\epsilon^{-1}) \zeta_{\min}^{-1} \ln(\zeta_{\min}^{-1}))$ times, where $\zeta_{\min} = \frac{1}{8Q(\ell+1)}$. This does not require running the adversary again.

The adversary's success in the game is determined as follows:

1. *Regular Abort.* If $\text{regabort}(\vec{M}, \vec{r}, t) = 1$, then the adversary wins.
2. *Balancing (Artificial) Abort.* Let $\zeta_{\min} = \frac{1}{8Q(\ell+1)}$ as derived from Claim D.5. If $\zeta' \geq \zeta_{\min}$, the simulator will abort with probability $\frac{\zeta' - \zeta_{\min}}{\zeta'}$ (not abort with probability $\frac{\zeta_{\min}}{\zeta'}$). If it aborts, then the adversary wins.

3. Otherwise, the adversary wins if and only if it outputs a valid forgery.

Game 3: The same as Game 2, with the exception that the simulator tests if any abort conditions are satisfied, with each new query or response from the adversary, and if so, follows the abort procedure immediately instead of waiting until the end.

Game 3 is exactly the view of the adversary when interacting with the simulator. We will shortly prove that if $\mathcal{A}$ succeeds in Game 1 with probability ϵ , then it succeeds in Game 3 with probability $\geq \frac{3\epsilon}{64Q(\ell+1)}$.

Claims Regarding the Probability of Aborting. We now establish one claim which was referenced above and two claims which will be needed shortly. Our first claim helps us establish a minimum probability that a given set of queries/forgery sets do not cause a *regular* abort. We use this minimum during our balancing abort in Game 2, to “even out” the probability of an abort over all possible queries/forgery sets. In the next two claims, we employ Chernoff Bounds to establish upper and lower bounds for *any* abort (regular or balancing) for any adversary behavior. The latter two claims will be used in the analysis of the adversary’s probability of success in Game 2.

Proofs of these claims are similar to related arguments in [40, 29], but we include them here for completeness.

Claim D.5 *Let $\zeta_{\min} = \frac{1}{8Q(\ell+1)}$. For any vector $\vec{M}$, $\zeta(\vec{M}) \geq \zeta_{\min}$.*

Proof. In other words, the probability of the simulation *not* triggering a general abort is at least $\zeta_{\min}$. This analysis follows that of [40], which we reproduce here for completeness. Without loss of generality, we can assume the adversary always makes the maximum Q queries and number of distinct messages in the output forgery set (since the probability of not aborting increases with fewer queries/smaller output set size). Fix an arbitrary $\vec{M} = (M^*, M_1, \dots, M_Q) \in \{0, 1\}^{(Q+1) \times \ell}$.

Then, with the probability over the choice of $\vec{r}, t$, we have that $\Pr[\overline{\text{abort}} \text{ on } \vec{M}]$ is

$$= \Pr\left[\bigwedge_{i=1}^Q K(M_i) = 1 \wedge C(M^*) = n\right] \quad (1)$$

$$= (1 - \Pr[\bigvee_{i=1}^Q K(M_i) = 0]) \Pr[(zt + r' + \sum_{i=1}^{\ell} r_{i,m_i^*} = n) | \bigwedge_{i=1}^Q K(M_i) = 1] \quad (2)$$

$$\geq (1 - \sum_{i=1}^Q \Pr[K(M_i) = 0]) \Pr[(zt + r' + \sum_{i=1}^{\ell} r_{i,m_i^*} = n) | \bigwedge_{i=1}^Q K(M_i) = 1] \quad (3)$$

$$= (1 - \frac{Q}{z}) \cdot \Pr[(zt + r' + \sum_{i=1}^{\ell} r_{i,m_i^*} = n) | \bigwedge_{i=1}^Q K(M_i) = 1] \quad (4)$$

$$= \frac{1}{\ell+1} \cdot (1 - \frac{Q}{z}) \cdot \Pr[K(M^*) = 0 | \bigwedge_{i=1}^Q K(M_i) = 1] \quad (5)$$

$$= \frac{1}{\ell+1} \cdot (1 - \frac{Q}{z}) \cdot \frac{\Pr[K(M^*) = 0] \cdot \Pr[\bigwedge_{i=1}^Q K(M_i) = 1 | K(M^*) = 0]}{\Pr[\bigwedge_{i=1}^Q K(M_i) = 1]} \quad (6)$$

$$\geq \frac{1}{(\ell+1)z} \cdot (1 - \frac{Q}{z}) \cdot \Pr[\bigwedge_{i=1}^Q K(M_i) = 1 | K(M^*) = 0] \quad (7)$$

$$= \frac{1}{(\ell+1)z} \cdot (1 - \frac{Q}{z}) \cdot (1 - \Pr[\bigvee_{i=1}^Q K(M_i) = 0 | K(M^*) = 0]) \quad (8)$$

$$\geq \frac{1}{(\ell+1)z} \cdot (1 - \frac{Q}{z}) \cdot (1 - \sum_{i=1}^Q \Pr[K(M_i) = 0 | K(M^*) = 0]) \quad (9)$$

$$= \frac{1}{(\ell+1)z} \cdot (1 - \frac{Q}{z})^2 \quad (10)$$

$$\geq \frac{1}{(\ell+1)z} \cdot (1 - \frac{2Q}{z}) \quad (11)$$

$$= \frac{1}{8Q(\ell+1)} \quad (12)$$

Equations 4 and 7 derive from $\Pr[K(M) = 0] = \frac{1}{z}$ for any query M . Equation 5 gets a factor of $\frac{1}{\ell+1}$ from the simulator taking a guess of t . Equation 6 follows from Bayes' Theorem. Equation 10 follows from the pairwise independence of the probabilities that $K(M) = 0, K(M') = 0$ for any pair of queries $M \neq M'$, since they will differ in at least one random r_j value. Equation 12 follows from our setting of $z = 4Q$. $\square$

Claim D.6 *For any vector $\vec{M}$, the probability that there is an abort (i.e., regular or balancing) is $\geq 1 - \zeta_{\min} - \frac{3}{8}\zeta_{\min}\epsilon$.*

Proof. Let $\zeta_x = \zeta(\vec{M})$ be the probability that the set of queries/forgery messages $\vec{M}$ do not cause a regular abort. In Game 2, $T = O(\epsilon^{-2} \ln(\epsilon^{-1}) \zeta_{\min}^{-1} \ln(\zeta_{\min}^{-1}))$ samples are taken to approximate this

value as ζ'_x . By Chernoff Bounds, we have that for all $\vec{M}$,

$$\Pr[T\zeta'_x < T\zeta_x(1 - \frac{\epsilon}{8})] < e^{-[128\epsilon^{-2} \ln((\epsilon/8)^{-1})\zeta_{\min}^{-1} \ln(\zeta_{\min}^{-1})(\zeta_{\min})(\epsilon/8)^2/2]},$$

which reduces to

$$\Pr[\zeta'_x < \zeta_x(1 - \frac{\epsilon}{8})] < \zeta_{\min} \frac{\epsilon}{8}.$$

The probability of not aborting is equal to the probability of not regular aborting (RA) times the probability of not artificial aborting (AA). Recall that for a measured ζ'_x an artificial abort will not happen with probability $\zeta_{\min}/\zeta'_x$. The probability of aborting is therefore

$$\begin{aligned} \Pr[\text{abort}] &= 1 - \Pr[\overline{\text{abort}}] = 1 - \Pr[\overline{\text{RA}}] \Pr[\overline{\text{AA}}] = 1 - \zeta_x \Pr[\overline{\text{AA}}] \\ &\geq 1 - \zeta_x(\zeta_{\min} \frac{\epsilon}{8} + \frac{\zeta_{\min}}{\zeta_x(1 - \epsilon/8)}) \\ &\geq 1 - (\zeta_{\min} \frac{\epsilon}{8} + \frac{\zeta_{\min}}{1 - \epsilon/8}) \\ &\geq 1 - (\frac{\zeta_{\min}\epsilon}{8} + \zeta_{\min}(1 + \frac{2\epsilon}{8})) \\ &\geq 1 - \zeta_{\min} - \zeta_{\min} \frac{3\epsilon}{8} \end{aligned}$$

□

Claim D.7 For any vector $\vec{M}$, the probability that there is no abort (i.e., regular or balancing) is $\geq \zeta_{\min} - \frac{1}{4}\zeta_{\min}\epsilon$.

Proof. Let $\zeta_x = \zeta(\vec{M})$ be the probability that the set of queries/forgery messages $\vec{M}$ do not cause a regular abort. In Game 2, $T = O(\epsilon^{-2} \ln(\epsilon^{-1})\zeta_{\min}^{-1} \ln(\zeta_{\min}^{-1}))$ samples are taken to approximate this value as ζ'_x . By Chernoff Bounds, we have that for all $\vec{M}$,

$$\Pr[T\zeta'_x > T\zeta_x(1 + \frac{\epsilon}{8})] < e^{-[256\epsilon^{-2} \ln((\epsilon/8)^{-1})\zeta_{\min}^{-1} \ln(\zeta_{\min}^{-1})(\zeta_{\min})(\epsilon/8)^2/4]},$$

which reduces to

$$\Pr[\zeta'_x > \zeta_x(1 + \frac{\epsilon}{8})] < \zeta_{\min} \frac{\epsilon}{8}.$$

Recall that for a measured ζ'_x an artificial abort (AA) will not happen with probability $\zeta_{\min}/\zeta'_x$. Therefore, for any $\vec{M}$, the $\Pr[\overline{\text{AA}}] \geq (1 - \frac{\zeta_{\min}\epsilon}{8}) \frac{\zeta_{\min}}{\zeta_x(1 + \epsilon/8)}$. It follows that

$$\Pr[\overline{\text{abort}}] \geq \zeta_x(1 - \frac{\zeta_{\min}\epsilon}{8}) \frac{\zeta_{\min}}{\zeta_x(1 + \epsilon/8)} \geq \zeta_{\min}(1 - \frac{\epsilon}{8})^2 \geq \zeta_{\min}(1 - \frac{1}{4}\epsilon).$$

□

Analyzing $\mathcal{A}$'s Probability of Winning in Each Game. Define $\mathcal{A}$'s probability of success in Game x as $\text{Adv}_{\mathcal{A}}[\text{Game } x]$. We reason about the probability of $\mathcal{A}$'s success in the series of games as follows.

Lemma D.8 *If $\text{Adv}_{\mathcal{A}}[\text{Game } 1] = \epsilon$, then $\text{Adv}_{\mathcal{A}}[\text{Game } 2] \geq \frac{3 \cdot \epsilon}{64Q(\ell+1)}$.*

Proof. We begin by observing that $\text{Adv}_{\mathcal{A}}[\text{Game } 2]$ is

$$= \text{Adv}_{\mathcal{A}}[\text{Game } 2 | \text{abort}] \cdot \Pr[\text{abort}] + \text{Adv}_{\mathcal{A}}[\text{Game } 2 | \overline{\text{abort}}] \cdot \Pr[\overline{\text{abort}}] \quad (13)$$

$$= \Pr[\text{abort}] + \text{Adv}_{\mathcal{A}}[\text{Game } 2 | \overline{\text{abort}}] \cdot \Pr[\overline{\text{abort}}] \quad (14)$$

$$= \Pr[\text{abort}] + \Pr[\mathcal{A} \text{ forges } | \overline{\text{abort}}] \cdot \Pr[\overline{\text{abort}}] \quad (15)$$

$$= \Pr[\text{abort}] + \Pr[\mathcal{A} \text{ forges}] \cdot \Pr[\overline{\text{abort}} | \mathcal{A} \text{ forges}] \quad (16)$$

$$= \Pr[\text{abort}] + \epsilon \cdot \Pr[\overline{\text{abort}} | \mathcal{A} \text{ forges}] \quad (17)$$

$$\geq (1 - \zeta_{\min} - \zeta_{\min} \frac{3\epsilon}{8}) + \epsilon \cdot (\zeta_{\min} - \zeta_{\min} \frac{\epsilon}{4}) \quad (18)$$

$$\geq \frac{3 \cdot \epsilon \cdot \zeta_{\min}}{8} \quad (19)$$

$$= \frac{3 \cdot \epsilon}{64Q(\ell+1)} \quad (20)$$

Equation 14 follows from the fact that, in the case of abort, $\mathcal{A}$ always wins. It would be very convenient if we could claim that $\text{Adv}_{\mathcal{A}}[\text{Game } 2 | \overline{\text{abort}}] = \text{Adv}_{\mathcal{A}}[\text{Game } 1]$, but unfortunately, this is false. The event that $\mathcal{A}$ wins Game 2 and the event of an abort are not independent; however, we have inserted the balancing abort condition in the attempt to lessen the dependence between these events. Equation 15 simply states that, when there is no abort, $\mathcal{A}$ wins if and only if it forges correctly. Equation 16 follows from Bayes' Theorem. In Equation 17, we observe that $\Pr[\mathcal{A} \text{ forges}]$ is exactly $\mathcal{A}$'s success in Game 1.

Now, the purpose of our balancing abort is to even the probability of aborting, for all queries and outputs of $\mathcal{A}$, to be roughly $\zeta_{\min}$. This will also get rid of the conditional dependence on $\mathcal{A}$ forging. There will be a small error, which must be taken into account. We set $\zeta_{\min} = \frac{1}{8Q(\ell+1)}$ from Claim D.5. We know, for all queries/outputs, that $\Pr[\text{abort}] \geq 1 - \zeta_{\min} - \frac{3}{8}\zeta_{\min}\epsilon$ from Claim D.6 and that $\Pr[\overline{\text{abort}}] \geq \zeta_{\min} - \frac{1}{4}\zeta_{\min}\epsilon$ from Claim D.7. Plugging these values into Equations 18 and 20 establishes the lemma. $\square$

Lemma D.9 $\mathcal{A}_D[\text{Game } 3] = \mathcal{A}_D[\text{Game } 2]$.

Proof. We make the explicit observation that these games are equivalent by observing that their only difference is the time at which the regular aborts occur. The artificial abort stage is identical. All public parameters and signatures provided by the simulator have the same distribution. $\square$

$\square$

E The Base Aggregate Construction in the GGH framework

We now describe how to modify the construction of Section 4.1 to use the GGH [21] graded algebras analogue of multilinear maps. The translation of our scheme above is straightforward to the GGH

setting. Please note that we use the same notation developed in [21], with some minor changes: Firstly, we use the canonical encoding function `cenc` provided by the GGH framework more than once at each level of the encoding, but only a globally fixed constant number of times per level. This is compatible with the GGH encoding [21], and allows for a simpler exposition of our scheme and proof. Also, **for ease of notation on the reader, we suppress repeated `params` arguments that are provided to every algorithm.** Thus, for instance, we will write $\alpha \leftarrow \text{samp}()$ instead of $\alpha \leftarrow \text{samp}(\text{params})$. Note that in our scheme, there will only ever be a single uniquely chosen value for `params` throughout the scheme, so there is no cause for confusion. Finally, we use the variant of the GGH framework with “strong” zero-testing, where the zero test statistically guarantees that a vector is a valid encoding of zero if it passes the zero test. For further details on the GGH framework, please refer to [21].

Setup($1^\lambda, \ell$) The trusted setup algorithm takes as input the security parameter as well as the length ℓ of messages. It then runs $(\text{params}, \mathbf{p}_{zt}) \leftarrow \text{InstGen}(1^\lambda, 1^{k=\ell+1})$. Recall that `params` will be implicitly given as input to all GGH-related algorithms below.

Next, it generates elements $(A_{1,0}, A_{1,1}), \dots, (A_{\ell,0}, A_{\ell,1})$, each equal to a fresh invocation of `cenc`₁(1, `samp`()).

These will be used to compute a function H mapping ℓ bit messages to level $k - 1$ encodings. This function serves as the analog of the full domain hash function of the BGLS [12] construction. Let $m_1, \dots, m_\ell$ be the bits of message M . It is computed iteratively as

$$H_1(M) = A_{1,m_1} \quad \text{for } i \in [2, \ell] \quad H_i(M) = H_{i-1}(M) \cdot A_{i,m_i}.$$

We define $H(M) = \text{cenc}_2(k - 1, H_\ell(M))$.

The public parameters, PP, consist of the `params`, $\mathbf{p}_{zt}$ plus:

$$(A_{1,0}, A_{1,1}), \dots, (A_{\ell,0}, A_{\ell,1})$$

Note that `params` includes a level 1 encoding of 1, which we denote as g .

KeyGen(PP) The key generation algorithm first chooses random $\alpha = \text{samp}()$. It outputs the public verification key as

$$\text{VK} = \text{cenc}_2(1, \alpha).$$

The secret key SK is α .

Sign(PP, SK, $M \in \{0, 1\}^\ell$) The signing algorithm computes the signature as

$$\sigma = \text{cenc}_3(k - 1, H(M) \cdot \alpha).$$

This serves as an aggregate signature for the (single element) multiset $S = \{(\text{VK}, M)\}$.

Aggregate(PP, $\tilde{S}, S', \tilde{\sigma}, \sigma'$). The aggregation algorithm simply computes the output signature σ as $\sigma = \tilde{\sigma} + \sigma'$. The serves as a signature on the multiset $S = \tilde{S} \cup S'$, where $\cup$ is a *multiset union*.

Verify(PP, S, σ). The verification algorithm parses S as $\{(VK_1, M_1), \dots, (VK_{|S|}, M_{|S|})\}$. It rejects if the multiplicity of any public key, message pair is greater than 2^λ . We don't expect this to naturally occur much in practice.

The algorithm then proceeds to check the signature by setting $\tau = \text{cenc}_2(1, g)$, and testing:

$$\text{isZero} \left(\mathbf{p}_{zt}, \tau \cdot \sigma - \sum_{i=1, \dots, |S|} H(M_i) \cdot \text{VK}_i \right)$$

and accepts if and only if the zero testing procedure outputs true. Recall that g above is a canonical level 1 encoding of 1 that is included in params , part of the public parameters.

Correctness. Correctness follows from the same argument as for the “basic” aggregate signature scheme in the generic multilinear setting.

Proof of Security. In Section 5.2, we generalize this construction to provide ID-based aggregate signatures in the GGH framework. We provide a proof of selective security for the ID-based version of this scheme in the GGH framework, based on a variant of the MCDH assumption. Since our main focus is the ID-based aggregate signature scheme, we omit the formal proof of selective security for this scheme, but we note that it would be essentially identical to the proof of the ID-based scheme that we give in Section 5.3.

Efficiency and Tradeoffs. An aggregate signature is one level $k-1$ encoding, independent of the number of messages aggregated. In a multilinear setting, the space to represent an encoding might grow with k (which is $\ell + 1$). Indeed, this happens in the GGH [21] graded algebra translation. One way to mitigate this is to differ the message alphabet size in a tradeoff of computation versus storage. The above construction uses a binary message alphabet. If it used an alphabet of 2^d symbols, then the aggregate signature could be an ℓ/d level encoding, with $\ell/d - 1$ multiplications required to compute it, at the cost of the public parameters requiring $2^d \ell$ encodings in order to define the hash function H .

F Proof of Security for the Generic ID-Based Construction

The k -MCDH assumption is defined in Appendix D.1.

Theorem F.1 (Selective Security of ID-Based Construction) *The ID-based aggregate signature scheme for message length ℓ and identity length n in Section 5.1 is selectively secure in the unforgeability game in Section 3 under the $(\ell + n)$ -MCDH assumption.*

Proof. We show that if there exists a PPT adversary $\mathcal{A}$ that can break the selective security of the ID-based aggregate signature scheme in the unforgeability game with probability ϵ for message length ℓ , identity length n and security parameter λ , then there exists a PPT simulator that can break the $(\ell + n)$ -MCDH assumption for security parameter λ with probability ϵ . The simulator takes as input a MCDH instance $(g, g^{c_1}, \dots, g^{c_k})$ together with the group descriptions where $k = \ell + n$. Let m_i denote the i th bit of M and id_i denote the i th bit of $\mathcal{I}$. The simulator plays the role of the challenger in the game as follows.

Init. Let $\mathcal{I}^* \in \{0, 1\}^n$ and $M^* \in \{0, 1\}^\ell$ be the forgery identity/message pair output by $\mathcal{A}$.

Setup. The simulator chooses random $x_1, \dots, x_\ell, y_1, \dots, y_n \in \mathbb{Z}_p$. For $i = 1$ to ℓ , let $A_{i,m_i^*} = g^{c_i+n}$ and $A_{i,m_i^*} = g^{x_i}$. For $i = 1$ to n , let $B_{i,\text{id}_i^*} = g^{c_i}$ and $B_{i,\text{id}_i^*} = g^{y_i}$. We remark that the parameters are distributed independently and uniformly at random as in the real scheme.

Queries. Conceptually, the simulator will be able to create keys or signatures for the adversary, because his requests will differ from the challenge identity or message in at least one bit. More specifically,

1. Create New Key: The simulator begins with an index $i = 1$ and an empty sequence of index/identity/private key triples T . On input an identity $\mathcal{I} \in \{0, 1\}^n$, if $\mathcal{I} = \mathcal{I}^*$, the simulator records $(i, \mathcal{I}^*, \perp)$ in T . Otherwise, the simulator computes the secret key as follows. Let β be the first index such that $\text{id}_i \neq \text{id}_i^*$. Use $n-2$ pairings on the B_{i,id_i} values to compute $s = (g_{n-1})^{\prod_{i=1, \dots, n \wedge i \neq \beta} b_{i,\text{id}_i}}$. Then compute $\text{SK}_{\mathcal{I}} = s^{y_\beta} = (g_{n-1})^{\prod_{i=1, \dots, n} b_{i,\text{id}_i}}$. Record $(i, \mathcal{I}, \text{SK}_{\mathcal{I}})$ in T . Secret keys are unique and perfectly distributed as in the real game.
2. Corrupt User: On input an index $i \in [1, |T|]$, the simulator returns to the adversary the triple $(i, \mathcal{I}_i, \text{SK}_{\mathcal{I}_i}) \in T$. It returns an error if T is empty or i is out of range. Recall that i cannot be associated with $\mathcal{I}^*$ in this game.
3. Sign: On input an index $i \in [1, |T|]$ and a message $M \in \{0, 1\}^\ell$, the simulator obtains the triple $(i, \mathcal{I}_i, \text{SK}_{\mathcal{I}_i}) \in T$ or returns an error if it does not exist. If $\mathcal{I}_i \neq \mathcal{I}^*$, then the simulator signs M with $\text{SK}_{\mathcal{I}_i}$ in the usual way.

If $\mathcal{I}_i = \mathcal{I}^*$, then we know $M \neq M^*$. Let β be the first index such that $m_\beta \neq m_\beta^*$. Use $\ell-2$ pairings on the A_{i,m_i} values to compute $\sigma' = (g_{\ell-1})^{\prod_{i=1, \dots, \ell \wedge i \neq \beta} a_{i,m_i}}$. Next, compute $\sigma'' = \sigma'^{x_i} = (g_{\ell-1})^{\prod_{i=1, \dots, \ell} a_{i,m_i}}$. Use $n-1$ pairings on the B_{i,id_i} values to compute $\gamma = (g_n)^{\prod_{i=1, \dots, n} b_{i,\text{id}_i}}$. Finally, compute $\sigma = e(\gamma, \sigma'') = (g_{k-1})^{(\prod_{i \in [1, n]} b_{i,\text{id}_i})(\prod_{i \in [1, \ell]} a_{i,m_i})}$. Return σ to $\mathcal{A}$. Signatures are unique and perfectly distributed as in the real game.

Response. Eventually, $\mathcal{A}$ outputs an aggregate signature σ^* on multiset S^* where $(\mathcal{I}^*, M^*) \in S^*$. The simulator will extract from this a solution to the MCDH problem. This works by iteratively computing all the other signatures in S^* and then dividing them out of the aggregate until only one or more signatures on $(\mathcal{I}^*, M^*)$ remain. That is, the simulator takes an aggregate for S^* and computes an aggregate signature for S' where S' has one less verification key/message pair than S at each step. These signatures will be computed as in the query phase.

Eventually, we have an aggregate σ' on $t \geq 1$ instances of $(\mathcal{I}^*, M^*)$. We have that $e(\sigma', g) = H(\mathcal{I}^*, M^*)^t = (g_k)^{t(\prod_{i \in [1, n]} b_{i,\text{id}_i^*})(\prod_{i \in [1, \ell]} a_{i,m_i^*})} = (g_k)^{t \prod_{i=1}^k c_i}$ and thus $\sigma' = (g_{k-1})^{t \prod_{i=1}^k c_i}$. The simulator computes $\sigma'^{1/t}$ (recall that t is not 0 mod p) which gives $(g_{k-1})^{\prod_{i=1}^k c_i}$ and this is given as the solution to the MCDH problem.

As remarked in the Setup and Query phase, the responses of the challenger are distributed identically to the real unforgeability game. The simulator succeeds whenever $\mathcal{A}$ does. $\square$

Zerocoin: Anonymous Distributed E-Cash from Bitcoin

Ian Miers, Christina Garman, Matthew Green, Aviel D. Rubin
The Johns Hopkins University Department of Computer Science, Baltimore, USA
{imiers, cgarman, mgreen, rubin}@cs.jhu.edu

Abstract—Bitcoin is the first e-cash system to see widespread adoption. While Bitcoin offers the potential for new types of financial interaction, it has significant limitations regarding privacy. Specifically, because the Bitcoin transaction log is completely public, users’ privacy is protected only through the use of pseudonyms. In this paper we propose Zerocoin, a cryptographic extension to Bitcoin that augments the protocol to allow for fully anonymous currency transactions. Our system uses standard cryptographic assumptions and does not introduce new trusted parties or otherwise change the security model of Bitcoin. We detail Zerocoin’s cryptographic construction, its integration into Bitcoin, and examine its performance both in terms of computation and impact on the Bitcoin protocol.

I. INTRODUCTION

Digital currencies have a long academic pedigree. As of yet, however, no system from the academic literature has seen widespread use. Bitcoin, on the other hand, is a viable digital currency with a market capitalization valued at more than \$100 million [1] and between \$2 and \$5 million USD in transactions a day [2]. Unlike many proposed digital currencies, Bitcoin is fully decentralized and requires no central bank or authority. Instead, its security depends on a distributed architecture and two assumptions: that a majority of its nodes are honest and that a substantive proof-of-work can deter Sybil attacks. As a consequence, Bitcoin requires neither legal mechanisms to detect and punish double spending nor trusted parties to be chosen, monitored, or policed. This decentralized design is likely responsible for Bitcoin’s success, but it comes at a price: all transactions are public and conducted between cryptographically binding pseudonyms.

While relatively few academic works have considered the privacy implications of Bitcoin’s design [2, 3], the preliminary results are not encouraging. In one example, researchers were able to trace the spending of 25,000 bitcoins that were allegedly stolen in 2011 [3, 4]. Although tracking stolen coins may seem harmless, we note that similar techniques could also be applied to trace sensitive transactions, thus violating users’ privacy. Moreover, there is reason to believe that sophisticated results from other domains (e.g., efforts to de-anonymize social network data using network topology [5]) will soon be applied to the Bitcoin transaction graph.

Since all Bitcoin transactions are public, anonymous transactions are necessary to avoid tracking by third parties even if we do not wish to provide the absolute anonymity

typically associated with e-cash schemes. On top of such transactions, one could build mechanisms to partially or explicitly identify participants to authorized parties (e.g., law enforcement). However, to limit this information to authorized parties, we must first anonymize the underlying public transactions.

The Bitcoin community generally acknowledges the privacy weaknesses of the currency. Unfortunately, the available mitigations are quite limited. The most common recommendation is to employ a *laundry service* which exchanges different users’ bitcoins. Several of these are in commercial operation today [6, 7]. These services, however, have severe limitations: operators can steal funds, track coins, or simply go out of business, taking users’ funds with them. Perhaps in recognition of these risks, many services offer short laundering periods, which lead to minimal transaction volumes and hence to limited anonymity.

Our contribution. In this paper we describe Zerocoin, a distributed e-cash system that uses cryptographic techniques to break the link between individual Bitcoin transactions *without* adding trusted parties. To do this, we first define the abstract functionality and security requirements of a new primitive that we call a *decentralized e-cash* scheme. We next propose a concrete instantiation and prove it secure under standard cryptographic assumptions. Finally, we describe the specific extensions required to integrate our protocol into the Bitcoin system and evaluate the performance of a prototype implementation derived from the original open-source `bitcoind` client.

We are not the first to propose e-cash techniques for solving Bitcoin’s privacy problems. However, a common problem with many e-cash protocols is that they rely fundamentally on a trusted currency issuer or “bank,” who creates electronic “coins” using a blind signature scheme. One solution (attempted unsuccessfully with Bitcoin [8]) is to simply appoint such a party. Alternatively, one can distribute the responsibility among a quorum of nodes using threshold cryptography. Unfortunately, both of these solutions introduce points of failure and seem inconsistent with the Bitcoin network model, which consists of many untrusted nodes that routinely enter and exit the network. Moreover, the problem of choosing long-term trusted parties, especially in the legal and regulatory grey area Bitcoin operates in, seems like a major impediment to adoption. Zerocoin eliminates

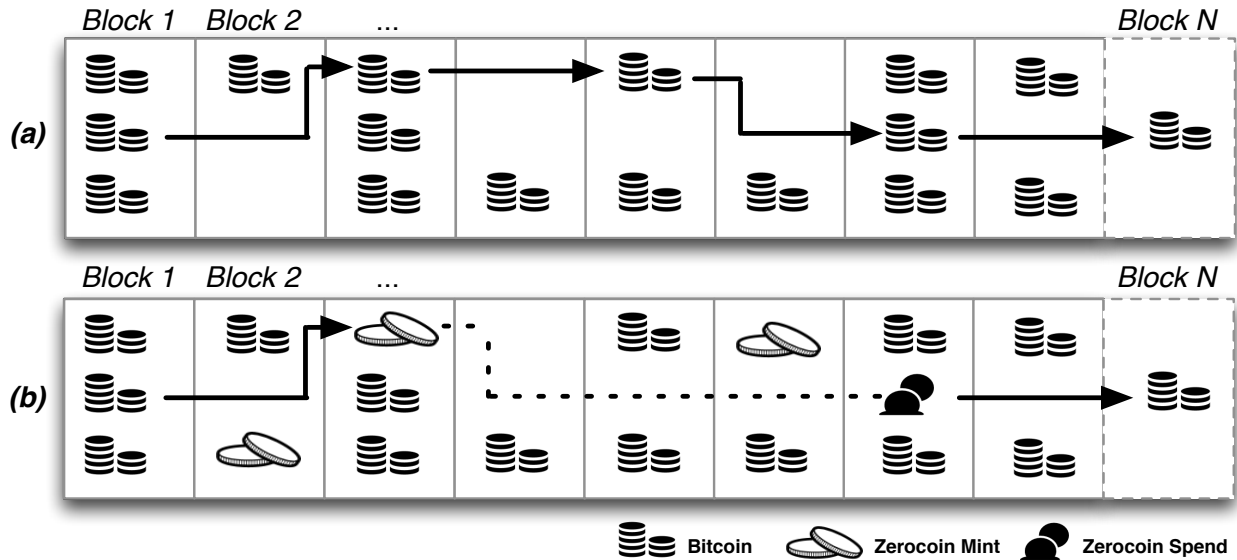


Figure 1: Two example block chains. Chain (a) illustrates a normal Bitcoin transaction history, with each transaction linked to a preceding transaction. Chain (b) illustrates a Zerocoin chain. The linkage between mint and spend (dotted line) cannot be determined from the block chain data.

the need for such coin issuers by allowing individual Bitcoin clients to generate their own coins — provided that they have sufficient classical bitcoins to do so.

Intuition behind our construction. To understand the intuition behind Zerocoin, consider the following “pencil and paper” protocol example. Imagine that all users share access to a physical bulletin board. To mint a zerocoin of fixed denomination \$1, a user Alice first generates a random coin serial number S , then commits to S using a secure digital commitment scheme. The resulting commitment is a coin, denoted C , which can only be opened by a random number r to reveal the serial number S . Alice pins C to the public bulletin board, along with \$1 of physical currency. All users will accept C provided it is correctly structured and carries the correct sum of currency.

To redeem her coin C , Alice first scans the bulletin board to obtain the set of valid commitments $(C_1, \dots, C_N)$ that have thus far been posted by *all* users in the system. She next produces a non-interactive zero-knowledge proof π for the following two statements: (1) she knows a $C \in (C_1, \dots, C_N)$ and (2) she knows a hidden value r such that the commitment C opens to S . In full view of the others, Alice, using a disguise to hide her identity,¹ posts a “spend” transaction containing (S, π) . The remaining users verify the proof π and check that S has not previously appeared in any other spend transaction. If these conditions are met, the users allow

Alice to collect \$1 from *any* location on the bulletin board; otherwise they reject her transaction and prevent her from collecting the currency.

This simple protocol achieves some important aims. First, Alice’s minted coin cannot be linked to her retrieved funds: in order to link the coin C to the serial number S used in her withdrawal, one must either know r or directly know which coin Alice proved knowledge of, neither of which are revealed by the proof. Thus, even if the original dollar bill is recognizably tainted (e.g., it was used in a controversial transaction), it cannot be linked to Alice’s new dollar bill. At the same time, if the commitment and zero-knowledge proof are secure, then Alice cannot double-spend any coin without re-using the serial number S and thus being detected by the network participants.

Of course, the above protocol is not workable: bulletin boards are a poor place to store money and critical information. Currency might be stolen or serial numbers removed to allow double spends. More importantly, to conduct this protocol over a network, Alice requires a distributed digital backing currency.²

The first and most basic contribution of our work is to recognize that Bitcoin answers all of these concerns, providing us with a backing currency, a bulletin board, and a conditional currency redemption mechanism. Indeed, the core of the Bitcoin protocol is the decentralized calculation

¹Of course, in the real protocol Alice will emulate this by using an anonymity network such as Tor [9].

²One could easily imagine a solution based on existing payment networks, e.g., Visa or Paypal. However, this would introduce the need for trusted parties or exchanges.

of a *block chain* which acts as a trusted, append-only bulletin board that can both store information and process financial transactions. Alice can add her commitments *and* escrow funds by placing them in the block chain while being assured that strict protocol conditions (and not her colleagues' scruples) determine when her committed funds may be accessed.

Of course, even when integrated with the Bitcoin block chain, the protocol above has another practical challenge. Specifically, it is difficult to *efficiently* prove that a commitment C is in the set $(C_1, \dots, C_N)$. The naive solution is to prove the disjunction $(C = C_1) \vee (C = C_2) \vee \dots \vee (C = C_N)$. Unfortunately such "OR proofs" have size $O(N)$, which renders them impractical for all but small values of N .

Our second contribution is to solve this problem, producing a new construction with proofs that do not grow linearly as N increases. Rather than specifying an expensive OR proof, we employ a "public" one-way accumulator to reduce the size of this proof. One-way accumulators [10, 11, 12, 13, 14], first proposed by Benaloh and de Mare [10], allow parties to combine many elements into a constant-sized data structure, while efficiently *proving* that one specific value is contained within the set. In our construction, the Bitcoin network computes an accumulator A over the commitments $(C_1, \dots, C_N)$, along with the appropriate membership witnesses for each item in the set. The spender need only prove knowledge of one such witness. In practice, this can reduce the cost of the spender's proof to $O(\log N)$ or even constant size.

Our application requires specific properties from the accumulator. With no trusted parties, the accumulator and its associated witnesses must be *publicly* computable and verifiable (though we are willing to relax this requirement to include a single, trusted *setup* phase in which parameters are generated). Moreover, the accumulator must bind even the computing party to the values in the set. Lastly, the accumulator must support an efficient non-interactive witness-indistinguishable or zero-knowledge proof of set membership. Fortunately such accumulators do exist. In our concrete proposal of Section IV we use a construction based on the Strong RSA accumulator of Camenisch and Lysyanskaya [12], which is in turn based on an accumulator of Baric and Pfizmann [11] and Benaloh and de Mare [10].

Outline of this work. The rest of this paper proceeds as follows. In Section II we provide a brief technical overview of the Bitcoin protocol. In Section III we formally define the notion of *decentralized e-cash* and provide correctness and security requirements for such a system. In Section IV we give a concrete realization of our scheme based on standard cryptographic hardness assumptions including the Discrete Logarithm problem and Strong RSA. Finally, in Sections V, VI, and VII, we describe how we integrate our e-cash construction into the Bitcoin protocol, discuss the

security and anonymity provided, and detail experimental results showing that our solution is practical.

II. OVERVIEW OF BITCOIN

In this section we provide a short overview of the Bitcoin protocol. For a more detailed explanation, we refer the reader to the original specification of Nakamoto [15] or to the summary of Barber *et al.* [2].

The Bitcoin network. Bitcoin is a peer-to-peer network of nodes that distribute and record transactions, and clients used to interact with the network. The heart of Bitcoin is the *block chain*, which serves as an append-only bulletin board maintained in a distributed fashion by the Bitcoin peers. The block chain consists of a series of blocks connected in a hash chain.³ Every Bitcoin block memorializes a set of transactions that are collected from the Bitcoin broadcast network.

Bitcoin peers compete to determine which node will generate the next canonical block. This competition requires each node to solve a proof of work based on identifying specific SHA-256 preimages, specifically a block B such that $\text{SHA256}(\text{SHA256}(B)) = (0^\ell || \{0, 1\}^{256-\ell})$.⁴ The value ℓ is selected by a periodic network vote to ensure that on average a block is created every 10 minutes. When a peer generates a valid solution, a process known as *mining*, it broadcasts the new block to all nodes in the system. If the block is valid (i.e., all transactions validate and a valid proof of work links the block to the chain thus far), then the new block is accepted as the head of the block chain. The process then repeats.

Bitcoin provides two separate incentives to peers that mine new blocks. First, successfully mining a new block (which requires a non-trivial computational investment) entitles the creator to a reward, currently set at 25 BTC.⁵ Second, nodes who mine blocks are entitled to collect *transaction fees* from every transaction they include. The fee paid by a given transaction is determined by its author (though miners may exclude transactions with insufficient fees or prioritize high fee transactions).

Bitcoin transactions. A Bitcoin transaction consists of a set of outputs and inputs. Each output is described by the tuple (a, V) where a is the amount, denominated in Satoshi (one bitcoin = 10^9 Satoshi), and V is a specification of who is authorized to spend that output. This specification, denoted *scriptPubKey*, is given in *Bitcoin script*, a stack-based non-Turing-complete language similar to Forth. Transaction inputs

³For efficiency reasons, this chain is actually constructed using a hash tree, but we use the simpler description for this overview.

⁴Each block includes a counter value that may be incremented until the hash satisfies these requirements.

⁵The Bitcoin specification holds that this reward should be reduced every few years, eventually being eliminated altogether.

```

Input:
Previous tx: 030b5937d9f4aa1a3133b...
Index: 0
scriptSig: 0dcd253cdf8ea11cdc710e5e92af7647...

Output:
Value: 5000000000
scriptPubKey: OP_DUP OP_HASH160
a45f2757f94fd2337ebf7ddd018c11a21fb6c283
OP_EQUALVERIFY OP_CHECKSIG

```

Figure 2: Example Bitcoin transaction. The output script specifies that the redeeming party provide a public key that hashes to the given value and that the transaction be signed with the corresponding private key.

are simply a reference to a previous transaction output,⁶ as well as a second script, `scriptSig`, with code and data that when combined with `scriptPubKey` evaluates to true. Coinbase transactions, which start off every block and pay its creator, do not include a transaction input.

To send d bitcoins to Bob, Alice embeds the hash⁷ of Bob's ECDSA public key pk_b , the amount d , and some script instructions in `scriptPubKey` as one output of a transaction whose referenced inputs total at least d bitcoins (see Figure 2). Since any excess input is paid as a transaction fee to the node who includes it in a block, Alice typically adds a second output paying the surplus change back to herself. Once the transaction is broadcasted to the network and included in a block, the bitcoins belong to Bob. However, Bob should only consider the coins his once at least five subsequent blocks reference this block.⁸ Bob can spend these coins in a transaction by referencing it as an input and including in `scriptSig` a signature on the claiming transaction under sk_b and the public key pk_b .

Anonymity. Anonymity was not one of the design goals of Bitcoin [3, 15, 17]. Bitcoin provides only *pseudonymity* through the use of Bitcoin identities (public keys or their hashes), of which a Bitcoin user can generate an unlimited number. Indeed, many Bitcoin clients routinely generate new identities in an effort to preserve the user's privacy.

Regardless of Bitcoin design goals, Bitcoin's user base seems willing to go through considerable effort to maintain their anonymity — including risking their money and paying transaction fees. One illustration of this is the existence of laundries that (for a fee) will mix together different users' funds in the hopes that shuffling makes them difficult to trace [2, 6, 7]. Because such systems require the users to trust the laundry to both (a) not record how the mixing is done

and (b) give the users back the money they put in to the pot, use of these systems involves a fair amount of risk.

III. DECENTRALIZED E-CASH

Our approach to anonymizing the Bitcoin network uses a form of cryptographic e-cash. Since our construction does not require a central coin issuer, we refer to it as a *decentralized e-cash scheme*. In this section we define the algorithms that make up a decentralized e-cash scheme and describe the correctness and security properties required of such a system.

Notation. Let λ represent an adjustable security parameter, let $\text{poly}(\cdot)$ represent some polynomial function, and let $\nu(\cdot)$ represent a negligible function. We use $\mathcal{C}$ to indicate the set of allowable coin values.

Definition 3.1 (Decentralized E-Cash Scheme): A decentralized e-cash scheme consists of a tuple of possibly randomized algorithms (Setup, Mint, Spend, Verify).

- $\text{Setup}(1^\lambda) \rightarrow \text{params}$. On input a security parameter, output a set of global public parameters params and a description of the set $\mathcal{C}$.
- $\text{Mint}(\text{params}) \rightarrow (c, \text{skc})$. On input parameters params , output a coin $c \in \mathcal{C}$, as well as a trapdoor skc .
- $\text{Spend}(\text{params}, c, \text{skc}, R, \mathbf{C}) \rightarrow (\pi, S)$. Given params , a coin c , its trapdoor skc , some transaction string $R \in \{0, 1\}^*$, and an arbitrary set of coins $\mathbf{C}$, output a coin spend transaction consisting of a proof π and serial number S if $c \in \mathbf{C} \subseteq \mathcal{C}$. Otherwise output $\perp$.
- $\text{Verify}(\text{params}, \pi, S, R, \mathbf{C}) \rightarrow \{0, 1\}$. Given params , a proof π , a serial number S , transaction information R , and a set of coins $\mathbf{C}$, output 1 if $\mathbf{C} \subseteq \mathcal{C}$ and (π, S, R) is valid. Otherwise output 0.

We note that the Setup routine may be executed by a trusted party. Since this setup occurs only once and does not produce any corresponding secret values, we believe that this relaxation is acceptable for real-world applications. Some concrete instantiations may use different assumptions.

Each coin is generated using a randomized minting algorithm. The *serial number* S is a unique value released during the spending of a coin and is designed to prevent any user from spending the same coin twice. We will now formalize the correctness and security properties of a decentralized e-cash scheme. Each call to the Spend algorithm can include an arbitrary string R , which is intended to store transaction-specific information (e.g., the identity of a transaction recipient).

Correctness. Every decentralized e-cash scheme must satisfy the following correctness requirement. Let $\text{params} \leftarrow \text{Setup}(1^\lambda)$ and $(c, \text{skc}) \leftarrow \text{Mint}(\text{params})$. Let $\mathbf{C} \subseteq \mathcal{C}$ be any valid set of coins, where $|\mathbf{C}| \leq \text{poly}(\lambda)$, and

⁶This reference consists of a transaction hash identifier as well as an index into the transaction's output list.

⁷A 34 character hash that contains the double SHA-256 hash of the key and some checksum data.

⁸Individual recipients are free to disregard this advice. However, this could make them vulnerable to double-spending attacks as described by Karame *et al.* [16].

assign $(\pi, S) \leftarrow \text{Spend}(params, c, skc, R, \mathbf{C})$. The scheme is *correct* if, over all $\mathbf{C}$, R , and random coins used in the above algorithms, the following equality holds with probability $1 - \nu(\lambda)$:

$$\text{Verify}(params, \pi, S, R, \mathbf{C} \cup \{c\}) = 1$$

Security. The security of a decentralized e-cash system is defined by the following two games: Anonymity and Balance. We first describe the Anonymity experiment, which ensures that the adversary cannot link a given coin spend transaction (π, S) to the coin associated with it, even when the attacker provides many of the coins used in generating the spend transaction.

Definition 3.2 (Anonymity): A decentralized e-cash scheme $\Pi = (\text{Setup}, \text{Mint}, \text{Spend}, \text{Verify})$ satisfies the *Anonymity* requirement if every *probabilistic polynomial-time* (*p.p.t.*) adversary $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ has negligible advantage in the following experiment.

Anonymity($\Pi, \mathcal{A}, \lambda$)
 $params \leftarrow \text{Setup}(1^\lambda)$
 For $i \in \{0, 1\}$: $(c_i, skc_i) \leftarrow \text{Mint}(params)$
 $(\mathbf{C}, R, z) \leftarrow \mathcal{A}_1(params, c_0, c_1)$; $b \leftarrow \{0, 1\}$
 $(\pi, S) \leftarrow \text{Spend}(params, c_b, skc_b, R, \mathbf{C} \cup \{c_0, c_1\})$
 Output: $b' \leftarrow \mathcal{A}_2(z, \pi, S)$

We define $\mathcal{A}$'s advantage in the above game as $|\Pr[b = b'] - 1/2|$.

The Balance property requires more consideration. Intuitively, we wish to ensure that an attacker cannot *spend* more coins than she mints, even when she has access to coins and spend transactions produced by honest parties. Note that to strengthen our definition, we also capture the property that an attacker might alter valid coins, e.g., by modifying their transaction information string R .

Our definition is reminiscent of the “one-more forgery” definition commonly used for blind signatures. We provide the attacker with a collection of valid coins and an oracle $\mathcal{O}_{\text{spend}}$ that she may use to spend any of them.⁹ Ultimately $\mathcal{A}$ must produce m coins and $m + 1$ valid spend transactions such that no transaction duplicates a serial number or modifies a transaction produced by the honest oracle.

Definition 3.3 (Balance): A decentralized e-cash scheme $\Pi = (\text{Setup}, \text{Mint}, \text{Spend}, \text{Verify})$ satisfies the *Balance* property if $\forall N \leq \text{poly}(\lambda)$ every *p.p.t.* adversary $\mathcal{A}$ has negligible advantage in the following experiment.

Balance($\Pi, \mathcal{A}, N, \lambda$)
 $params \leftarrow \text{Setup}(1^\lambda)$
 For $i = 1$ to N : $(c_i, skc_i) \leftarrow \text{Mint}(params)$
 Output: $(c'_1, \dots, c'_m, S_1, \dots, S_m, S_{m+1})$
 $\leftarrow \mathcal{A}^{\mathcal{O}_{\text{spend}}(\cdot, \cdot)}(params, c_1, \dots, c_N)$

⁹We provide this functionality as an oracle to capture the possibility that the attacker can specify *arbitrary* input for the value $\mathbf{C}$.

The oracle $\mathcal{O}_{\text{spend}}$ operates as follows: on the j th query $\mathcal{O}_{\text{spend}}(c_j, \mathbf{C}_j, R_j)$, the oracle outputs $\perp$ if $c_j \notin \{c_1, \dots, c_N\}$. Otherwise it returns $(\pi_j, S_j) \leftarrow \text{Spend}(params, c_j, skc_j, R_j, \mathbf{C}_j)$ to $\mathcal{A}$ and records (S_j, R_j) in the set $\mathcal{T}$.

We say that $\mathcal{A}$ wins (i.e., she produces more spends than minted coins) if $\forall \mathbf{s} \in \{S_1, \dots, S_m, S_{m+1}\}$ where $\mathbf{s} = (\pi', S', R', \mathbf{C}')$:

- $\text{Verify}(params, \pi', S', R', \mathbf{C}') = 1$.
- $\mathbf{C}' \subseteq \{c_1, \dots, c_N, c'_1, \dots, c'_m\}$.
- $(S', R') \notin \mathcal{T}$.
- S' appears in only one tuple from $\{S_1, \dots, S_m, S_{m+1}\}$.

We define $\mathcal{A}$'s advantage as the probability that $\mathcal{A}$ wins the above game.

IV. DECENTRALIZED E-CASH FROM STRONG RSA

In this section we describe a *concrete* instantiation of a decentralized e-cash scheme. We first define the necessary cryptographic ingredients.

A. Cryptographic Building Blocks

Zero-knowledge proofs and signatures of knowledge. Our protocols use zero-knowledge proofs that can be instantiated using the technique of Schnorr [18], with extensions due to e.g., [19, 20, 21, 22]. We convert these into *non-interactive* proofs by applying the Fiat-Shamir heuristic [23]. In the latter case, we refer to the resulting non-interactive proofs as *signatures of knowledge* as defined in [24].

When referring to these proofs we will use the notation of Camenisch and Stadler [25]. For instance, $\text{NIZKPoK}\{(x, y) : h = g^x \wedge c = g^y\}$ denotes a non-interactive zero-knowledge proof of knowledge of the elements x and y that satisfy both $h = g^x$ and $c = g^y$. All values not enclosed in $()$'s are assumed to be known to the verifier. Similarly, the extension $\text{ZKSoK}[m]\{(x, y) : h = g^x \wedge c = g^y\}$ indicates a *signature of knowledge* on message m .

Accumulators. Our construction uses an accumulator based on the Strong RSA assumption. The accumulator we use was first proposed by Benaloh and de Mare [10] and later improved by Baric and Pfitzmann [11] and Camenisch and Lysyanskaya [12]. We describe the accumulator using the following algorithms:

- $\text{AccumSetup}(\lambda) \rightarrow params$. On input a security parameter, sample primes p, q (with polynomial dependence on the security parameter), compute $N = pq$, and sample a seed value $u \in \mathbb{Z}_N, u \neq 1$. Output (N, u) as $params$.
- $\text{Accumulate}(params, \mathbf{C}) \rightarrow A$. On input $params$ (N, u) and a set of prime numbers $\mathbf{C} = \{c_1, \dots, c_i \mid c \in [\mathcal{A}, \mathcal{B}]\}$,¹⁰ compute the accumulator A as $u^{c_1 c_2 \dots c_n} \bmod N$.

¹⁰See Appendix A for a more precise description.

- $\text{GenWitness}(params, v, \mathbf{C}) \rightarrow w$. On input $params$ (N, u), a set of prime numbers $\mathbf{C}$ as described above, and a value $v \in \mathbf{C}$, the witness w is the accumulation of all the values in $\mathbf{C}$ besides v , i.e., $w = \text{Accumulate}(params, \mathbf{C} \setminus \{v\})$.
- $\text{AccVerify}(params, A, v, \omega) \rightarrow \{0, 1\}$. On input $params$ (N, u), an element v , and witness ω , compute $A' \equiv \omega^v \pmod N$ and output 1 if and only if $A' = A$, v is prime, and $v \in [\mathcal{A}, \mathcal{B}]$ as defined previously.

For simplicity, the description above uses the full calculation of A . Camenisch and Lysyanskaya [12] observe that the accumulator may also be *incrementally* updated, i.e., given an existing accumulator A_n it is possible to add an element x and produce a new accumulator value A_{n+1} by computing $A_{n+1} = A_n^x \pmod N$. We make extensive use of this optimization in our practical implementation.

Camenisch and Lysyanskaya [12] show that the accumulator satisfies a strong *collision-resistance* property if the Strong RSA assumption is hard. Informally, this ensures that no *p.p.t.* adversary can produce a pair (v, ω) such that $v \notin \mathbf{C}$ and yet AccVerify is satisfied. Additionally, they describe an efficient zero-knowledge proof of knowledge that a committed value is in an accumulator. We convert this into a non-interactive proof using the Fiat-Shamir transform and refer to the resulting proof using the following notation:

$$\text{NIZKPoK}\{(v, \omega) : \text{AccVerify}((N, u), A, v, \omega) = 1\}.$$

B. Our Construction

We now describe a concrete decentralized e-cash scheme. Our scheme is secure assuming the hardness of the Strong RSA and Discrete Logarithm assumptions, and the existence of a zero-knowledge proof system.

We now describe the algorithms:

- $\text{Setup}(1^\lambda) \rightarrow params$. On input a security parameter, run $\text{AccumSetup}(1^\lambda)$ to obtain the values (N, u) . Next generate primes p, q such that $p = 2^w q + 1$ for $w \geq 1$. Select random generators g, h such that $\mathbb{G} = \langle g \rangle = \langle h \rangle$ and $\mathbb{G}$ is a subgroup of $\mathbb{Z}_q^*$. Output $params = (N, u, p, q, g, h)$.
- $\text{Mint}(params) \rightarrow (c, skc)$. Select $S, r \leftarrow \mathbb{Z}_q^*$ and compute $c \leftarrow g^S h^r \pmod p$ such that $\{c \text{ prime} \mid c \in [\mathcal{A}, \mathcal{B}]\}$.¹¹ Set $skc = (S, r)$ and output (c, skc) .
- $\text{Spend}(params, c, skc, R, \mathbf{C}) \rightarrow (\pi, S)$. If $c \notin \mathbf{C}$ output $\perp$. Compute $A \leftarrow \text{Accumulate}((N, u), \mathbf{C})$ and $\omega \leftarrow \text{GenWitness}((N, u), c, \mathbf{C})$. Output (π, S) where π comprises the following signature of knowledge:¹²

$$\pi = \text{ZKSoK}[R]\{(c, w, r) : \\ \text{AccVerify}((N, u), A, c, w) = 1 \wedge c = g^S h^r\}$$
- $\text{Verify}(params, \pi, S, R, \mathbf{C}) \rightarrow \{0, 1\}$. Given a proof π , a serial number S , and a set of coins $\mathbf{C}$, first compute

$A \leftarrow \text{Accumulate}((N, u), \mathbf{C})$. Next verify that π is the aforementioned signature of knowledge on R using the known public values. If the proof verifies successfully, output 1, otherwise output 0.

Our protocol assumes a trusted setup process for generating the parameters. We stress that the accumulator trapdoor (p, q) is not used subsequent to the Setup procedure and can therefore be destroyed immediately after the parameters are generated. Alternatively, implementers can use the technique of Sander for generating so-called RSA UFOs for accumulator parameters without a trapdoor [26].

C. Security Analysis

We now consider the security of our construction.

Theorem 4.1: If the zero-knowledge signature of knowledge is computationally zero-knowledge in the random oracle model, then $\Pi = (\text{Setup}, \text{Mint}, \text{Spend}, \text{Verify})$ satisfies the Anonymity property.

We provide a proof sketch for Theorem 4.1 in Appendix A. Intuitively, the security of our construction stems from the fact that the coin commitment C is a perfectly-hiding commitment and the signature proof π is at least computationally zero-knowledge. These two facts ensure that the adversary has at most negligible advantage in guessing which coin was spent.

Theorem 4.2: If the signature proof π is *sound* in the random oracle model, the Strong RSA problem is hard, and the Discrete Logarithm problem is hard in $\mathbb{G}$, then $\Pi = (\text{Setup}, \text{Mint}, \text{Spend}, \text{Verify})$ satisfies the Balance property.

A proof of Theorem 4.1 is included in Appendix A. Briefly, this proof relies on the binding properties of the coin commitment, as well as the soundness and unforgeability of the ZKSoK and collision-resistance of the accumulator. We show that an adversary who wins the Balance game with non-negligible advantage can be used to *either* find a collision in the commitment scheme (allowing us to solve the Discrete Logarithm problem) *or* find a collision in the accumulator (which leads to a solution for Strong RSA).

V. INTEGRATING WITH BITCOIN

While the construction of the previous section gives an overview of our approach, we have yet to describe how our techniques integrate with Bitcoin. In this section we address the specific challenges that come up when we combine a decentralized e-cash scheme with the Bitcoin protocol.

The general overview of our approach is straightforward. To mint a zerocoin c of denomination d , Alice runs $\text{Mint}(params) \rightarrow (c, skc)$ and stores skc securely.¹³ She then embeds c in the output of a Bitcoin transaction that spends $d + \text{fees}$ classical bitcoins. Once a mint transaction has been accepted into the block chain, c is included in the

¹¹See Appendix A for a more precise description.

¹²See Appendix B for the construction of the ZKSoK.

¹³In our implementation all bitcoins have a single fixed value. However, we can support multiple values by running distinct Zerocoin instantiations simultaneously, all sharing the same set of public parameters.

global accumulator A , and the currency cannot be accessed except through a Zerocoin spend, i.e., it is essentially placed into escrow.

To spend c with Bob, Alice first constructs a partial transaction ptx that references an unclaimed mint transaction as input and includes Bob's public key as output. She then traverses all valid mint transactions in the block chain, assembles the set of minted coins C , and runs $\text{Spend}(params, c, skc, hash(ptx), C) \rightarrow (\pi, S)$. Finally, she completes the transaction by embedding (π, S) in the `scriptSig` of the input of ptx . The output of this transaction could also be a further Zerocoin mint transaction — a feature that may be useful to transfer value between multiple Zerocoin instances (i.e., of different denomination) running in the same block chain.

When this transaction appears on the network, nodes check that $\text{Verify}(params, \pi, S, hash(ptx), C) = 1$ and check that S does not appear in any previous transaction. If these conditions hold and the referenced mint transaction is not claimed as an input into a different transaction, the network accepts the spend as valid and allows Alice to redeem d bitcoins.

Computing the accumulator. A naive implementation of the construction in Section IV requires that the verifier recompute the accumulator A with each call to $\text{Verify}(\dots)$. In practice, the cost can be substantially reduced.

First, recall that the accumulator in our construction can be computed *incrementally*, hence nodes can add new coins to the accumulation when they arrive. To exploit this, we require any node mining a new block to add the zerocoins in that block to the previous block's accumulator and store the resulting new accumulator value in the *coinbase transaction* at the start of the new block.¹⁴ We call this an *accumulator checkpoint*. Peer nodes validate this computation before accepting the new block into the blockchain. Provided that this verification occurs routinely when blocks are added to the chain, some clients may choose to trust the accumulator in older (confirmed) blocks rather than re-compute it from scratch.

With this optimization, Alice need no longer compute the accumulator A and the full witness w for c . Instead she can merely reference the current block's *accumulator checkpoint* and compute the witness starting from the checkpoint preceding her mint (instead of starting at T_0), since computing the witness is equivalent to accumulating $C \setminus \{c\}$.

New transaction types. Bitcoin transactions use a flexible scripting language to determine the validity of each transaction. Unfortunately, Bitcoin script is (by design) *not* Turing-complete. Moreover, large segments of the already-limited

script functionality have been disabled in the Bitcoin production network due to security concerns. Hence, the existing script language cannot be used for sophisticated calculations such as verifying zero-knowledge proofs. Fortunately for our purposes, the Bitcoin designers chose to reserve several script operations for future expansion.

We extend Bitcoin by adding a new instruction: `ZEROCOIN_MINT`. Minting a zerocoin constructs a transaction with an output whose `scriptPubKey` contains this instruction and a coin c . Nodes who receive this transaction should validate that c is a well-formed coin. To spend a zerocoin, Alice constructs a new transaction that claims as input some Zerocoin mint transaction and has a `scriptSig` field containing (π, S) and a reference to the block containing the accumulator used in π . A verifier extracts the accumulator from the referenced block and, using it, validates the spend as described earlier.

Finally, we note that transactions must be signed to prevent an attacker from simply changing who the transaction is paid to. Normal Bitcoin transactions include an ECDSA signature by the key specified in the `scriptPubKey` of the referenced input. However, for a spend transaction on an *arbitrary* zerocoin, there is no ECDSA public key. Instead, we use the ZKSoK π to sign the transaction hash that normally would be signed using ECDSA.¹⁵

Statekeeping and side effects. Validating a zerocoin changes Bitcoin's semantics: currently, Bitcoin's persistent state is defined solely in terms of transactions and blocks of transactions. Furthermore, access to this state is done via explicit reference by hash. Zerocoin, on the other hand, because of its strong anonymity requirement, deals with existentials: the coin is in the set of thus-far-minted coins and its serial number is *not* yet in the set of spent serial numbers. To enable these type of qualifiers, we introduce side effects into Bitcoin transaction handling. Processing a mint transaction causes a coin to be accumulated as a side effect. Processing a spend transaction causes the coin serial number to be added to a list of spent serial numbers held by the client.

For coin serial numbers, we have little choice but to keep a full list of them per client and incur the (small) overhead of storing that list and the larger engineering overhead of handling all possible ways a transaction can enter a client. The accumulator state is maintained within the accumulator checkpoints, which the client verifies for each received block.

Proof optimizations. For reasonable parameter sizes, the proofs produced by $\text{Spend}(\dots)$ exceed Bitcoin's 10KB transaction size limits. Although we can simply increase this limit, doing so has two drawbacks: (1) it drastically increases the storage requirements for Bitcoin since current transactions

¹⁴The coinbase transaction format already allows for the inclusion of arbitrary data, so this requires no fundamental changes to the Bitcoin protocol.

¹⁵In practice, this modification simply requires us to include the transaction digest in the hash computation of the challenge for the Fiat-Shamir proofs. See Appendix A for details.

are between 1 and 2 KB and (2) it may increase memory pressure on clients that store transactions in memory.¹⁶

In our prototype implementation we store our proofs in a separate, well-known location (a simple server). A full implementation could use a Distributed Hash Table or non block-chain backed storage in Bitcoin. While we recommend storing proofs in the block chain, these alternatives do not increase the storage required for the block chain.¹⁷

A. Suggestions for Optimizing Proof Verification

The complexity of the proofs will also lead to longer verification times than expected with a standard Bitcoin transaction. This is magnified by the fact that a Bitcoin transaction is verified once when it is included by a block and again by every node when that block is accepted into the block chain. Although the former cost can be accounted for by charging transaction fees, it would obviously be ideal for these costs to be as low as possible.

One approach is to distribute the cost of verification over the entire network and not make each node verify the entire proof. Because the ZKSoK we use utilizes cut-and-choose techniques, it essentially consists of n repeated iterations of the same proof (reducing the probability of forgery to roughly 2^{-n}). We can simply have nodes *randomly* select which iterations of the proofs they verify. By distributing this process across the network, we should achieve approximately the same security with less duplication of effort.

This optimization involves a time-space tradeoff, since the existing proof is verified by computing a series of (at a minimum) 1024 bit values $T_1, \dots, T_n$ and hashing the result. A naive implementation would require us to send $T_1, \dots, T_n$ fully computed — greatly increasing the size of the proof — since the client will only compute some of them but needs all of them to verify the hash. We can avoid this issue by replacing the standard hash with a Merkel tree where the leaves are the hashed T_i values and the root is the challenge hash used in the proof. We can then send the 160 bit or 256 bit intermediate nodes instead of the 1024 bit T_i values, allowing the verifier to compute *only* a subset of the T_i values and yet still validate the proof against the challenge without drastically increasing the proof size.

B. Limited Anonymity and Forward Security

A serious concern in the Bitcoin community is the loss of wallets due to poor endpoint security. In traditional Bitcoin, this results in the theft of coins [4]. However, in the Zerocoin setting it may also allow an attacker to *de-anonymize* Zerocoin transactions using the stored *skc*. The

obvious solution is to securely delete *skc* immediately after a coin is spent. Unfortunately, this provides no protection if *skc* is stolen at some earlier point.

One solution is to generate the spend transaction immediately (or shortly after) the coin is minted, possibly using an earlier checkpoint for calculating C . This greatly reduces the user's anonymity by decreasing the number of coins in C and leaking some information about when the coin was minted. However, no attacker who compromises the wallet can link any zerocoins in it to their mint transactions.

C. Code Changes

For our implementation, we chose to modify `bitcoind`, the original open-source Bitcoin C++ client. This required several modifications. First, we added instructions to the Bitcoin script for minting and spending zerocoins. Next, we added transaction types and code for handling these new instructions, as well as maintaining the list of spent serial numbers and the accumulator. We used the Charm cryptographic framework [27] to implement the cryptographic constructions in Python, and we used Boost's Python utilities to call that code from within `bitcoind`. This introduces some performance overhead, but it allowed us to rapidly prototype and leave room for implementing future constructions as well.

D. Incremental Deployment

As described above, Zerocoin requires changes to the Bitcoin protocol that must happen globally: while transactions containing the new instructions will be validated by updated servers, they will fail validation on older nodes, potentially causing the network to split when a block is produced that validates for some, but not all, nodes. Although this is not the first time Bitcoin has faced this problem, and there is precedent for a flag day type upgrade strategy [28], it is not clear how willing the Bitcoin community is to repeat it. As such, we consider the possibility of an incremental deployment.

One way to accomplish this is to embed the above protocol as comments in standard Bitcoin scripts. For non Zerocoin aware nodes, this data is effectively inert, and we can use Bitcoin's n of k signature support to specify that such comment embedded zerocoins are valid only if signed by some subset of the Zerocoin processing nodes. Such Zerocoin aware nodes can parse the comments and charge transaction fees for validation according to the proofs embedded in the comments, thus providing an incentive for more nodes to provide such services. Since this only changes the validation mechanism for Zerocoin, the Anonymity property holds as does the Balance property if no more than $n - 1$ Zerocoin nodes are malicious.

Some care must be taken when electing these nodes to prevent a Sybil attack. Thankfully, if we require that such a node also produce blocks in the Bitcoin block chain, we have

¹⁶The reference `bitcoind` client stores transactions as STL Vectors, which require contiguous segments of memory. As such, storing Zerocoin proofs in the transaction might cause memory issues far faster than expected.

¹⁷Furthermore, this solution allows for the intriguing possibility that proofs be allowed to vanish after they have been sufficiently verified by the network and entombed in the block chain. However, it is not clear how this interacts with Bitcoin in theory or practice.

a decent deterrent. Furthermore, because any malfeasance of these nodes is readily detectable (since they signed an invalid Zerocoin transaction), third parties can audit these nodes and potentially hold funds in escrow to deter fraud.

VI. REAL WORLD SECURITY AND PARAMETER CHOICE

A. Anonymity of Zerocoin

Definition 3.2 states that given two Zerocoin mints and one spend, one cannot do much better than guess which minted coin was spent. Put differently, an attacker learns no more from our scheme than they would from observing the mints and spends of some ideal scheme. However, even an ideal scheme imposes limitations. For example, consider a case where N coins are minted, then all N coins are subsequently spent. If another coin is minted after this point, the size of the anonymity set for the next spend is $k = 1$, not $k = 11$, since it is clear to all observers that the previous coins have been used. We also stress that — as in many anonymity systems — privacy may be compromised by an attacker who mints a large fraction of the active coins. Hence, a lower bound on the anonymity provided is the number of coins minted by honest parties between a coin’s mint and its spend. An upper bound is the total set of minted coins.

We also note that Zerocoin reveals the number of minted and spent coins to all users of the system, which provides a potential source of information to attackers. This is in contrast to many previous e-cash schemes which reveal this information primarily to merchants and the bank. However, we believe this may be an advantage rather than a loss, since the bank is generally considered an adversarial party in most e-cash security models. The public model of Zerocoin actually removes an information asymmetry by allowing users to determine when such conditions might pose a problem.

Lastly, Zerocoin does not hide the denominations used in a transaction. In practice, this problem can be avoided by simply fixing one or a small set of coin denominations and exchanging coins until one has those denominations, or by simply using Zerocoin to anonymize bitcoins.

B. Parameters

Generally, cryptographers specify security in terms of a single, adjustable security parameter λ . Indeed, we have used this notation throughout the previous sections. In reality, however, there are three distinct security choices for Zerocoin which affect either the system’s anonymity, its resilience to counterfeiting, or both. These are:

- 1) The size of the Schnorr group used in the coin commitments.
- 2) The size of the RSA modulus used in the accumulator.
- 3) λ_{zkp} , the security of the zero-knowledge proofs.

Commitments. Because Pedersen commitments are information theoretically hiding for *any* Schnorr group whose order is large enough to fit the committed values, the size of

the group used does not affect the long term anonymity of Zerocoin. The security of the commitment scheme does, however, affect counterfeiting: an attacker who can break the binding property of the commitment scheme can mint a zerocoin that opens to at least two different serial numbers, resulting in a double spend. As a result, the Schnorr group must be large enough that such an attack cannot be feasibly mounted in the lifetime of a coin. On the other hand, the size of the signature of knowledge π used in coin spends increases linearly with the size of the Schnorr group.

One solution is to minimize the group size by announcing fresh parameters for the commitment scheme periodically and forcing old zerocoins to expire unless exchanged for new zerocoins minted under the fresh parameters.¹⁸ Since all coins being spent on the network at time t are spent with the current parameters and all previous coins can be converted to fresh ones, this does not decrease the anonymity of the system. It does, however, require users to convert old zerocoins to fresh ones before the old parameters expire. For our prototype implementation, we chose to use 1024 bit parameters on the assumption that commitment parameters could be regenerated periodically. We explore the possibility of extensions to Zerocoin that might enable smaller groups in Section IX.

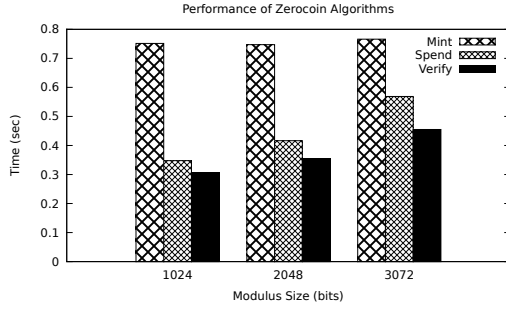
Accumulator RSA key. Because generating a new accumulator requires either a new trusted setup phase or generating a new RSA UFO [26], we cannot re-key very frequently. As a result, the accumulator is long lived, and thus we truly need long term security. Therefore we currently propose an RSA key of at least 3072 bits. We note that this does not greatly affect the size of the coins themselves, and, because the proof of accumulator membership is efficient, this does not have a large adverse effect on the overall coin spend proof size. Moreover, although re-keying the accumulator is expensive, it need not reduce the anonymity of the system since the new parameters can be used to re-accumulate the existing coin set and hence anonymize spends over that whole history.

Zero-knowledge proof security λ_{zkp} . This parameter affects the anonymity and security of the zero-knowledge proof. It also greatly affects the size of the spend proof. Thankfully, since each proof is independent, it applies per proof and therefore per spend. As such, a dishonest party would have to expend roughly $2^{\lambda_{zkp}}$ effort to forge a *single* coin or could link a *single* coin mint to a spend with probability roughly $\frac{1}{2^{\lambda_{zkp}}}$. As such we pick $\lambda_{zkp} = 80$ bits.

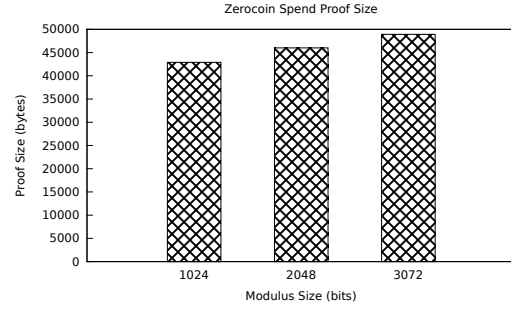
VII. PERFORMANCE

To validate our results, we conducted several experiments using the modified `bitcoind` implementation described in Section V. We ran our experiments with three different

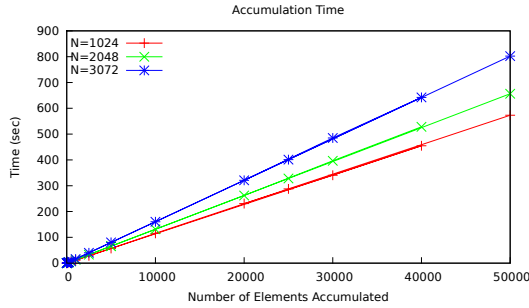
¹⁸Note that this conversion need not involve a full spend of the coins. The user may simply reveal the trapdoor for the old coin, since the new zerocoin will still be unlinkable when properly spent.



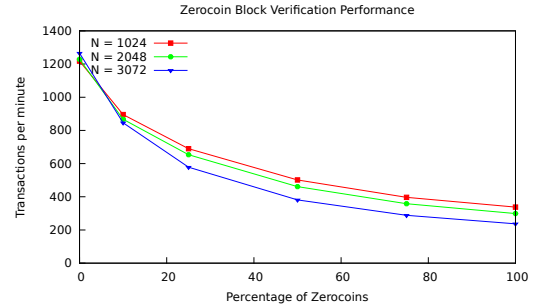
(a) Times for a single Zerocoin operation measured in seconds. These operations do not include the time required to compute the accumulator.



(b) Zerocoin proof sizes measured in bytes as a function of RSA modulus size.



(c) Time required to accumulate x elements. Note, this cost is amortized when computing the global accumulator.



(d) Transaction verifications per minute as a function of the percentage of Zerocoin transactions in the network (where half are mints and half are spends). Note, since we plot the reciprocal of transaction time, this graph appears logarithmic even though Zerocoin scales linearly.

Figure 3: Zerocoin performance as a function of parameter size.

parameter sizes, where each corresponds to a length of the RSA modulus N : 1024 bits, 2048 bits, and 3072 bits.¹⁹

We conducted two types of experiments: (1) *microbenchmarks* that measure the performance of our cryptographic constructions and (2) tests of our whole modified Bitcoin client measuring the time to verify Zerocoin carrying blocks. The former gives us a reasonable estimate of the cost of minting a single zerocoin, spending it, and verifying the resulting transaction. The latter gives us an estimate of Zerocoin’s impact on the existing Bitcoin network and the computational cost that will be born by each node that verifies Zerocoin transactions.

All of our experiments were conducted on an Intel Xeon E3-1270 V2 (3.50GHz quad-core processor with hyper-threading) with 16GB of RAM, running 64-bit Ubuntu Server 11.04 with Linux kernel 2.6.38.

¹⁹These sizes can be viewed as *roughly* corresponding to a discrete logarithm/factorization security level of 2^{80} , 2^{112} , and 2^{128} respectively. Note that the choice of N determines the size of the parameter p . We select $|q|$ to be roughly twice the estimated security level.

A. Microbenchmarks

To evaluate the performance of our Mint, Spend, and Verify algorithms in isolation, we conducted a series of microbenchmarks using the Charm (Python) implementation. Our goal in these experiments was to provide a direct estimate of the performance of our cryptographic primitives.

Experimental setup. One challenge in conducting our microbenchmarks is the accumulation of coins in $\mathbf{C}$ for the witness in $\text{Spend}(\dots)$ or for the global accumulator in both $\text{Spend}(\dots)$ and $\text{Verify}(\dots)$. This is problematic for two reasons. First, we do not know how large $\mathbf{C}$ will be in practice. Second, in our implementation accumulations are incremental. To address these issues we chose to break our microbenchmarks into two separate experiments. The first experiment simply computes the accumulator for a number of possible sizes of $\mathbf{C}$, ranging from 1 to 50,000 elements. The second experiment measures the runtime of the $\text{Spend}(\dots)$ and $\text{Verify}(\dots)$ routines with a precomputed accumulator and witness (A, ω) .

We conducted our experiments on a single thread of the processor, using all three parameter sizes. All experiments

were performed 500 times, and the results given represent the average of these times. Figure 3a shows the measured times for computing the coin operations, Figure 3b shows the resulting proof sizes for each security parameter, and Figure 3c shows the resulting times for computing the accumulator. We stress that accumulation in our system is *incremental*, typically over at most the 200–500 transactions in a block (which takes at worst eight seconds), and hence the cost of computing the global accumulator is therefore amortized. The only time one might accumulate 50,000 coins at one time would be when generating the witness for a very old zerocoin.

B. Block Verification

How Zerocoin affects network transaction processing determines its practicality and scalability. Like all transactions, Zerocoin spends must be verified first by the miner to make sure he is not including invalid transactions in a block and then again by the network to make sure it is not including an invalid block in the block chain. In both cases, this entails checking that $\text{Verify}(\dots) = 1$ for each Zerocoin transaction and computing the accumulator checkpoint.

We need to know the impact of this for two reasons. First, the Bitcoin protocol specifies that a new block should be created on average once every 10 minutes.²⁰ If verification takes longer than 10 minutes for blocks with a reasonable number of zerocoins, then the network cannot function.²¹ Second, while the cost of generating these blocks and verifying their transactions can be offset by transaction fees and coin mining, the cost of verifying blocks prior to appending them to the block chain is only offset for mining nodes (who can view it as part of the cost of mining a new block). This leaves anyone else verifying the block chain with an uncompensated computational cost.

Experimental setup. To measure the effect of Zerocoin on block verification time, we measure how long it takes our modified `bitcoind` client to verify externally loaded test blocks containing 200, 400, and 800 transactions where 0, 10, 25, 75, or 100 percent of the transactions are Zerocoin transactions (half of which are mints and half are spends). We repeat this experiment for all three security parameters.

Our test data consists of two blocks. The first contains z Zerocoin mints that must exist for any spends to occur. The second block is our actual test vector. It contains, in a random order, z Zerocoin spends of the coins in the previous block, z Zerocoin mints, and s standard Bitcoin `sendToAddress` transactions. We measure how long the `processblock` call of the `bitcoind` client takes to verify the second block containing the mix of Zerocoin and classical Bitcoin

transactions. For accuracy, we repeat these measurements 100 times and average the results. The results are presented in Figure 3d.

C. Discussion

Our results show that Zerocoin scales beyond current Bitcoin transaction volumes. Though we require significant computational effort, verification does not fundamentally threaten the operation of the network: even with a block containing 800 Zerocoin transactions — roughly double the average size of a Bitcoin block currently — verification takes less than five minutes. This is under the unreasonable assumption that all Bitcoin transactions are supplanted by Zerocoin transactions.²² In fact, we can scale well beyond Bitcoin’s current average of between 200 and 400 transactions per block [29] if Zerocoin transactions are not the majority of transactions on the network. If, as the graph suggests, we assume that verification scales linearly, then we can support a 50% transaction mix out to 350 transactions per minute (3,500 transactions per block) and a 10% mixture out to 800 transactions per minute (8,000 per block).

One remaining question is at what point we start running a risk of coin serial number collisions causing erroneous double spends. Even for our smallest serial numbers — 160 bits — the collision probability is small, and for the 256 bit serial numbers used with the 3072 bit accumulator, our collision probability is at worst equal to the odds of a collision on a normal Bitcoin transaction which uses SHA-256 hashes.

We stress several caveats about the above data. First, our prototype system does not exploit any parallelism either for verifying multiple Zerocoin transactions or in validating an individual proof. Since the only serial dependency for either of these tasks is the (fast) duplicate serial number check, this offers the opportunity for substantial improvement.

Second, the above data is not an accurate estimate of the financial cost of Zerocoin for the network: (a) it is an *overestimate* of a mining node’s extra effort when verifying proposed blocks since in practice many transactions in a received block will already have been received and validated by the node as it attempts to construct its own contribution to the block chain; (b) execution time is a poor metric in the context of Bitcoin, since miners are concerned with actual monetary operating cost; (c) since mining is typically performed using GPUs and to a lesser extent FPGAs and ASICs, which are far more efficient at computing hash collisions, the CPU cost measured here is likely insignificant.

Finally, our experiment neglects the load on a node both from processing incoming transactions and from solving the proof of work. Again, we contend that most nodes will probably use GPUs for mining, and as such the latter is not an issue. The former, however, remains an unknown. At

²⁰This rate is maintained by a periodic network vote that adjusts the difficulty of the Bitcoin proof of work.

²¹For blocks with unreasonable numbers of Zerocoin transaction we can simply extend `bitcoind`’s existing anti-DoS mechanisms to reject the block and blacklist its origin.

²²In practice we believe Zerocoin will be used to anonymize bitcoins that will then be spent in actual transactions, resulting in far lower transaction volumes.

the very least it seems unlikely to disproportionately affect Zerocoin performance.

VIII. PREVIOUS WORK

A. E-Cash and Bitcoin

Electronic cash has long been a research topic for cryptographers. Many cryptographic e-cash systems focus on user privacy and typically assume the existence of a semi-trusted coin issuer or bank. E-cash schemes largely break down into *online* schemes where users have contact with a bank or registry and *offline* schemes where spending can occur even without a network connection. Chaum introduced the first online cryptographic e-cash system [30] based on RSA signatures, later extending this work to the offline setting [31] by de-anonymizing users who double-spent. Many subsequent works improved upon these techniques while maintaining the requirement of a trusted bank: for example, by making coins divisible [32,33] and reducing wallet size [34]. One exception to the rule above comes from Sander and Ta-Shma [35] who presciently developed an alternative model that is reminiscent of our proposal: the central bank is replaced with a hash chain and signatures with accumulators. Unfortunately the accumulator was not practical, a central party was still required, and no real-world system existed to compute the chain.

Bitcoin's primary goal, on the other hand, is not anonymity. It has its roots in a non-academic proposal by Wei Dai for a distributed currency based on solving computational problems [36]. In Dai's original proposal anyone could create currency, but all transactions had to be broadcast to all clients. A second variant limited currency generation and transaction broadcast to a set of servers, which is effectively the approach Bitcoin takes. This is a marked distinction from most, if not all, other e-cash systems since there is no need to select one or more trusted parties. There is a general assumption that a majority of the Bitcoin nodes are honest, but anyone can join a node to the Bitcoin network, and anyone can get the entire transaction graph. An overview of Bitcoin and some of its shortcomings was presented by Barber *et. al.* in [2].

B. Anonymity

Numerous works have shown that "pseudonymized" graphs can be re-identified even under passive analysis. Narayanan and Shmatikov [5] showed that real world social networks can be passively de-anonymized. Similarly, Backstrom *et al.* [37] constructed targeted attacks against anonymized social networks to test for relationships between vertices. Previously, Narayanan and Shmatikov de-anonymized users in the Netflix prize data set by correlating data from IMDB [38].

Bitcoin itself came into existence in 2009 and is now beginning to receive scrutiny from privacy researchers. De-anonymization techniques were applied effectively to Bitcoin even at its relatively small 2011 size by Reid and Harrigan [3].

Ron and Shamir examined the general structure of the Bitcoin network graph [1] after its nearly 3-fold expansion. Finally, we have been made privately aware of two other early-stage efforts to examine Bitcoin anonymity.

IX. CONCLUSION AND FUTURE WORK

Zerocoin is a distributed e-cash scheme that provides strong user anonymity and coin security under the assumption that there is a distributed, online, append-only transaction store. We use Bitcoin to provide such a store and the backing currency for our scheme. After providing general definitions, we proposed a concrete realization based on RSA accumulators and non-interactive zero-knowledge signatures of knowledge. Finally, we integrated our construction into Bitcoin and measured its performance.

Our work leaves several open problems. First, although our scheme is workable, the need for a double-discrete logarithm proof leads to large proof sizes and verification times. We would prefer a scheme with both smaller proofs and greater speed. This is particularly important when it comes to reducing the cost of third-party verification of Zerocoin transactions. There are several promising constructions in the cryptographic literature, e.g., bilinear accumulators, mercurial commitments [13,39]. While we were not able to find an analogue of our scheme using alternative components, it is possible that further research will lead to other solutions. Ideally such an improvement could produce a drop-in replacement for our existing implementation.

Second, Zerocoin currently derives both its anonymity and security against counterfeiting from strong cryptographic assumptions at the cost of substantially increased computational complexity and size. As discussed in section VI-B, anonymity is relatively cheap, and this cost is principally driven by the anti-counterfeiting requirement, manifesting itself through the size of the coins and the proofs used.

In Bitcoin, counterfeiting a coin is not computationally prohibitive, it is merely computationally costly, requiring the user to obtain control of at least 51% of the network. This provides a possible alternative to our standard cryptographic assumptions: rather than the strong assumption that computing discrete logs is infeasible, we might construct our scheme on the weak assumption that there is no financial incentive to break our construction as the cost of computing a discrete log exceeds the value of the resulting counterfeit coins.

For example, if we require spends to prove that fresh and random bases were used in the commitments for the corresponding mint transaction (e.g., by selecting the bases for the commitment from the hash of the coin serial number and proving that the serial number is fresh), then it appears that an attacker can only forge a *single* zerocoin per discrete log computation. Provided the cost of computing such a discrete log is greater than the value of a zerocoin, forging a coin is not profitable. How small this allows us to make

the coins is an open question. There is relatively little work comparing the asymptotic difficulty of solving multiple distinct discrete logs in a fixed group,²³ and it is not clear how theory translates into practice. We leave these questions, along with the security of the above proposed construction, as issues for future work.

Finally, we believe that further research could lead to different tradeoffs between security, accountability, and anonymity. A common objection to Bitcoin is that it can facilitate money laundering by circumventing legally binding financial reporting requirements. We propose that additional protocol modifications (e.g., the use of anonymous credentials [40]) might allow users to maintain their anonymity while demonstrating compliance with reporting requirements.

Acknowledgements. We thank Stephen Checkoway, George Danezis, and the anonymous reviewers for their helpful comments. The research in this paper was supported in part by the Office of Naval Research under contract N00014-11-1-0470, and DARPA and the Air Force Research Laboratory (AFRL) under contract FA8750-11-2-0211.

REFERENCES

- [1] D. Ron and A. Shamir, "Quantitative Analysis of the Full Bitcoin Transaction Graph," Cryptology ePrint Archive, Report 2012/584, 2012, <http://eprint.iacr.org/>.
- [2] S. Barber, X. Boyen, E. Shi, and E. Uzun, "Bitter to better – how to make bitcoin a better currency," in *Financial Cryptography 2012*, vol. 7397 of LNCS, 2012, pp. 399–414.
- [3] F. Reid and M. Harrigan, "An analysis of anonymity in the Bitcoin system," in *Privacy, security, risk and trust (PASSAT), 2011 IEEE Third International Conference on Social Computing (SOCIALCOM)*. IEEE, 2011, pp. 1318–1326.
- [4] T. B. Lee, "A risky currency? Alleged \$500,000 Bitcoin heist raises questions," Available at <http://arstechnica.com/>, June 2011.
- [5] A. Narayanan and V. Shmatikov, "De-anonymizing social networks," in *Security and Privacy, 2009 30th IEEE Symposium on*. IEEE, 2009, pp. 173–187.
- [6] "Bitcoin fog company," <http://www.bitcoinfog.com/>.
- [7] "The Bitcoin Laundry," <http://www.bitcoinlaundry.com/>.
- [8] "Blind Bitcoin," Information at https://en.bitcoin.it/wiki/Blind_Bitcoin_Transfers.
- [9] [Online]. Available: <https://www.torproject.org/>
- [10] J. Benaloh and M. de Mare, "One-way accumulators: a decentralized alternative to digital signatures," in *EUROCRYPT '93*, vol. 765 of LNCS, 1994, pp. 274–285.
- [11] N. Barić and B. Pfitzmann, "Collision-free accumulators and fail-stop signature schemes without trees," in *EUROCRYPT '97*, vol. 1233 of LNCS, 1997, pp. 480–494.
- [12] J. Camenisch and A. Lysyanskaya, "Dynamic accumulators and application to efficient revocation of anonymous credentials," in *CRYPTO '02*, 2002, pp. 61–76.
- [13] L. Nguyen, "Accumulators from bilinear pairings and applications," in *Topics in Cryptology – CT-RSA 2005*, 2005, vol. 3376 LNCS, pp. 275–292.
- [14] J. Camenisch, M. Kohlweiss, and C. Soriente, "An accumulator based on bilinear maps and efficient revocation for anonymous credentials," in *PKC '09*, vol. 5443 of LNCS, 2009, pp. 481–500.
- [15] S. Nakamoto, "Bitcoin: A peer-to-peer electronic cash system, 2009," 2012. [Online]. Available: <http://www.bitcoin.org/bitcoin.pdf>
- [16] G. O. Karame, E. Androulaki, and S. Capkun, "Two bitcoins at the price of one? double-spending attacks on fast payments in bitcoin," Cryptology ePrint Archive, Report 2012/248, 2012, <http://eprint.iacr.org/>.
- [17] European Central Bank, "Virtual currency schemes," Available at <http://www.ecb.europa.eu/pub/pdf/other/virtualcurrencyschemes201210en.pdf>, October 2012.
- [18] C.-P. Schnorr, "Efficient signature generation for smart cards," *Journal of Cryptology*, vol. 4, no. 3, pp. 239–252, 1991.
- [19] R. Cramer, I. Damgård, and B. Schoenmakers, "Proofs of partial knowledge and simplified design of witness hiding protocols," in *CRYPTO '94*, vol. 839 of LNCS, 1994, pp. 174–187.
- [20] J. Camenisch and M. Michels, "Proving in zero-knowledge that a number n is the product of two safe primes," in *EUROCRYPT '99*, vol. 1592 of LNCS, 1999, pp. 107–122.
- [21] J. L. Camenisch, "Group signature schemes and payment systems based on the discrete logarithm problem," Ph.D. dissertation, ETH Zürich, 1998.
- [22] S. Brands, "Rapid demonstration of linear relations connected by boolean operators," in *EUROCRYPT '97*, vol. 1233 of LNCS, 1997, pp. 318–333.
- [23] A. Fiat and A. Shamir, "How to prove yourself: Practical solutions to identification and signature problems," in *CRYPTO '86*, vol. 263 of LNCS, 1986, pp. 186–194.
- [24] M. Chase and A. Lysyanskaya, "On signatures of knowledge," in *CRYPTO '06*, vol. 4117 of LNCS, 2006, pp. 78–96.
- [25] J. Camenisch and M. Stadler, "Efficient group signature schemes for large groups," in *CRYPTO '97*, vol. 1296 of LNCS, 1997, pp. 410–424.
- [26] T. Sander, "Efficient accumulators without trapdoor extended abstract," in *Information and Communication Security*, vol. 1726 of LNCS, 1999, pp. 252–262.
- [27] J. A. Akinyele, C. Garman, I. Miers, M. W. Pagano, M. Rushanan, M. Green, and A. D. Rubin, "Charm: A framework for rapidly prototyping cryptosystems," *To appear, Journal of Cryptographic Engineering*, 2013. [Online]. Available: <http://dx.doi.org/10.1007/s13389-013-0057-3>
- [28] [Online]. Available: https://en.bitcoin.it/wiki/BIP_0016

²³We note that both SSH and the Internet Key Exchange protocol used in IPv6 use fixed Diffie-Hellman parameters.

- [29] [Online]. Available: <http://blockchain.info/charts/n-transactions-per-block>
- [30] D. Chaum, "Blind signatures for untraceable payments," in *CRYPTO '82*. Plenum Press, 1982, pp. 199–203.
- [31] D. Chaum, A. Fiat, and M. Naor, "Untraceable electronic cash," in *CRYPTO 88*, 1990, vol. 403 of LNCS, pp. 319–327.
- [32] T. Okamoto and K. Ohta, "Universal electronic cash," in *CRYPTO 91*, 1992, vol. 576 of LNCS, pp. 324–337.
- [33] T. Okamoto, "An efficient divisible electronic cash scheme," in *Crypt '95*, 1995, vol. 963 of LNCS, pp. 438–451.
- [34] J. Camenisch, S. Hohenberger, and A. Lysyanskaya, "Compact e-cash," in *EUROCRYPT '05*, 2005, vol. 3494 of LNCS, pp. 566–566.
- [35] T. Sander and A. Ta-Shma, "Auditable, anonymous electronic cash (extended abstract)," in *CRYPTO '99*, vol. 1666 of LNCS, 1999, pp. 555–572.
- [36] W. Dai. B-money proposal. [Online]. Available: <http://www.weidai.com/bmoney.txt>
- [37] L. Backstrom, C. Dwork, and J. Kleinberg, "Wherefore art thou r3579x?: Anonymized social networks, hidden patterns, and structural steganography," in *Proceedings of the 16th international conference on World Wide Web*, ser. WWW '07. New York, NY, USA: ACM, 2007, pp. 181–190.
- [38] A. Narayanan and V. Shmatikov, "Robust de-anonymization of large sparse datasets," in *IEEE Symposium on Security and Privacy*. IEEE, 2008, pp. 111–125.
- [39] M. Chase, A. Healy, A. Lysyanskaya, T. Malkin, and L. Reyzin, "Mercurial commitments with applications to zero-knowledge sets," in *EUROCRYPT '05*, vol. 3494, 2005, pp. 422–439.
- [40] J. Camenisch and A. Lysyanskaya, "An efficient system for non-transferable anonymous credentials with optional anonymity revocation," in *EUROCRYPT '01*, vol. 2045 of LNCS, 2001, pp. 93–118.
- [41] —, "Dynamic accumulators and application to efficient revocation of anonymous credentials," in *CRYPTO '02*, 2002, extended Abstract. [Online]. Available: <http://cs.brown.edu/~anna/papers/camllys02.pdf>
- [42] D. Pointcheval and J. Stern, "Provably secure blind signature schemes," in *ASIACRYPT '96*, vol. 1163 of LNCS, 1996, pp. 252–265.

APPENDIX A. SECURITY PROOFS

A. Proof Sketch of Theorem 4.1

Proof sketch. Consider the following simulation. First, the simulation generates $params \leftarrow \text{Setup}(1^\lambda)$ and two primes C_0, C_1 that are uniformly sampled from the set of prime numbers in the range $[\mathcal{A}, \mathcal{B}]$.²⁴ $\mathcal{A}_1$ takes these values as input and outputs a set $\mathbf{C}$ and transaction string R using

²⁴Where $\mathcal{A}$ and $\mathcal{B}$ can be chosen with arbitrary polynomial dependence on the security parameter, as long as $2 < \mathcal{A}$ and $\mathcal{B} < \mathcal{A}^2$. [41] For a full description, see [41, §3.2 and §3.3].

any strategy it wishes. Next the simulation runs $\mathcal{A}_2$ with a simulated²⁵ zero-knowledge signature of knowledge π and a random coin serial number S sampled from $\mathbb{Z}_q^*$. Note that if π is at least computationally zero-knowledge then with all but negligible probability, all values provided to $\mathcal{A}$ are distributed as in the real protocol. Moreover, all are independent of the bit b . By implication, $\Pr[b = b'] = 1/2 + \nu(\lambda)$ and $\mathcal{A}$'s advantage is negligible. $\square$

B. Proof of Theorem 4.2

Proof: Let $\mathcal{A}$ be an adversary that wins the Balance game with non-negligible advantage ϵ . We construct an algorithm $\mathcal{B}$ that takes input (p, q, g, h) , where $\mathbb{G} = \langle g \rangle = \langle h \rangle$ is a subgroup of $\mathbb{Z}_p^*$ of order q , and outputs $x \in \mathbb{Z}_q$ such that $g^x \equiv h \pmod{p}$. $\mathcal{B}$ works as follows:

On input (p, q, g, h) , first generate accumulator parameters N, u as in the Setup routine and set $params \leftarrow (N, u, p, q, g, h)$. For $i = 1$ to K , compute $(c_i, skc_i) \leftarrow \text{Mint}(params)$, where $skc_i = (S_i, r_i)$, and run $\mathcal{A}(params, c_1, \dots, c_K)$. Answer each of $\mathcal{A}$'s queries to $\mathcal{O}_{spend}$ using the appropriate trapdoor information. Let $(S_1, R_1), \dots, (S_l, R_l)$ be the set of values recorded by the oracle.

At the conclusion of the game, $\mathcal{A}$ outputs a set of M coins $(c'_1, \dots, c'_M)$ and a corresponding set of $M + 1$ valid tuples $(\pi'_i, S'_i, R'_i, C'_i)$. For $j = 1$ to $M + 1$, apply the ZKSoK extractor to the j^{th} zero-knowledge proof π'_j to extract the values (c_j^*, r_j^*) and perform the following steps:

- 1) If the extractor fails, abort and signal $\text{EVENT}_{\text{EXT}}$.
- 2) If $c_j^* \notin \mathbf{C}'_j$, abort and signal $\text{EVENT}_{\text{ACC}}$.
- 3) If $c_j^* \in \{c_1, \dots, c_K\}$:
 - a) If for some i , $(S'_j, r_j^*) = (S_i, r_i)$ and $R'_j \neq R_i$, abort and signal $\text{EVENT}_{\text{FORGE}}$.
 - b) Otherwise if for some i , $(S'_j, r_j^*) = (S_i, r_i)$, abort and signal $\text{EVENT}_{\text{COL}}$.
 - c) Otherwise set $(a, b) = (S_i, r_i)$.
- 4) If for some i , $c_j^* = c_i^*$, set $(a, b) = (S'_i, r_i^*)$.

If the simulation did not abort, we now have $(c_j^*, r_j^*, S'_j, a, b)$ where (by the soundness of π) we know that $c_j^* \equiv g^{S'_j} h^{r_j^*} \equiv g^a h^b \pmod{p}$. To solve for $\log_g h$, output $(S'_j - a) \cdot (b - r_j^*)^{-1} \pmod{q}$.

Analysis. Let us briefly explain the conditions behind this proof. When the simulation does *not* abort, we are able to extract $(c'_1, \dots, c'_{M+1})$ where the win conditions enforce that $\forall j \in [1, M + 1]$, $c'_j \in \mathbf{C}'_j \in \{c_1, \dots, c_K, c'_1, \dots, c'_M\}$ and each S'_j is distinct (and does not match any serial number output by $\mathcal{O}_{spend}$). Since $\mathcal{A}$ has produced M coins and yet spent $M + 1$, there are only two possibilities:

- 1) $\mathcal{A}$ has spent one of the challenger's coins but has provided a new serial number for it. For some (i, j) ,

²⁵Our proofs assume the existence of an efficient simulator and extractor for the ZKSoK. See Appendix B.

$c_j^* = c_i \in \{c_1, \dots, c_K\}$. Observe that in cases where the simulation does not abort, the logic of the simulation always results in a pair $(a, b) = (S_i, r_i)$ where $g^a h^b \equiv g^{S_i} h^{r_i} \equiv c_j^* \pmod{p}$ and $(a, b) \neq (S_j', r_j^*)$.

- 2) $\mathcal{A}$ has spent the same coin twice. For some (i, j) , $c_j^* = c_i^*$ and yet $(S_j' \neq S_i')$. Thus again we identify a pair $(a, b) = (S_i', r_i^*)$ that satisfies $g^a h^b \equiv c_j^* \pmod{p}$ where $(a, b) \neq (S_j', r_j^*)$.

Finally, we observe that given any such pair (a, b) we can solve for $x = \log_g h$ using the equation above.

Abort probability. It remains only to consider the probability that the simulation aborts. Let $\nu_1(\lambda)$ be the (negligible) probability that the extractor fails on input π . By summation, $\Pr[\text{EVENT}_{\text{EXT}}] \leq (M+1)\nu_1(\lambda)$. Next consider the probability of $\text{EVENT}_{\text{COL}}$. This implies that for some i , $\mathcal{A}$ has produced a pair $(S_j', r_j^*) = (S_i, r_i)$ where S_j' has not been produced by $\mathcal{O}_{\text{spend}}$. Observe that there are l distinct pairs (S, r) that satisfy $c_j^* = g^S h^r \pmod{p}$ and $\mathcal{A}$'s view is independent of the specific pair chosen. Thus $\Pr[\text{EVENT}_{\text{COL}}] \leq 1/l$.

Next, we argue that under the Strong RSA and Discrete Log assumptions, $\Pr[\text{EVENT}_{\text{ACC}}] \leq \nu_2(\lambda)$ and $\Pr[\text{EVENT}_{\text{FORGE}}] \leq \nu_3(\lambda)$. We show this in Lemmas A.1 and A.2 below. If $\mathcal{A}$ succeeds with advantage ϵ , then by summing the above probabilities we show that $\mathcal{B}$ succeeds with probability $\geq \epsilon - ((M+1)\nu_1(\lambda) + \nu_2(\lambda) + \nu_3(\lambda) + 1/l)$. We conclude with the remaining Lemmas.

Lemma A.1: Under the Strong RSA assumption, $\Pr[\text{EVENT}_{\text{ACC}}] \leq \nu_2(\lambda)$.

Proof sketch. The basic idea of this proof is that an $\mathcal{A}'$ who induces $\text{EVENT}_{\text{ACC}}$ with non-negligible probability can be used to find a witness ω to the presence of a non-member in a given accumulator. Given this value, we apply the technique of [12, §3] to solve the Strong RSA problem. For the complete details we refer the reader to [12, §3] and simply outline the remaining details of the simulation.

Let $\mathcal{A}'$ be an adversary that induces $\text{EVENT}_{\text{ACC}}$ with non-negligible probability ϵ' in the simulation above. We use $\mathcal{A}'$ to construct a Strong RSA solver $\mathcal{B}'$ that succeeds with non-negligible probability. On input a Strong RSA instance (N, u) , $\mathcal{B}'$ selects (p, q, g, h) as in Setup and sets $\text{params} = (N, u, p, q, g, h)$. It generates $(c_1, \dots, c_K)$ as in the previous simulation and runs $\mathcal{A}'$. To induce $\text{EVENT}_{\text{ACC}}$, $\mathcal{A}'$ produces valid output $(\pi', \mathbf{C}')$ and (by extraction from π') a $c^* \notin \mathbf{C}'$. $\mathcal{B}'$ now extracts ω^* from π' using the technique described in [12, §3] and uses the resulting value to compute a solution to the Strong RSA instance. $\square$

Lemma A.2: Under the Discrete Logarithm assumption, $\Pr[\text{EVENT}_{\text{FORGE}}] \leq \nu_3(\lambda)$.

Proof sketch. We leave a proof for the full version of this paper, but it is similar to those used by earlier schemes,

e.g., [25]. Let $\mathcal{A}'$ be an adversary that induces $\text{EVENT}_{\text{FORGE}}$ with non-negligible probability ϵ' in the simulation above. On input a discrete logarithm instance, we run $\mathcal{A}'$ as in the main simulation except that we do not use the trapdoor information to answer $\mathcal{A}'$'s oracle queries. Instead we select random serial numbers and simulate the ZKSoK responses to $\mathcal{A}'$ by programming the random oracle. When $\mathcal{A}'$ outputs a forgery on a repeated serial number but a different string R' than used in any previous proof, we rewind $\mathcal{A}'$ to extract the pair (S_j', r_j^*) and solve for the discrete logarithm as in the main simulation. $\square$

APPENDIX B.

ZERO-KNOWLEDGE PROOF CONSTRUCTION

The signature of knowledge

$$\pi = \text{ZKSoK}[R]\{(c, w, r) :$$

$$\text{AccVerify}((N, u), A, c, w) = 1 \wedge c = g^S h^r\}$$

is composed of two proofs that (1) a committed value c is accumulated and (2) that c is a commitment to S . The former proof is detailed in [41, §3.3 and Appendix A]. The latter is a double discrete log signature of knowledge that, although related to previous work [21, §5.3.3], is new (at least to us). A proof of its security can be found in the full version of this paper. It is constructed as follows:

Given $y_1 = g^{a^x b^z} h^w$.

Let $l \leq k$ be two security parameters and $H : \{0, 1\}^* \rightarrow \{0, 1\}^k$ be a cryptographic hash function. Generate $2l$ random numbers $r_1, \dots, r_l$ and $v_1, \dots, v_l$. Compute, for $1 \leq i \leq l$, $t_i = g^{a^x b^{r_i}} h^{v_i}$. The signature of knowledge on the message m is $(c, s_1, s_2, \dots, s_l, s'_1, s'_2, \dots, s'_l)$, where:

$$c = H(m \| y_1 \| a \| b \| g \| h \| x \| t_1 \| \dots \| t_l)$$

and

if $c[i] = 0$ **then** $s_i = r_i, s'_i = v_i$;

else $s_i = r_i - z, s'_i = v_i - w b^{r_i - z}$;

To verify the signature it is sufficient to compute:

$$c' = H(m \| y_1 \| a \| b \| g \| h \| x \| \bar{t}_1 \| \dots \| \bar{t}_l)$$

with

if $c[i] = 0$ **then** $\bar{t}_i = g^{a^x b^{s_i}} h^{s'_i}$;

else $\bar{t}_i = y_1^{b^{s_i}} h^{s'_i}$;

and check whether $c = c'$.

Simulating and extracting. Our proofs in Appendix A assume the existence of an efficient simulator and extractor for the signature of knowledge. These may be constructed using well-understood results in the random oracle model, e.g., [25, 42]. We provide further details in the full version of this work.

PCF: A Portable Circuit Format For Scalable Two-Party Secure Computation

Ben Kreuter
Computer Science Dept.
U. Virginia

Benjamin Mood
Computer and Info. Science Dept.
U. Oregon

abhi shelat
Computer Science Dept.
U. Virginia

Kevin Butler
Computer and Info. Science Dept.
U. Oregon

Abstract

A secure computation protocol for a function $f(x,y)$ must leak no information about inputs x,y during its execution; thus it is imperative to compute the function f in a data-oblivious manner. Traditionally, this has been accomplished by compiling f into a boolean circuit. Previous approaches, however, have scaled poorly as the circuit size increases. We present a new approach to compiling such circuits that is substantially more efficient than prior work. Our approach is based on online circuit compression and lazy gate generation. We implemented an optimizing compiler for this new representation of circuits, and evaluated the use of this representation in two secure computation environments. Our evaluation demonstrates the utility of this approach, allowing us to scale secure computation beyond any previous system while requiring substantially less CPU time and disk space. In our largest test, we evaluate an RSA-1024 signature function with more than 42 billion gates, that was generated and optimized using our compiler. With our techniques, the bottleneck in secure computation lies with the cryptographic primitives, not the compilation or storage of circuits.

1 Introduction

Secure function evaluation (SFE) refers to several related cryptographic constructions for evaluating functions on unknown inputs. Typically, these constructions require an *oblivious* representation of the function being evaluated, which ensures that the control flow of the algorithm will not depend on its input; in the two party case, boolean circuits are most frequently seen. These oblivious representations are often large, with millions and in some cases billions of gates even for relatively simple functions, which has motivated the creation of software tools for producing such circuits. While there has been substantial work on the practicality of secure function

evaluation, it was only recently that researchers began investigating the practicality of compiling such oblivious representations from high-level descriptions.

The work on generating boolean circuits for SFE has largely focused on two approaches. In one approach, a library for a general purpose programming language such as Java is created, with functions for emitting circuits [13, 20]. For convenience, these libraries typically include pre-built gadgets such as adders or multiplexers, which can be used to create more complete functions. The other approach is to write a compiler for a high level language, which computes and optimizes circuits based on a high level description of the functionality that may not explicitly state how the circuit should be organized [18, 21]. It has been shown in previous work that both of these approaches can scale up to circuits with at least hundreds of millions of gates on modern computer hardware, and in some cases even billions of gates [13, 18].

The approaches described above were limited in terms of their practical utility. Library-based approaches like HEKM [13] or VMCrypt [20] require users to understand the organization of the circuit description of their function, and were unable to apply any optimizations across modules. The Fairplay compiler [21] was unable to scale to circuits with only millions of gates, which excludes many interesting functions that have been investigated. The poor scalability of Fairplay is a result of the compiler first unrolling all loops and inlining all subroutines, storing the results in memory for later compiler stages. The PALC system [23] was more resource efficient than Fairplay, but did not attempt to optimize functions, relying instead on precomputed optimizations of specific subcircuits. The KSS12 [18] system was able to apply some global optimizations and used less memory than Fairplay, but also had to unroll all loops and store the complete circuit description, which caused some functions to require days to compile. Additionally, the language used to describe circuits in the KSS12 system was

brittle and difficult to use; for example, array index values could not be arbitrary functions of loop indices.

1.1 Our Approach

In this work, we demonstrate a new approach to compiling, optimizing, and storing circuits for SFE systems. At a high level, our approach is based on representing the function to be evaluated as a program that computes the circuit representation of the function, similar to the circuit library approaches described in previous work. Our compiler then optimizes this program with the goal of producing a smaller circuit. We refer to our circuit representation as the *Portable Circuit Format* (PCF).

When the SFE system is run, it uses our interpreter to load the PCF program and execute it. As the PCF program runs, it interacts with the SFE system, managing information about gates internally based on the responses from the SFE system itself. In our system, the circuit is ephemeral; it is not necessary to store the entire circuit, and wires will be deleted from memory once they are no longer required.

The key insight of our approach is that it is not necessary to unroll loops until the SFE protocol runs. While previous compilers discard the loop structure of the function, ours emits it as part of the control structure of the PCF program. Rather than dealing directly with wires, our system treats wire IDs as *memory addresses*; a wire is “deleted” by overwriting its location in memory. Loop termination conditions have only one constraint: they must not depend on any secret wire values. There is no upper bound on the number of loop iterations, and the programmer is responsible for ensuring that there are no infinite loops.

To summarize, we present the following contributions:

- A new compiler that has the same advantages as the circuit library approach
- A novel, more general algorithm for translating conditional statements into circuits
- A new representation of circuits that is more compact than previous representations which scales to arbitrary circuit sizes.
- A portable interpreter that can be used with different SFE execution systems regardless of the security model.

Our compiler is a *back end* that can read the bytecode emitted by a *front end*; thus our compiler allows any language to be used for SFE. Instead of focusing on global optimizations of boolean functions, our optimization strategy is based on using higher-level information

from the bytecode itself, which we show to be more effective and less resource-intensive. We present comparisons of our compiler with previous work and show experimental results using our compiler in two complete SFE systems, one based on an updated version of the KSS12 system and one based on HEKM. In some of our test cases, our compiler produced circuits only 30% as large as previous compilers starting from the same source code. With the techniques presented in this work, we demonstrate that the RSA algorithm with a real-world key size and real-world security level can be compiled and run in a garbled circuit protocol using a typical desktop computer. To the best of our knowledge, the RSA-1024 circuit we tested is larger than any previous garbled circuit experiment, with more than 42 billion gates. We also present preliminary results of our system running on smartphones, using a modified version of the HEKM system.

For testing purposes, we used the LCC compiler [8] as a front-end to our system. A high-level view of our system, with the LCC front-end, is given in Figure 1.

The rest of this paper is organized as follows: Section 2 is a review of SFE and garbled circuits; Section 3 presents an overview of bytecode languages; Section 4 explains our compiler design and describes our representation; Section 5 discusses the possibility of using different bytecode and SFE systems; Section 6 details the experiments we performed to evaluate our system and results of those experiments; Section 7 details other work which is related to our own; and Section 8 presents future lines of research.

2 Secure Function Evaluation

The problem of secure two-party computation is to allow two mutually distrustful parties to compute a function of their two inputs without revealing their inputs to the opposing party (privacy) and with a guarantee that the output could not have been manipulated (correctness). Yao was the first to show that such a protocol can be constructed for any computable function, by using the *garbled circuits* technique [30]. In his original formulation, Yao proposed a system that would allow users to describe the function in a high level language, which would then be compiled into a circuit to be used in the garbled circuits protocol. The first complete implementation of this design was the Fairplay system given by Malkhi et al. [21].

Oblivious Transfer One of the key building blocks in Yao’s protocol is *oblivious transfer*, a cryptographic primitive first proposed by Rabin [25]. In this primitive, the “sender” party holds a database of n strings, and the “receiver” party learns exactly k strings with the guarantee that the sender will not learn which k strings were

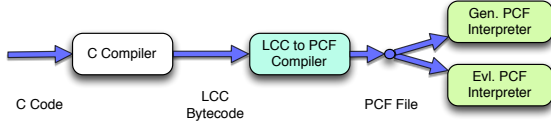


Figure 1: High-level design of our system. We take a C program and compile it down to the LCC bytecode. Our compiler then transforms the LCC bytecode to our new language PCF. Both parties then execute the protocol in their respective role in the SFE protocol. The interpreter could be any execution system.

sent and the receiver will not learn more than k strings; this is known as a k -out-of- n oblivious transfer. Given a public key encryption system it is possible to construct a 1-out-of-2 oblivious transfer protocol [7], which is the building block used in Yao’s protocol.

Garbled Circuits The core of Yao’s protocol is the construction of garbled circuits, which involves encrypting the truth table of each gate in a circuit description of the function. When the protocol is run, the truth values in the circuit will be represented as decryption keys for some cipher, with each gate receiving a unique pair of keys for its output wire. The keys for a gate’s input wires are then used to encrypt the keys for its output wires. Given a single key for each input wire of the circuit, the party that evaluates the circuit can decrypt a single key that represents a hidden truth value for each gate’s output wire, until the output gates are reached. Since this encryption process can be applied to any circuit, and since any computable function has a corresponding circuit family, this allows the construction of a secure protocol for any computable function.

The typical garbled circuit protocol has two parties though it can be expanded to more. Those two parties are Bob, the generator of the garbled circuit, and Alice, the evaluator of the garbled circuit. Bob creates the garbled circuit and therefore knows the decryption keys, but does not know which specific keys Alice uses. Alice will receive the input keys from Bob using an oblivious transfer protocol, and thus learns only one key for each input wire; if the keys are generated independent of Bob’s input, Alice will learn only enough to compute the output of the circuit.

Several variations on the Yao protocol have been published; a simple description of the garbling and evaluation process follows. Let $f : \{0, 1\}^A \times \{0, 1\}^B \rightarrow \{0, 1\}^j \times \{0, 1\}^k$ be a computable function, which will receive input bits from two parties and produce output bits for each party (not necessarily the same outputs). To garble the circuit, a block cipher $\langle E, D, G \rangle$ will be used.

For each wire in the circuit, Bob computes a pair of random keys $(k_0, k_1) \leftarrow (G(1^n), G(1^n))$, which represent

logical 0 and 1 values. For each of Alice’s outputs, Bob uses these keys to encrypt a 0 and a 1 and sends the pair of ciphertexts to Alice. Bob records the keys corresponding to his own outputs. The rest of the wires in the circuit are inputs to gates. For each gate, if the truth table is $[v_{0,0}, v_{0,1}, v_{1,0}, v_{1,1}]$, Bob computes the following ciphertext:

$$\begin{bmatrix} E_{k_{l,0}}(E_{k_{r,0}}(k_{v_{0,0}})), E_{k_{l,0}}(E_{k_{r,1}}(k_{v_{0,1}})) \\ E_{k_{l,1}}(E_{k_{r,0}}(k_{v_{1,0}})), E_{k_{l,1}}(E_{k_{r,1}}(k_{v_{1,1}})) \end{bmatrix}$$

where $k_{l,*}$ and $k_{r,*}$ are the keys for the left and right input wires (this can be generalized for gates with more than two inputs). The order of the four ciphertexts is then randomly permuted and sent to Alice.

Now that Alice has the garbled gates, she can begin evaluating the circuit. Bob will send Alice his input wire keys. Alice and Bob then use an oblivious transfer to give Alice the keys for her input wires. For each gate, Alice will only be able to decrypt one entry, and will receive one key for the gate’s output, and will continue to decrypt truth table entries until the output wires have been computed. Alice will then send Bob his output keys, and decrypt her own outputs.

Optimizations Numerous optimizations to the basic Yao protocol have been published [10, 13, 17, 24, 27]. Of these, the most relevant to compiling circuits is the “free XOR trick” given by Kolesnikov and Schneider [17]. This technique allows XOR gates to be evaluated without the need to garble them, which greatly reduces the amount of data that must be transferred and the CPU time required for both the generator and the evaluator. One basic way to take advantage of this technique is to choose subcircuits with fewer non-XOR gates; Schneider published a list of XOR-optimal circuits for even three-input functions [27].

Huang et al. noted that there is no need for the evaluator to wait for the generator to garble all gates in the circuit [13]. Once a gate is garbled, it can be sent to the evaluator, allowing generation and evaluation to occur in parallel. This technique is very important for large circuits, which can quickly become too large to store in RAM [18]. Our approach unifies this technique with the use of an optimizing compiler.

3 Bytecode

A common approach to compiler design is to translate a high level language into a sequence of instructions for a simple, abstract machine architecture; this is known as the *intermediate representation* or *bytecode*. Bytecode representations have the advantage of being machine-independent, thus allowing a compiler front-end to be used for multiple target architectures. Optimizations per-

formed on bytecode are machine independent as well; for example, dead code elimination is typically performed on bytecode, as removing dead code causes programs to run faster on all realistic machines.

For the purposes of this work, we focus on a commonly used bytecode abstraction, the *stack machine*. In this model, operands must be pushed onto an abstract stack, and operations involve popping operands off of the stack and pushing the result. In addition to the stack, a stack machine has RAM, which is accessed by instructions that pop an address off the stack. Instructions in a stack machine are partially ordered, and are divided into subroutines in which there is a total ordering. In addition to simple operations and operations that interact with RAM, a stack machine has operations that can modify the *program counter*, a pointer to the next instruction to be executed, either conditionally or unconditionally.

At a high level, our system translates bytecode programs for a stack machine into boolean circuits for SFE. At first glance, this would appear to be at least highly inefficient, if not impossible, because of the many ways such an input program could loop. We show, however, that imposing only a small set of restrictions on permissible sequences of instructions enables an efficient and practical translator, without significantly reducing the usability or expressive power of the high level language.

4 System Design

Our system divides the compiler into several stages, following a common compiler design. For testing, we used the LCC compiler front end to parse C source code and produce a bytecode intermediate representation (IR). Our back end performs optimizations and translates the bytecode into a description of a secure computation protocol using our new format. This representation greatly reduces the disk space requirements for large circuits compared to previous work, while still allowing optimizations to be done at the bit level. We wrote our compiler in Common Lisp, using the Steel Bank Common Lisp system.

4.1 Compact Representations of Boolean Circuits

In Fairplay and the systems that followed its design, the common pattern has been to represent Boolean circuits as adjacency lists, with each node in the graph being a gate. This introduces a scalability problem, as it requires storage proportional to the size of the circuit. Generating, optimizing, and storing circuits has been a bottleneck for previous compilers, even for relatively simple functions like RSA. Loading such large circuits into RAM

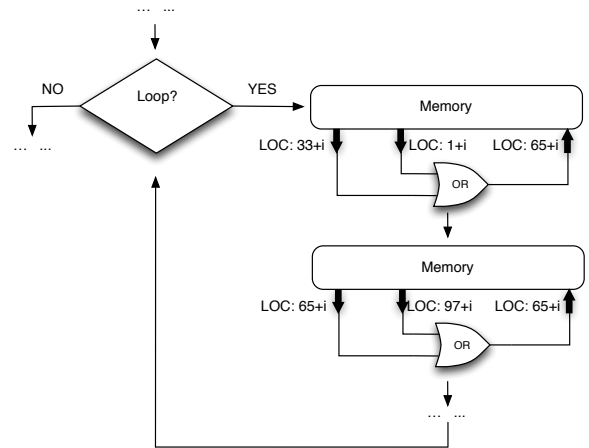


Figure 2: The high-level concept of the PCF design. It is not necessary to unroll loops at compile time, even to perform optimizations on the circuit. Instead, loops can be evaluated at runtime, with gates being computed on-the-fly, and loop indices being updated locally by each party. Wire values are stored in a table, with each gate specifying which two table entries should be used as inputs and where the output should be written; previous wire values in the table can be overwritten during this process, if they are no longer needed.

is a challenge, as even very high-end machines may not have enough RAM for relatively simple functions.

There have been some approaches to addressing this scalability problem presented in previous work. The KSS12 system reduced the RAM required for protocol executions by assigning each gate's output wire a reference count, allowing the memory used for a wire value to be deallocated once the gate is no longer needed. However, the compiler bottleneck was not solved in KSS12, as even computing the reference count required memory proportional to the size of the circuit. Even with the engineering improvements presented by Kreuter, Shelat, and Shen, the KSS12 compiler was unable to compile circuits with more than a few billion gates, and required several days to compile their largest test cases [18].

The PAL system [23] also addresses memory requirements, by adding control structures to the circuit description, allowing parts of the description to be re-used. In the original presentation of PAL, however, a large circuit file would still be emitted in the Fairplay format when the secure protocol was run. An extension of this work presented by Mood [22] allowed the PAL description to be used directly at runtime, but this work sacrificed the ability to optimize circuits automatically.

Our system builds upon the PAL and KSS12 systems to solve the memory scalability problem without sacri-

ficing the ability to optimize circuits automatically. Two observations are key to our approach.

Our first observation is that it is possible to free the memory required for storing wire values without computing a reference count for the wire. In previous work, each wire in a circuit is assigned a unique global identifier, and gate input wires are specified in terms of these identifiers (output wires can be identified by the position of the gate in the gate list). Rather than using global identifiers, we observe that wire values are ephemeral, and only require a unique identity until their last use as the input to a gate.

We therefore maintain a table of “active” wire values, similar to KSS12, but change the gate description. In this format, wire values are identified by their index in the table, and gates specify the index of each input wire and an index for the output wire; in other words, a gate is a tuple $\langle t, i_1, i_2, o \rangle$, where t is a truth table, i_1, i_2 are the input wire indexes, and o is the output wire index. When a wire value is no longer needed, its index in the table can be safely used as an output wire for a gate.

Now, consider the following example of a circuit described in the above format, which accumulates the Boolean AND of seven wire values:

```

 $\langle AND_1, 1, 2, 0 \rangle$ 
 $\langle AND_2, 0, 3, 0 \rangle$ 
 $\langle AND_3, 0, 4, 0 \rangle$ 
 $\langle AND_4, 0, 5, 0 \rangle$ 
 $\langle AND_5, 0, 6, 0 \rangle$ 
 $\langle AND_6, 0, 7, 0 \rangle$ 

```

Our second observation is that circuits such as this can be described more compactly using a loop. This builds on our first observation, which allows wire values to be overwritten once they are no longer needed. A simple approach to allowing this would add a conditional branch operation to the description format. This is more general than the approach of PAL, which includes loops but allows only simple iteration. Additionally, it is necessary to allow the loop index to be used to specify the input or output wire index of the gates; as a general solution, we add support for indirection, allowing wire values to be copied.

This representation of Boolean circuits is a bytecode for a one-bit CPU, where the operations are the 16 possible two-arity Boolean gates, a conditional branch, and indirect copy. In our system, we also add instructions for function calls (which need not be inlined at compile time) and handling the parties’ inputs/outputs. When the secure protocol is run, a three-level logic is used for wire values: 0, 1, or $\perp$, where $\perp$ represents an “unknown” value that depends on one of the party’s inputs. In the case of a Yao protocol, the $\perp$ value is represented by a

garbled wire value. Conditional branches are not allowed to depend on $\perp$ values, and indirection operations use a separate table of pointers that cannot be computed from $\perp$ values (if such an indirection operation is required, it must be translated into a large multiplexer, as in previous work).

We refer to our circuit representation as the *Portable Circuit Format* or PCF. In addition to gates and branches, PCF includes support for copying wires indirectly, a function call stack, data stacks, and setting function parameters. These additional operations do not emit any gates and can therefore be viewed as “free” operations. PCF is modeled after the concept of PAL, but instead of using predefined sub-circuits for complex operations, a PCF file defines the sub-circuits for a given function to allow for circuit structure optimization. PCF includes lower level control structures compared to PAL, which allows for more general loop structures.

In Appendix A, we describe in detail the semantics of the PCF instructions. Example PCF files are available at the authors’ website.

4.2 Describing Functions for SFE

Most commonly used programming languages can describe processes that cannot be translated to SFE; for example, a program that does not terminate, or one which terminates after reading a specific input pattern. It is therefore necessary to impose some limitation on the descriptions of functions for SFE. In systems with domain specific languages, these limitations can be imposed by the grammar of the language, or can be enforced by taking advantage of particular features of the grammar. However, one goal of our system is to allow any programming language to be used to describe functionality for SFE, and so we cannot rely on the grammar of the language being used.

We make a compromise when it comes to restricting the inputs to our system. Unlike model checking systems [2], we impose no upper bound on loop iterations or on recursive function calls (other than the memory available for the call stack), and leave the responsibility of ensuring that programs terminate to the user. On the other hand, our system does forbid certain easily-detectable conditions that could result in infinite loops, such as unconditional backwards jumps, conditional backwards jumps that depend on input, and indirect function calls. These restrictions are similar to those imposed by the Fairplay and KSS12 systems [18,21], but allow for more general iteration than incrementing the loop index by a constant. Although false positives, i.e., programs that terminate but which contain such constructs are possible, our hypothesis is that useful functions and typical compilers would not result in such instruction sequences, and

we observed no such functions in our experiments with LCC.

4.3 Algorithms for Translating Bytecode

Our compiler reads a bytecode representation of the function, which lacks the structure of higher-level descriptions and poses a unique challenge in circuit generation. As mentioned above, we do not impose any upper limit on loop iterations or the depth of the function call stack. Our approach to translation does not use any symbolic analysis of the function. Instead, we translate the bytecode into PCF, using conditional branches and function calls as needed and translating other instructions into lists of gates. For testing, we use the IR from the LCC compiler, which is based on the common stack machine model; we will use examples of this IR to illustrate our design, but note that none of our techniques strictly require a stack machine model or any particular features of the LCC bytecode.

In our compiler, we divide bytecode instructions into three classes:

Normal Instructions which have exactly one successor and which can be represented by a simple circuit. Examples of such instructions are arithmetic and bitwise logic operations, operations that push data onto the stack or move data to memory, etc.

Jump Instructions that result in an unconditional control flow switch to a specific label. This does not include function calls, which we represent directly in PCF. Such instructions are usually used for if/else constructs or preceding the entry to a loop.

Conditional Instructions that result in control flow switching to either a label or the subsequent instruction, depending on the result of some conditional statement. Examples include arithmetic comparisons.

In the stack machine model, all operands and the results of operations are pushed onto a global stack. For “normal” instructions, the translation procedure is straightforward: the operands are popped off the stack and assigned temporary wires, the subcircuit for the operation is connected to these wires, and the output of the operation is pushed onto the stack. “Jump” instructions appear, at first, to be equally straightforward, but actually require special care as we describe below.

“Conditional” instructions present a challenge. Conditional jumps whose targets precede the jump are assumed to be loop constructs, and are translated directly into PCF branch instructions. All other conditional jumps require the creation of multiplexers in the circuit to deal with

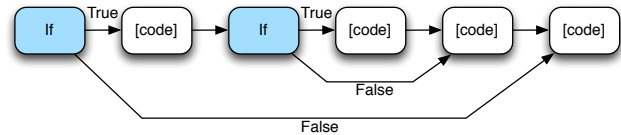


Figure 3: Nested if statements, which can be handled using the stack-based algorithm.

conditional assignments. Therefore, the branch targets must be tracked to ensure that the appropriate condition wires are used to control those multiplexers.

In the Fairplay and KSS12 compilers, the condition wire for an “if” statement is pushed onto a stack along with a “scope” that is used to track the values (wire assignments) of variables. When a conditional block is closed, the condition wire at the top of the stack is used to multiplex the value of all the variables in the scope at the top with the values from the scope second to the top, and then the stack is popped. This procedure relies on the grammar of “if/else” constructs, which ensures that conditional blocks can be arranged as a tree. An example of this type of “if/else” construct is in Figure 3. In a bytecode representation, however, it is possible for conditional blocks to “overlap” with each other without being nested.

In the sequence shown in Figure 4, the first branch’s target *precedes* the second branch’s target, and indirect loads and assignments exist in the overlapping region of these two branches. The control flow of such an overlap is given in Figure 5. A stack is no longer sufficient in this case, as the top of the stack will not correspond to the appropriate branch when the next branch target is encountered. Such instruction sequences are not uncommon in the code generated by production compilers, as they are a convenient way to generate code for “else” blocks and ternary operators.

To handle such sequences, we use a novel algorithm based on a priority queue rather than a stack, and we maintain a global condition wire that is modified as branches and branch targets are reached. When a branch instruction is reached, the global condition wire is updated by logically ANDing the branch condition with the global condition wire. The priority queue is updated with the branch condition and a scope, as in the stack-based algorithm; the priority is the target, with lower targets having higher priority. When an assignment is performed, the scope at the top of the priority queue is updated with the value being assigned, the location being assigned to, the old value, and a copy of the global condition wire. When a branch target is reached, multiplexers are emitted for each assignment recorded in the scope at the top of the priority queue, using the copy of the global condition wire that was recorded. After the

```

EQU4 A
INDIRI4 16
EQU4 B
INDIRI4 24
LABELV A
ASGNI4
LABELV B
ASGNI4

```

Figure 4: A bytecode sequence where overlapping conditional blocks are not nested; note that the target of the first branch, “A,” precedes the target of the second branch, “B.”

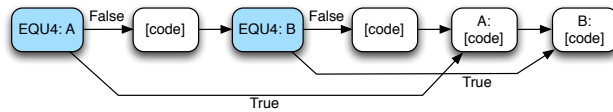


Figure 5: A control flow with overlapping conditional blocks.

multiplexers are emitted, the global condition wire is updated by ORing the *inverse* of the condition wire at the top of the priority queue, and then the top is removed.

Unconditional jumps are only allowed in the forward direction, i.e., only if the jump precedes its target. When such instructions are encountered, they are translated into conditional branches whose condition wire is the inverse of the conjunction of the condition wires of all enclosing branches. In the case of a jump that is not in any conditional block, the condition wire is set to false; this does not necessarily mean that subsequent assignments will not occur, as the multiplexers for these assignments will be emitted and will depend on a global control line that may be updated as part of a loop construct. The optimizer is responsible for determining whether such assignments can occur, and will rewrite the multiplexers as direct assignments when possible.

Finally, it is possible that the operand stack will have changed in the fall-through path of a conditional jump. In that case, the stack itself must be multiplexed. For simplicity, we require that the depth of the stack not change in a fall-through path. We did not observe any such changes to the stack in our experiments with LCC.

4.4 Optimization

One of the shortcomings of the KSS12 system was the amount of time and memory required to perform optimizations on the computed circuit. In our system, optimization is performed before loops are unrolled but after the functionality is translated into a PCF representation. This allows optimizations to be performed on a smaller

representation, but increases the complexity of the optimization process somewhat.

The KSS12 compiler bases its optimization on a rudimentary dataflow analysis, but without any conditional branches or loops, and with single assignments to each wire. In our system, loops are not eliminated and wires may be overwritten, but conditional branches are eliminated. As in KSS12, we use an approach based on dataflow analysis, but we must make multiple passes to find a fixed point solution to the dataflow equations. Our dataflow equations take advantage of the logical rules of each gate, allowing more gates to be identified for elimination than the textbook equations identify.

We perform our dataflow analysis on individual PCF instructions, which allows us to remove single gates even where entire bytecode instructions could not be removed, but which carries the cost of somewhat longer compilation time, on the order of minutes for the experiments we ran. Currently, our framework only performs optimization within individual functions, without any interprocedural analysis. Compile times in our system can be reduced by splitting a large procedure into several smaller procedures.

Optimization	128 mult.	5x5 matrix	256 RSA
None	707,244	260,000	904,171,008
Const. Prop.	296,960	198,000	651,504,495
Dead Elim.	700,096	255,875	883,307,712
Both	260,073	131,875	573,156,735

Table 1: Effects of constant propagation and dead code elimination on circuit size, measured with simulator that performs no simplification rules. For each function, the number of non-XOR gates are given for all combinations of optimizations enabled.

4.4.1 Constant Propagation

The constant propagation framework we use is straightforward, similar to the methods used in typical compilers. However, for some gates, simplification rules can result in constants being computed even when the inputs to a gate are not constant; for example, XORing a variable with itself. The transfer function we use is augmented with a check against logic simplification rules to account for this situation, but remains monotonic and so convergence is still guaranteed.

4.4.2 Dead Gate Removal

The last step of our optimizer is to remove gates whose output wires are never used. This is a standard bit vector dataflow problem that requires little tailoring for our system. As is common in compilers, performing this step

Function	With	Without	Ratio
16384-bit Comp.	32,228	49,314	65%
128-bit Sum	345	508	67%
256-bit Sum	721	1,016	70%
1024-bit Sum	2,977	4,064	73%
128-bit Mult.	76,574	260,073	20%
256-bit Mult.	300,634	1,032,416	20%
1024-bit Mult.	8,301,962	19,209,120	21%

Table 2: Non-XOR gates in circuits computed by the interpreter with and without the application of simplification rules by the runtime system.

last yields the best results, as large numbers of gates become dead following earlier optimizations.

4.5 Externally-Defined Functions

Some functionality is difficult to describe well in bytecode formats. For example, the graph isomorphism experiment presented in Section 6 uses AES as a PRNG building block, but the best known description of the AES S-box is given at the bit-level [4], whereas the smallest width operation supported by LCC is a single byte. To compensate for this difficulty, we allow users to specify functions with the same language used internally to translate bytecode operations into circuits; an example of this language is shown in Section 5.1. This allows for possible combinations of our compiler with other circuit generation and optimization tools.

4.6 PCF Interpreter

To use a PCF description of a circuit in a secure protocol, an interpreter is needed. The interpreter simulates the execution of the PCF file for a single-bit machine, emitting gates as needed for the protocol. Loops are not explicitly unrolled; instead, PCF branch instructions are conditionally followed, based on the logic value of some wire, and each wire identifier is treated as an address in memory. This is where the requirement that loop bounds be independent of both parties' inputs is ultimately enforced: the interpreter cannot determine whether or not to take a branch if it cannot determine the condition wire's value.

For testing purposes, we wrote two PCF interpreters: one in C, which is packaged as a reusable library, and one in Java that was used for tests on smartphones. The C library can be used as a simulator or for full protocol execution. As a simulator it simply evaluates the PCF file without any garbling to measure the size of the circuit that would have been garbled in a real protocol. This interpreter was used for the LAN tests, using an updated version of the KSS12 protocol. The Java interpreter was

Function	With (s)	Without (s)
16384-bit Comp.	$4.41 \pm 0.3\%$	$4.44 \pm 0.3\%$
128-bit Sum	$0.0581 \pm 0.3\%$	$0.060 \pm 2\%$
256-bit Sum	$0.103 \pm 0.3\%$	$0.105 \pm 0.3\%$
1024-bit Sum	$0.365 \pm 0.3\%$	$0.367 \pm 0.2\%$
128-bit Mult.	$0.892 \pm 0.1\%$	$0.894 \pm 0.1\%$
256-bit Mult.	$3.02 \pm 0.1\%$	$3.04 \pm 0.1\%$
1024-bit Mult.	$39.7 \pm 0.2\%$	$39.9 \pm 0.06\%$

Table 3: Simulator time with simplification rules versus without, using the C interpreter. Times are averaged over 50 samples, with 95% confidence intervals, measured using the *time* function implemented by SBCL.

incorporated into the HEKM system for the smartphone experiments, and can also be used in a simulator mode.

4.7 Threat Model

The PCF system treats the underlying secure computation protocol as a black box, without making any assumptions about the threat model. In Section 6, we present running times for smaller circuits in the malicious model version of the KSS12 protocol. This malicious model implementation simply invokes multiple copies of the same PCF interpreter used for the semi-honest version, one for each copy of the circuit needed in the protocol.

4.8 Runtime Optimization

Some optimizations cannot be performed without unrolling loops, and so we defer these optimizations until the PCF program is interpreted. As an example, logic simplification rules that eliminate gates whose output values depend on no more than one of their input wires can only be partially applied at compile time, as some potential applications of these rules might only be possible for some iterations of a loop. While it is possible to compute this information at compile time, in the general case this would involve storing information about each gate for every iteration of every loop, which would be as expensive as unrolling all loops at compile time.

A side effect of applying such logic simplification rules is copy propagation. A gate that always takes on the same value as one of its inputs is equivalent to a copy operation. The application of logic simplification rules to such a gate results in the interpreter simply copying the value of the input wire to the output wire, without emitting any gate. As there is little overhead resulting from the application of simplification rules at runtime, we are able to reduce compile times further by not performing this optimization at compile time.

Function	This Work	KSS12	HFKV
16384 Comp.	32,229	49,149	-
RSA 256	235,925,023	332,085,981	-
Hamming 160	880	-	3,003
Hamming 1600	9,625	-	30,318
3x3 Matrix	27,369	160,949	47,871
5x5 Matrix	127,225	746,177	221,625
8x8 Matrix	522,304	3,058,754	907,776
16x16 Matrix	4,186,368	24,502,530	7,262,208

Table 4: Comparisons between our compiler’s output and the output of the KSS12 and Holzer et al. (HFKV) compilers, in terms of non-XOR gates.

For each gate, the interpreter checks if the gate’s value can be statically determined, i.e., if its output value does not rely on either party’s input bits. This is critical, as some of the gates in a PCF file are used for control flow, e.g., to increment a loop index. Additionally, logic simplification rules are applied where possible in the interpreter. This allows the interpreter to not emit gates that follow an input or which have static outputs even when their inputs cannot be statically determined. As shown in Table 2, we observed cases where up to 80% of the gates could be removed in this manner. Even in a simulator that performs no garbling, applying this runtime optimization not only shows no performance overhead, but actually a very slight performance gain, as shown in Table 3. The slight performance gain is a result of the transfer of control that occurs when a gate is emitted, which has a small but non-trivial cost in the simulator. In a garbled circuit protocol, this cost would be even higher, because of the time spent garbling gates.

5 Portability

5.1 Portability Between Bytecodes

Our compiler can be given a description of how to translate bytecode instructions into boolean circuits using a special internal language. An example, for the LCC instruction “ADDU,” is shown in Figure 6. The first line is specific to LCC, and would need to be modified for use with other front-ends. The second line assumes a stack machine model: this instruction reads two instructions from the stack. Following that is the body of the translation rule, which can be used in general to describe circuit components and how the input variables should be connected to those components.

The description follows an abstraction similar to VM-Crypt, in which a unit gadget is “chained” to create a larger gadget. It is possible to create chains of chains, e.g., for a shift-and-add multiplier as well. For more complex operations, Lisp source code can be embedded,

```
(`ADDU' nil second normal nil nil
  (two-stack-arg (x y) (var var)
    (chain [o1 = i1 + i2 + i3,
      o2 = i1 + (i1 + i2) * (i1 + i3)]
    (o2 -> i3
      x -> i1
      y -> i2
      o1 -> stack)
    (0 -> i3))))
```

Figure 6: Code used in our compiler to map the bytecode instruction for unsigned integer addition to the subcircuit for that operation.

which can interact directly with the compiler’s internal data structures.

5.2 Portability Between SFE Systems

Both the PCF compiler and the interpreter can treat the underlying secure computation system as a black box. Switching between secure computation systems, therefore, requires work only at the “back end” of the interpreter, where gates are emitted. We envision two possible approaches to this, both of which we implemented for our tests:

1. A single function should be called when a gate should be used in the secure computation protocol. The Java implementation of PCF uses this approach, with the HEKM system.
2. Gates should be generated as if they are being read from a file, with the secure computation system calling a function. The secure computation system may need to provide “callback” functions to the PCF interpreter for copying protocol-specific data between wires. The C implementation we tested uses this abstraction for the KSS12 system.

6 Evaluation

We compiled a variety of functions to test our compiler, optimizer, and PCF interpreter. For each circuit, we tested the performance of the KSS12 system on a LAN, described below. For the KSS12 timings, we averaged the runtime for 50 runs, alternating which computer acted as the generator and which as the evaluator to account for slight configuration differences between the systems. Compiler timings are based on 50 runs of the compiler on a desktop PC with an Intel Xeon 5560 processor, 8GB of RAM, a 7200 RPM hard disk, Scientific Linux 6.3 (kernel version 2.6.32, SBCL version 1.0.38).

Function	Total Gates	non-XOR Gates	Compile Time (s)	Simulator Time (s)
16384-bit Comp.	97,733	32,229	3.40 ± 4%	4.40 ± 0.2%
Hamming 160	4,368	880	9.81 ± 1%	0.0810 ± 0.3%
Hamming 1600	32,912	6,375	11.0 ± 0.4%	0.52 ± 8%
Hamming 16000	389,312	97,175	10.8 ± 0.2%	4.83 ± 0.5%
128-bit Sum	1,443	345	4.70 ± 3%	0.0433 ± 0.4%
256-bit Sum	2,951	721	4.60 ± 3%	0.0732 ± 0.4%
1024-bit Sum	11,999	2,977	4.60 ± 3%	0.250 ± 0.5%
64-bit Mult.	105,880	24,766	71.7 ± 0.2%	0.332 ± 0.4%
128-bit Mult.	423,064	100,250	74.9 ± 0.1%	0.903 ± 0.3%
256-bit Mult.	1,659,808	400,210	79.5 ± 0.9%	3.07 ± 0.2%
1024-bit Mult.	25,592,368	6,371,746	74.0 ± 0.2%	40.9 ± 0.4%
256-bit RSA	673,105,990	235,925,023	381. ± 0.2%	980. ± 0.3%
512-bit RSA	5,397,821,470	1,916,813,808	350. ± 0.2%	7,330 ± 0.2%
1024-bit RSA	42,151,698,718	15,149,856,895	564. ± 0.2%	56,000 ± 0.3%
3x3 Matrix Mult.	92,961	27,369	306. ± 1%	0.256 ± 0.5%
5x5 Matrix Mult.	433,475	127,225	343. ± 0.7%	0.94 ± 2%
8x8 Matrix Mult.	1,782,656	522,304	109. ± 0.1%	3.14 ± 0.3%
16x16 Matrix Mult.	14,308,864	4,186,368	109. ± 0.1%	23.7 ± 0.3%
4-Node Graph Iso.	482,391	97,819	684. ± 0.2%	3.63 ± 0.5%
16-Node Graph Iso.	10,908,749	4,112,135	1040 ± 0.1%	47.0 ± 0.1%

Table 5: Summary of circuit sizes for various functions and the time required to compile and interpret the PCF files in a protocol simulator. Times are averaged over 50 samples, with 95% confidence intervals, except for RSA-1024 simulator time, which is averaged over 8 samples. Run times were measured using the *time* function implemented in SBCL.

Source code for our compiler, our test systems, and our test functions is available at the authors’ website.

6.1 Effect of Array Sizes on Timing

Some changes in compile time can be observed as some of the functions grow larger. The dataflow analysis deals with certain pointer operations by traversing the entire local variable space of the function and all global memory, which in functions with large local arrays or programs with large global arrays is costly as it increases the number of wires that optimizer must analyze. Reducing this cost is an ongoing engineering effort.

6.2 Experiments

We compiled and executed the circuits described below to evaluate our compiler and representation. Several of these circuits were tested in other systems; we present the non-XOR gate counts of the circuits generated by our compiler and other work in Table 4. The sizes, compile times, and interpreter times required for these circuits are listed in Table 5. By comparison, we show compile times and circuit sizes using the KSS12 and HFKV compilers in Table 6. As expected, the PCF compiler outperforms

these previous compilers as the size of the circuits grow, due to the improved scalability of the system.

Arbitrary-Width Millionaire’s Problem As a simple sanity check for our system, we tested an arbitrary-width function for the millionaire’s problem; this can be viewed as a string comparison function on 32 bit characters. It outputs a 1 to the party which has the larger input. We found that for this simple function, our performance was only slightly better than the performance of the KSS12 compiler on the same circuit.

Matrix Multiplication To compare our system with the work of Holzer et al. [12], we duplicated some of their experiments, beginning with matrix multiplication on 32-bit integers. We found that our system performed favorably, particularly due to the optimizations our compiler and PCF interpreter perform. On average, our system generated circuits that are 60% smaller. We tested matrices of 3x3, 5x5, 8x8, and 16x16, with 32 bit integer elements.

Hamming Distance Here, we duplicate the Hamming distance experiment from Holzer et al. [12]. Again, we found that our system generated substantially smaller circuits. We tested input sizes of 160, 1600, and 16000 bits.

Integer Sum We implemented a basic arbitrary-width integer addition function, using ripple-carry addition. No

Function	HFKV			KSS12		
	Total Gates	non-XOR gates	Time (s)	Total Gates	non-XOR gates	Time (s)
16384-bit Comp.	330,784	131,103	105. $\pm$ 0.1%	98,303	49,154	4.66 $\pm$ 0.5%
3x3 Matrix Mult.	172,315	47,871	2.2 $\pm$ 4%	424,748	160,949	10.5 $\pm$ 0.5%
5x5 Matrix Mult.	797,751	221,625	8.40 $\pm$ 0.3%	1,968,452	746,177	48.2 $\pm$ 0.2%
8x8 Matrix Mult.	3,267,585	907,776	59.4 $\pm$ 0.3%	8,067,458	3,058,754	210 $\pm$ 2%
16x16 Matrix Mult.	26,140,673	7,262,208	2,600 $\pm$ 7%	64,570,969	24,502,530	2,200 $\pm$ 1%
32-bit Mult.	65,121	26,624	6.43 $\pm$ 0.3%	15,935	5,983	0.55 $\pm$ 5%
64-bit Mult.	321,665	126,529	71.4 $\pm$ 0.3%	64,639	24,384	1.6 $\pm$ 2%
128-bit Mult.	1,409,025	546,182	999. $\pm$ 0.1%	260,351	97,663	6.10 $\pm$ 0.6%
256-bit Mult.	5,880,833	2,264,860	16,000 $\pm$ 2%	1,044,991	391,935	24.5 $\pm$ 0.2%
512-bit Mult.	-	-	-	4,187,135	1,570,303	105. $\pm$ 0.2%
1024-bit Mult.	-	-	-	16,763,518	6,286,335	430. $\pm$ 0.3%

Table 6: Times of HFKV and KSS12 compilers with circuit sizes. The Mult. program uses a Shift-Add implementation. All times are averaged over 50 samples with the exception of the HFKV 256-bit multiplication, which was run for 10 samples; times are given with 95% confidence intervals.

array references are needed, and so our compiler easily handles this function even for very large input sizes. We tested input sizes of 128, 256, and 1024 bits.

Integer Multiplication Building on the integer addition function, we tested an integer multiplication function that uses the textbook shift-and-add algorithm. Unlike the integer sum and hamming distance functions, the multiplication function requires arrays for both input and output, which slows the compiler down as the problem size grows. We tested bit sizes of 64, 128, 256, and 1024.

RSA (Modular Exponentiation) In the KSS12 system [18], it was possible to compile an RSA circuit for toy problem sizes, and it took over 24 hours to compile a circuit for 256-bit RSA. This lengthy compile time and large memory requirement stems from the fact that all loops are unrolled before any optimization is performed, resulting in a very large intermediate representation to be analyzed. As a demonstration of the improvement our approach represents, we compiled not only toy RSA sizes, but also an RSA-1024 circuit, using only modest computational resources. We tested bit sizes of 256, 512, and 1024.

Graph Isomorphism We created a program that allows two parties to jointly prove the zero knowledge proof of knowledge for graph isomorphism, first presented by Goldreich et al. [9]. In Goldreich et al.’s proof system, the prover has secret knowledge of an isomorphism between two graphs, g_1 and g_2 . To prove this, the prover sends the verifier a random graph g_3 that is isomorphic to g_1 and g_2 , and the verifier will then choose to learn either the $g_1 \rightarrow g_3$ isomorphism or the $g_2 \rightarrow g_3$ isomorphism. We modify this protocol so that Alice and Bob must jointly act as the prover; each is given shares of an isomorphism between graphs g_1 and g_2 , and will use the online protocol to compute g_3 and shares of the two isomorphisms.

Our implementation works as follows: the program takes in XOR shares of the isomorphism between g_1 and g_2 and a random seed from both participants. It also takes the adjacency matrix representation of g_1 as input by a single party. The program XORs the shares together to create the $g_1 \rightarrow g_2$ isomorphism. The program then creates a random isomorphism from $g_1 \rightarrow g_3$ using AES as the PRNG (to reduce the input sizes and thus the OT costs), which effectively also creates g_3 .

Once the random isomorphism $g_1 \rightarrow g_3$ is created, the original isomorphism, $g_1 \rightarrow g_2$, is inverted to get an isomorphism from $g_2 \rightarrow g_1$. Then the two isomorphisms are “followed” in a chain to get the g_2 to g_3 isomorphism, i.e., for the i^{th} instance in the isomorphic matrix, $iso_{2 \rightarrow 3}[i] = iso_{1 \rightarrow 3}[iso_{2 \rightarrow 1}[i]]$. The program outputs shares of both the isomorphism from g_1 to g_3 and the isomorphism from g_2 to g_3 to both parties.

An adjacency matrix of g_3 is also an output for the party which input the adjacency matrix g_1 . This is calculated by using g_1 and the $g_1 \rightarrow g_3$ isomorphism.

6.3 Online Running Times

To test the online performance of our new format, we modified the KSS12 protocol to use the PCF interpreter. Two sets of tests were run: one between two computers with similar specifications on the University of Virginia LAN, a busy 100 megabit Ethernet network, and one between two smartphones communicating over a wifi network.

For the LAN experiments, we used two computers running ScientificLinux 6.3, a four core Intel Xeon E5506 2.13GHz CPU, and 8GB of RAM. No time limit on computation was imposed on these machines, so we were able to run the RSA-1024 circuit, which requires a little less than two days. To compensate for slight con-

Function	CPU (s)	Network (s)	CPU (s)	Network (s)
	Generator		Evaluator	
16384-bit Comp.	$99.8 \pm 0.2\%$	$5.63 \pm 0.6\%$	$26.0 \pm 0.6\%$	$79.4 \pm 0.2\%$
Hamming 1600	$9.13 \pm 0.4\%$	$0.64 \pm 4\%$	$2.9 \pm 4\%$	$6.87 \pm 2\%$
Hamming 16000	$91.2 \pm 0.2\%$	$5.67 \pm 0.7\%$	$28. \pm 3\%$	$69. \pm 2\%$
64-bit Mult.	$0.749 \pm 0.3\%$	$0.158 \pm 0.7\%$	$0.409 \pm 0.3\%$	$0.494 \pm 0.6\%$
128-bit Mult.	$2.04 \pm 0.3\%$	$0.52 \pm 1\%$	$1.25 \pm 0.2\%$	$1.31 \pm 0.6\%$
256-bit Mult.	$5.74 \pm 0.5\%$	$1.2 \pm 2\%$	$4.2 \pm 2\%$	$2.7 \pm 3\%$
1024-bit Mult.	$72.7 \pm 0.2\%$	$28. \pm 4\%$	$60. \pm 2\%$	$40. \pm 3\%$
256-bit RSA	$1940 \pm 0.2\%$	$767. \pm 0.7\%$	$1620 \pm 2\%$	$1080 \pm 3\%$
1024-bit RSA	$1.15 \times 10^5 \pm 0.5\%$	$4.4 \times 10^4 \pm 4\%$	$9.5 \times 10^4 \pm 5\%$	$6.5 \times 10^4 \pm 7\%$
3x3 Matrix Mult.	$5.33 \pm 0.4\%$	$0.403 \pm 0.6\%$	$1.45 \pm 0.8\%$	$4.28 \pm 0.6\%$
5x5 Matrix Mult.	$24.4 \pm 0.2\%$	$1.81 \pm 0.4\%$	$6.75 \pm 0.9\%$	$19.5 \pm 0.4\%$
8x8 Matrix Mult.	$100. \pm 0.2\%$	$7.39 \pm 0.4\%$	$26.8 \pm 0.7\%$	$81.1 \pm 0.3\%$
4-node ISO	$10.1 \pm 0.1\%$	$1.05 \pm 0.7\%$	$4.96 \pm 0.3\%$	$6.15 \pm 0.4\%$
16-node ISO	$116. \pm 0.2\%$	$15.7 \pm 0.6\%$	$71.6 \pm 0.3\%$	$60.3 \pm 0.6\%$

Table 7: Total running time, including PCF operations and protocol operations such as oblivious transfer, for online protocols using the PCF interpreter and the KSS12 two party computation system, on two computers communicating over the University of Virginia LAN. With the exception of RSA-1024, all times are averaged over 50 samples; RSA-1024 is averaged over 8 samples. Running time is divided into time spent on computation and time spent on network operations (including blocking).

figuration differences between the two systems, we alternated between each machine acting as the generator and acting as the evaluator.

We give the results of this experiment in Table 7. We note that while the simulator times given in Table 5 are more than half the CPU time measured, they are also on par with the time spent waiting on the network. Non-blocking I/O or a background thread for the PCF interpreter may improve performance somewhat, which is an ongoing engineering task in our implementation.

6.4 Malicious Model Tests

The PCF system is not limited to the semi-honest model. We give preliminary results in the malicious model version of KSS12. These experiments were run on the same test systems as above, using two cores for each party. We present our results in Table 9. The increased running times are expected, as we used only two cores per party. In the case of 16384-bit comparison, the increase is very dramatic, due to the large amount of time spent on oblivious transfer (as both parties have long inputs).

6.5 Phone Execution

We created a PCF interpreter for use with the HEKM execution system and ported it to the Android environment. We then ran it on two Galaxy Nexus phones where one

phone was the generator and another phone was the evaluator. These phones have dual core 1.2Ghz processors and were linked over Wi-Fi using an Apple Airport.

6.6 Phone Trials

As seen in Table 8, we were able to run the smaller programs directly on two phones. Since the interpreter executes slower on a phone and what would have taken a week of LAN trials would have taken years of phone time, we did not complete trials of the larger programs. Not all of the programs had output for the generator, allowing the generator to finish before the evaluator. This leads to a noticeable difference in total running time between the two parties.

Mood’s work on designing SFE applications for mobile devices [22] found that allocation and deallocation was a bottleneck to circuit execution. This issue was addressed by substituting the standard *BigInteger* type for a custom class that reduced the amount of allocation required for numeric operations, resulting in a four-fold improvement in execution time. The lack of this optimization in our mobile phone experiments may contribute to the reduced performance that we observed.

In future work, we will port the C interpreter and KSS12 system to Android and run the experiment with that execution system. Since overhead appears to be tied to Android’s Dalvik Virtual Machine (DVM), running programs natively should reduce overhead and hence re-

Function	CPU (s)	Network (s)	CPU (s)	Network (s)
	Generator		Evaluator	
16384-bit Comp.	163. $\pm$ 0.5%	12. $\pm$ 3%	142. $\pm$ 0.5%	68. $\pm$ 1%
128-bit Sum	5.8 $\pm$ 8.2%	1. $\pm$ 30%	5.6 $\pm$ 8%	3. $\pm$ 20%
256-bit Sum	7.3 $\pm$ 5.0%	1. $\pm$ 30%	6. $\pm$ 5%	4. $\pm$ 20%
1024-bit Sum	16. $\pm$ 3.1%	2. $\pm$ 20%	16. $\pm$ 3%	6.4 $\pm$ 7%
64-bit Mult.	63.3 $\pm$ 0.5%	1. $\pm$ 10%	66.3 $\pm$ 0.6%	5. $\pm$ 10%
128-bit Mult.	257. $\pm$ 0.2%	3.8 $\pm$ 5%	280. $\pm$ 0.3%	12. $\pm$ 6%
3x3 Matrix Mult.	76.9 $\pm$ 0.4%	12. $\pm$ 2%	82.0 $\pm$ 0.5%	8.5 $\pm$ 4%
5x5 Matrix Mult.	352. $\pm$ 0.3%	49. $\pm$ 2%	371. $\pm$ 0.3%	32. $\pm$ 4%
8x8 Matrix Mult.	1,588. $\pm$ 0.1%	82. $\pm$ 3%	1,550. $\pm$ 0.1%	120. $\pm$ 1%

Table 8: Execution results from the phone interpreter using the HEKM execution system on two Galaxy Nexus phones. Times are averages of 50 samples, with 95% confidence intervals.

Function	CPU (s)	Network (s)	CPU (s)	Network (s)
	Generator		Evaluator	
16384-bit comp.	3900 $\pm$ 3%	76 $\pm$ 4%	2820 $\pm$ 2%	1200 $\pm$ 10%
128-bit sum	23. $\pm$ 2%	21 $\pm$ 2%	33.3 $\pm$ 0.5%	11.2 $\pm$ 0.2%
256-bit sum	63.0 $\pm$ 0.4%	10 $\pm$ 20%	49. $\pm$ 6%	27. $\pm$ 4%
1024-bit sum	260 $\pm$ 10%	16 $\pm$ 6%	187. $\pm$ 2%	100 $\pm$ 40%
128-bit mult.	192. $\pm$ 0.3%	47.2 $\pm$ 0.6%	168. $\pm$ 0.4%	70.1 $\pm$ 1%
256-bit mult.	637. $\pm$ 0.5%	160 $\pm$ 1%	577. $\pm$ 0.3%	210 $\pm$ 2%

Table 9: Online running time in the malicious model for several circuits. Times are averaged over 50 samples, with 95% confidence intervals.

duce the performance differential between the phone and PC environments. Additionally, the KSS12 system uses more efficient cryptographic primitives, potentially further improving performance.

7 Related Work

Compiler approaches to secure two-party computation have attracted significant attention in recent years. The TASTY system presented by Henecka et al. [11] combines garbled circuit approaches with homomorphic encryption, and includes a compiler that emits circuits that can be used in both models. As with Fairplay and KSS12, TASTY requires functions to be described in a domain-specific language. The TASTY compiler performs optimizations on the abstract syntax tree for the function being compiled. Kruger et al. developed an ordered BDD compiler to test the performance of their system relative to Fairplay [19]. Mood et al. focused on compiling secure functions on mobile devices with the PALC system, which involved a modification to the Fairplay compiler [23].

Recently, a compiler approach based on bounded model checking was present by Holzer et al. [12]. In that

work, the CBMC system [5] was used to construct circuits, which were then rewritten to have fewer non-XOR gates. This approach had several advantages over previous approaches, most prominent being that functions could be described in the widely used C programming language, and that the use of CBMC allows for more advanced software engineering techniques to be applied to secure computation protocols. Like KSS12, however, this approach unrolls all loops (up to some fixed number of iterations), and converts a high level description directly to a boolean circuit which must then be optimized.

In addition to SFE, work on efficient compilers for proof systems has also been presented. Almeida et al. developed a zero-knowledge proof of knowledge compiler for Σ -protocols, which converts a protocol specification given in a domain-specific language into a program for the prover and the verifier to run [1]. Setty et al. presented a system for verifiable computation that uses a modification of the Fairplay compiler, which computes a system of quadratic constraints instead of boolean circuits, and emits executables for the prover and verifier [28, 29]. Our system is somewhat similar to these approaches, in that the circuit representation we present can be viewed as a program that is executed by the par-

ties in the SFE system; however, our approach is unique in its handling of control flow and iterative constructs.

Closely related to our work is the Sharemind system [3, 14], which uses secure computation as a building block for privacy-preserving distributed applications. As in our approach, the circuits used in the secure computation portions of Sharemind are not fully unrolled until the protocol is actually run. Functions in Sharemind are described using a domain-specific language called SecreC. Although there has been work on static analysis for SecreC [26], the SecreC compiler does not perform automatic optimizations. By contrast, our approach is focused on allowing circuit optimizations at the bit-level to occur without having to unroll an entire circuit.

Kerschbaum has presented work on automatically optimizing secure computation at the protocol level, with an approach based on term and expression rewriting [15, 16]. This approach is based on maximizing the use of off-line computation by inferring what each party can compute without knowledge of the other party's input, and does not treat the underlying secure computation primitives as a black box. It therefore requires additional work to remain secure in the malicious model. Our techniques could conceivably be combined with Kerschbaum's to reduce the overhead of online components.

8 Future Work

Our compiler can conceivably read any bytecode representation as input; one immediate future direction is to write translations for the instructions of another bytecode format, such as LLVM or the JVM, which would allow functions to be expressed in a broader range of languages. Additionally, we believe that our techniques could be combined with Sharemind, by having our compiler read the bytecode for the Sharemind VM and compute optimized PCF files for cases where garbled circuit computations are used in a Sharemind protocol.

The PCF format does not convey high-level information about data operations or types. Such information may further reduce the size of the circuits that are computed. Static analysis of such information by compilers has been widely studied, and it is possible that our compiler could be extended to support further reductions in the sizes of circuits emitted by the PCF interpreter. High-level information about data structures could also be used to improve the generation of circuits prior to optimization, using techniques recently presented by Evans and Zahur [6].

Our system and techniques can likely be generalized to the multiparty case, and to other representations of functions, such as arithmetic circuits. This would require significant changes to the optimization strategies and goals in our compiler, but fewer changes would be necessary

for the PCF interpreter. Similar modifications to support homomorphic encryption systems are also possible.

9 Conclusion

We have presented an approach to compiling and storing circuits for secure computation systems that requires substantially lower computational resources than previous approaches. Empirical evidence of the improvement and utility of our approach is given, using a variety of functions with different circuit sizes and control flow structures. Additionally, we have presented a compiler for secure computation that reads bytecode as an input, rather than a domain-specific language, and have explored the challenges associated with such an approach. We also presented interpreters, which evaluate our new language on both PCs and phones.

The code for the compiler, PCF interpreters, and test cases will be available on the authors' website.

Acknowledgments We would like to thank Elaine Shi for her helpful advice. We also thank Chih-hao Shen for his help with porting KSS12 to use PCF. This material is based on research sponsored by the Defense Advanced Research Projects Agency (DARPA) and the Air Force Research Laboratory (AFRL) under contract FA8750-11-2-0211. The U.S. Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright notation thereon. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of DARPA or the U.S. Government.

References

- [1] J. B. Almeida, E. Bangerter, M. Barbosa, S. Krenn, A.-R. Sadeghi, and T. Schneider. A Certifying Compiler For Zero-Knowledge Proofs of Knowledge Based on Σ -Protocols. In *Proceedings of the 15th European conference on Research in computer security, ESORICS'10*, pages 151–167, Berlin, Heidelberg, 2010. Springer-Verlag.
- [2] A. Biere, A. Cimatti, E. M. Clarke, and Y. Zhu. Symbolic Model Checking without BDDs. In *Proceedings of the 5th International Conference on Tools and Algorithms for Construction and Analysis of Systems, TACAS '99*, pages 193–207, London, UK, UK, 1999. Springer-Verlag.
- [3] D. Bogdanov, S. Laur, and J. Willemson. Sharemind: A Framework for Fast Privacy-Preserving Computations. In *Proceedings of the 13th European Symposium on Research in Computer Security - ESORICS'08*, 2008.
- [4] J. Boyar and R. Peralta. A New Combinational Logic Minimization Technique with Applications to Cryptology. In P. Festa, editor, *Experimental Algorithms*, volume 6049 of *Lecture Notes in Computer Science*, pages 178–189. Springer Berlin / Heidelberg, 2010.

- [5] E. Clarke, D. Kroening, and F. Lerda. A Tool for Checking ANSI-C Programs. In K. Jensen and A. Podelski, editors, *Tools and Algorithms for the Construction and Analysis of Systems (TACAS 2004)*, volume 2988 of *Lecture Notes in Computer Science*, pages 168–176. Springer, 2004.
- [6] D. Evans and S. Zahur. Circuit structures for improving efficiency of security and privacy tools. In *IEEE Symposium on Security and Privacy (to appear)*, 2013.
- [7] S. Even, O. Goldreich, and A. Lempel. A randomized protocol for signing contracts. *Commun. ACM*, 28(6):637–647, June 1985.
- [8] C. W. Fraser and D. R. Hanson. *A Retargetable C Compiler: Design and Implementation*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1995.
- [9] O. Goldreich, S. Micali, and A. Wigderson. Proofs that yield nothing but their validity or all languages in np have zero-knowledge proof systems. *J. ACM*, 38(3):690–728, July 1991.
- [10] V. Goyal, P. Mohassel, and A. Smith. Efficient Two Party and Multi Party Computation Against Covert Adversaries. In *Proceedings of 27th annual international conference on Advances in cryptology, EUROCRYPT’08*, pages 289–306, Berlin, Heidelberg, 2008. Springer-Verlag.
- [11] W. Henecka, S. Kögl, A.-R. Sadeghi, T. Schneider, and I. Wehrenberg. TASTY: Tool for Automating Secure Two-party computations. In *ACM Conference on Computer and Communications Security*, 2010.
- [12] A. Holzer, M. Franz, S. Katzenbeisser, and H. Veith. Secure Two-Party computations in ANSI C. In *Proceedings of the 2012 ACM conference on Computer and communications security, CCS ’12*, pages 772–783, New York, NY, USA, 2012. ACM.
- [13] Y. Huang, D. Evans, J. Katz, and L. Malka. Faster Secure Two-Party Computation Using Garbled Circuits. In *USENIX Security Symposium*, 2011.
- [14] R. Jagomägis. SecreC: a Privacy-Aware Programming Language with Applications in Data Mining. Master’s thesis, University of Tartu, 2010.
- [15] F. Kerschbaum. Automatically optimizing secure computation. In *Proceedings of the 18th ACM conference on Computer and communications security, CCS ’11*, pages 703–714, New York, NY, USA, 2011. ACM.
- [16] F. Kerschbaum. Expression rewriting for optimizing secure computation. In *Conference on Data and Application Security and Privacy*, 2013.
- [17] V. Kolesnikov and T. Schneider. Improved Garbled Circuit: Free XOR Gates and Applications. In L. Aceto, I. Damgård, L. Goldberg, M. Halldórsson, A. Ingólfssdóttir, and I. Walukiewicz, editors, *ALP 2008*, volume 5126 of *LNCS*, pages 486–498. Springer, 2008.
- [18] B. Kreuter, A. Shelat, and C.-H. Shen. Billion-gate secure computation with malicious adversaries. In *Proceedings of the 21st USENIX conference on Security symposium, Security’12*, pages 14–14, Berkeley, CA, USA, 2012. USENIX Association.
- [19] L. Kruger, S. Jha, E.-J. Goh, and D. Boneh. Secure function evaluation with ordered binary decision diagrams. In *Proceedings of the 13th ACM conference on Computer and communications security (CCS’06)*, Alexandria, VA, Oct. 2006.
- [20] L. Malka. VMCrypt: modular software architecture for scalable secure computation. In *ACM Conference on Computer and Communications Security*, pages 715–724, 2011.
- [21] D. Malkhi, N. Nisan, B. Pinkas, and Y. Sella. Fairplay: A Secure Two-Party Computation System. In *13th Conference on USENIX Security Symposium*, volume 13, pages 287–302. USENIX Association, 2004.
- [22] B. Mood. Optimizing Secure Function Evaluation on Mobile Devices. Master’s thesis, 2012, University of Oregon.
- [23] B. Mood, L. Letaw, and K. Butler. Memory-Efficient Garbled Circuit Generation for Mobile Devices. In *Financial Cryptography and Data Security*, volume 7397. Springer Berlin Heidelberg, 2012.
- [24] B. Pinkas, T. Schneider, N. Smart, and S. Williams. Secure Two-Party Computation Is Practical. In M. Matsui, editor, *Asiacrypt*, volume 5912 of *LNCS*, pages 250–267. Springer, 2009.
- [25] M. Rabin. How to Exchange Secrets by Oblivious Transfer. Technical Report TR-81, Harvard Aiken Computation Laboratory, 1981.
- [26] J. Ristioja. An analysis framework for an imperative privacy-preserving programming language. Master’s thesis, Institute of Computer Science, University of Tartu, 2010.
- [27] T. Schneider. *Engineering Secure Two-Party Computation Protocols - Design, Optimization, and Applications of Efficient Secure Function Evaluation*. Springer, 2012.
- [28] S. Setty, R. McPherson, A. J. Blumberg, and M. Walfish. Making Argument Systems for Outsourced Computation Practical (Sometimes). In *NDSS*, 2012.
- [29] S. Setty, V. Vu, N. Panpalia, B. Braun, A. J. Blumberg, and M. Walfish. Taking proof-based verified computation a few steps closer to practicality. In *Proceedings of the 21st USENIX conference on Security symposium*, Berkeley, CA, USA, 2012.
- [30] A. Yao. Protocols for Secure Computations. In *23rd Symposium on Foundations of Computer Science*, pages 160–164. IEEE Computer Society, 1982.

A PCF Semantics

The PCF file format consists of a header section that declares the input size, followed by a list of operations that are divided into subroutines. At runtime, these operations manipulate the internal state of the PCF interpreter, causing gates to be emitted when necessary. The internal state of the PCF interpreter consists of an instruction pointer, a call stack, an array of wire values, and an array of pointers. The pointers are positive integers. Wire values are 0, 1, or $\perp$, where $\perp$ represents a value that depends on input data, which is supplied by the code that invokes the interpreter. Each position in the wire table can be treated as a stack.

Each PCF instruction can take up to 3 arguments. The instructions and their semantics are as follows:

CLABEL/SETLABELC Appears only in the header, used for setting the input size for each party. CLABEL declares the bit width of a value, SETLABELC sets the value.

FUNCTION Denotes the beginning of a subroutine. When the subroutine is called, the instruction pointer is set to the position following this instruction.

GADGET Denotes a branch target

BRANCH Takes two arguments: a target, declared with **GADGET**, and a location in the wire table. In the wire value is 0, the instruction pointer is set to the instruction following the target. If the wire value is 1, the instruction pointer is incremented. If the wire value is $\perp$, evaluation halts with an error.

FUNC Calls a subroutine, pushing the current instruction pointer onto the call stack.

PUSH Pushes a copy of the wire value at a specified position onto the stack at that position.

POP Pops a stack at a specified position. If there is only one value on that stack, evaluation halts with an error.

ALICEIN32/BOBIN32 Fetches 32 input bits from one party, beginning at a specified *bit* position in that party's input. The bit position is specified by an array of 32 values in the wire table. If any of the values is $\perp$, evaluation halts with an error. The input values will all have the value $\perp$, and will be stored in the wire table at positions 0 through 31.

SHIFT OUT Outputs a single bit for a given party

RETURN Return from a subroutine. The instruction pointer is repositioned to the value popped from the top of the call stack.

STORECONSTPTR Sets a value in the pointer table

OFFSETPTR Adds a value to a pointer, specified by an array of 32 wire values starting at a position in the wire table. If any value in the array is $\perp$, evaluation halts with an error.

PTRTOWIRE Saves a pointer value as a 32 bit unsigned integer. Each of the bits is pushed onto the stack at a location in the wire table.

PTRTOPTR Copies a value from one position in the pointer table to another.

CPY121 Copy a wire value from a position specified by a pointer to a statically specified position.

CPY32 Copy a wire value from a statically specific position to a position specified by a pointer.

80,080,181,081,1 Compute a gate with the specified truth table on two input values from the wire table, with output stored at a specified position. Logic simplification rules are applied when one or both of the input values is $\perp$. If no simplification is possible, then the output will be $\perp$ and the interpreter will emit a gate. This is used for both local computations such as updating a loop index, and for computing the gates used by the protocol.

A.1 Example PCF Description

Below is an example of a PCF file. It iterates over a loop several times times, XORing the two parties' inputs with a bit from the internal state.

```
GADGET: main
CLABEL ALICEINLENGTH 32
CLABEL BOBINLEGNTN 32
CLABEL xxx 32
SETLABELC ALICEINLENGTH 128
SETLABELC ALICEINLENGTH 128
FUNCTION: main
1111 32 0 0
0000 33 0 0
0000 34 0 0
0000 35 0 0
GADGET: L
0110 36 35 34
0001 35 36 36
0110 36 34 33
0001 34 36 36
0110 36 33 32
0001 33 36 36
ALICEINPUT32 0 0
0001 36 0 0
BOBINPUT32 0 0
0001 37 0 0
0110 38 37 36
0110 39 33 38
SHIFT OUT ALICE 39
BRANCH L 35
RETURN xxx
```

Machine-Generated Algorithms, Proofs and Software for the Batch Verification of Digital Signature Schemes

Joseph Ayo Akinyele Matthew Green Susan Hohenberger Matthew Pagano
Johns Hopkins University

Abstract—¹

Digital signatures provide assurance and trust in almost all aspects of today's electronic communications from SSL certificates to software signing to email. For applications that process many signed messages at once, batch verification is a tempting way to increase throughput, since it is a method for saving on computation time by processing many signatures together.

This is a challenge, because human design error has been a problem historically in batch verification and yet it is desirable to have a large set of signature schemes with their batch verifiers to choose from to take advantage of various features and as a hedge against security flaws.

We address this by presenting AutoBatch, an automated tool for generating batch verification code from the code of a signature scheme. While prior works suggested generic techniques for batch verification, to our knowledge, this is the first work to systematically identify when these techniques are applicable and apply them. Moreover, we argue that this process preserves the unforgeability of the original scheme. AutoBatch can handle any pairing-based signature scheme, including variants of signatures, such as identity-based and ring.

The techniques behind AutoBatch and its implementation are described herein with performance measurements for the output of AutoBatch on several existing signature schemes. The conversion process executes quickly and the resulting batch savings is significant.

We believe that AutoBatch is a valuable cryptographic design, proof and implementation tool and, moreover, that it opens up a new direction in the larger landscape of computer-aided cryptography.

I. INTRODUCTION

Digital signatures are fundamental to establishing trust in today's electronic communications. For applications where devices are asked to process many certificates and signed messages, several prior works focused on batch verification. In batch verification, a batch of signatures are processed together according to a

special algorithm to save on the overhead. It is desirable to have batch verifiers for many signature schemes, to take advantage of their various properties and as a hedge against a security flaw being found in any one scheme. Unfortunately, the lesson of the past fifteen years is that designing batch verifiers is hard and error prone.

For instance, in 1994, an interactive batch verifier for DSA presented in an early version of [32] was broken by Lim and Lee [28]. In 1995 Lai and Yen proposed a new method for batch verification of DSA and RSA signatures [25], but the RSA batch verifier was broken five years later by Boyd and Pavlovski [8]. In 1998, two batch verification techniques were presented for DSA and RSA [19], [20] but both were later broken [8], [23], [24]. The same year, Bellare, Garay and Rabin took the first systematic look at batch verification [4] and presented three generic methods for batching modular exponentiations, one of which was called the *small exponents test*. Unfortunately, in 2000, Boyd and Pavlovski [8] published attacks against various batching schemes which were using the small exponents test incorrectly. In 2003 and 2004, several batch verification schemes based on bilinear maps (a.k.a., pairings) were proposed [12], [37], [38], [39] but all were later broken by Cao, Lin and Xue [11]. In 2006, a method was proposed for identifying invalid signatures in RSA-type batch signatures [27], but it was also flawed [35].

These examples highlight that the design of batch verifiers can be challenging and that human error has historically been a problem in this area. Even when general frameworks for designing schemes have been made available [4], they have often been misapplied [8].

A. Our Contributions

We address this by presenting AutoBatch, an automated tool that transforms a digital signature scheme into an optimized batch verification program. To our knowledge, this is the first such attempt to remove the human element, as much as possible, from the batch verification *design and implementation* process. The algorithm behind AutoBatch is a combination of batching techniques, including the small exponents test

¹This document is a special courtesy copy created for the DARPA PROCEED administrators of a work in progress which was partially funded by the Defense Advanced Research Projects Agency (DARPA) and the Air Force Research Laboratory (AFRL) under contract FA8750-11-2-0211. It is not intended for public distribution. Applying to all authors, the views expressed are those of the authors and do not reflect the official policy or position of the Department of Defense or the U.S. Government.

from [4], the bilinear equation substitutions in [15], the testing of bilinear group membership and a divide-and-conquer approach to finding invalid signatures. While the majority of these ideas are drawn from prior work, unlike prior work, we are able to automatically identify when they are applicable and automatically apply them to the verification equation in a consistent and secure way. We also introduce new logic for altering the behavior of the batching algorithm based on its input size or past input.

Importantly, the way in which we combine these techniques and optimizations preserves the unforgeability of the original scheme. Specifically, with all but a negligible probability, the batch verifier will accept a batch S of signatures if and only if every $s \in S$ would have been accepted by the individual verification algorithm. To provide transparency, AutoBatch also produces a machine-generated PDF document written in LaTeX that specifies each technique applied and provides a proof that security holds.

AutoBatch can handle various flavors of signatures, as we illustrate with tests on identity-based and ring signatures. AutoBatch exclusively accepts pairing-based schemes. Given that the pairing setting offers low bandwidth solutions, this is consistent with our overall goal of keeping cryptographic overhead low.

We believe AutoBatch is a tool with many applications, assisting both scheme designers and practitioners. It removes the human element from a cryptographic design task that appears to be easier for machines to navigate. It concentrates what human experts must verify to AutoBatch itself and its proof of security. Moreover, it makes meaningful progress toward the larger goal of reducing security errors through computer-aided design and implementation tools.

B. Overview of Our Approach

We present a detailed explanation of AutoBatch in §III. In this section and in Figure 1 we provide a brief overview of the techniques. At a high level, AutoBatch is designed to analyze a scheme, extract the signature verification equation, and derive working code for a batch verifier. This involves three distinct components:

- 1) A Code Parser, which retrieves the verification equation and variable types from some existing scheme implementation. This process naturally assumes that the scheme has been implemented within certain constraints, which we discuss later in the paper. Given such an implementation, the Parser obtains the signature verification equation and encodes it into an intermediate representation called *Scheme Description Language* (SDL).
- 2) A Batcher, which takes as input an SDL file describing a signature verification equation. The Batchers applies a series of rules to optimize the equation and thus derive a new equation for a batch verifier. The output of this equation is second SDL file containing the individual and batch equations, along with an analysis of the batcher's estimated running time. The Batchers optionally outputs human-readable LaTeX file containing a security proof for the batch verifier.
- 3) A Code Generator, which takes the output of the Batchers and generates working source code to implement the batch verifier. Beyond simply implementing the verification equation, the Generator adds a series of additional components, including group element membership checks, a recursive divide-and-conquer process to handle batches that contain *invalid* signatures, and additional logic to identify cases where individual verification is likely to outperform batch verification.

There are two usage scenarios for AutoBatch. In the first case, a user already has a working implementation of a (non-batched) signature scheme, and proceeds via the steps above. However, if the user does *not* have a working implementation, a second usage of AutoBatch allows the user to skip the Code Parsing phase of the process, and begin with a hand-coded SDL file. Since SDL files are human-readable ASCII-based files containing a mathematical representation of the scheme, some developers may prefer to implement new schemes directly in this language, which is agnostic to the final programming language that will be used for the implementation.

Although the techniques behind Autobatch can be applied to many languages and development environments, our implementation is based on Charm [1]. Charm is a Python-based rapid prototyping framework created by Akinyele, Green and Rubin that provides infrastructure for developing advanced cryptographic schemes. Charm implements all high performance operations (*e.g.*, bilinear group operations) in native C code using libraries such as MIRACL [33]. By using Charm we are able to obtain many of the performance advantages of C code, while taking advantage of the features offered by an interpreted language. One such advantage is that AutoBatch verifiers may be generated at runtime and executed dynamically when they are needed by an application.

C. Related Work

Computer-aided tools to assist cryptographers has long been a goal of high importance. Recently, the best

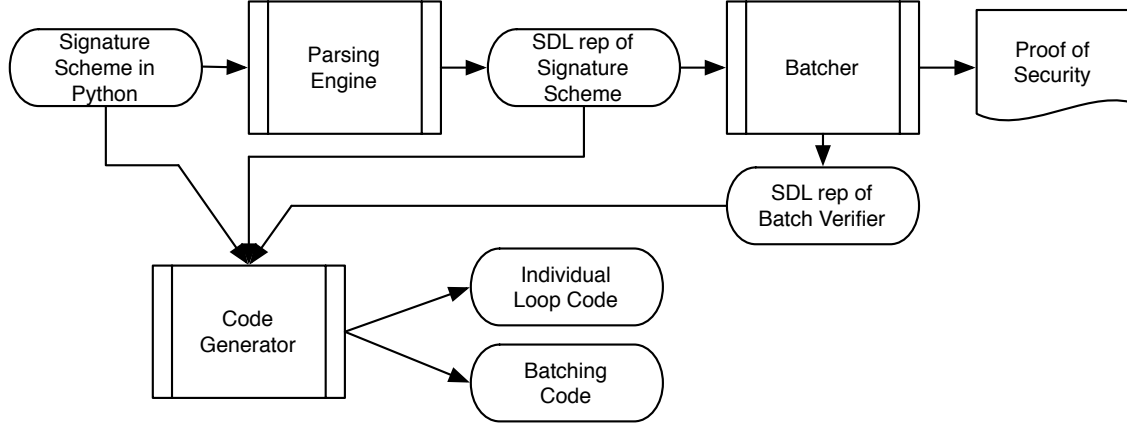


Figure 1. The flow of AutoBatch is illustrated above. The input is a signature scheme, comprised of key generation, signing and verification algorithms, coded in Python. The Parsing Engine extracts from this code the verification equations and classifies each element involved, e.g., an element of the public parameters, public key, signature or message space. This output is passed in an SDL representation of the signature scheme to the Batcher. The Batcher applies the techniques and optimizations from Section III to produce an SDL representation of a batch verification algorithm together with some performance estimates. It also outputs a proof of security (as a PDF written in LaTeX) that explains, line by line, each technique applied and its security justification. Finally, the Code Generator produces executable Python code implementing the batching verifier, as well as a loop over the individual verification algorithm to use as a comparison. The Code Generator can be set with different flags to offer further optimize the batching code, such as beginning a few levels into the recursive divide-and-conquer procedure when expecting batches with a moderate number of invalid signatures.

paper award at CRYPTO 2011 was given to Barthe, Grégoire, Heraud and Zanella Béguelin [3] for their invention of EasyCrypt, an automated tool for generating security proofs of cryptographic system from proof sketches. The reader is referred to this work for a summary of prior efforts to automate the verification of cryptographic security proofs.

In 1989, batch cryptography was introduced by Fiat [16] for a variant of RSA. In addition to the batching work mentioned before, we provide a few notes. Shacham and Boneh presented a modified version of Fiat’s batch verifier for RSA to improve the efficiency of SSL handshakes on a busy server [34]. Shacham, Lynn and Boneh provided a single-signer batch verifier for their BLS signatures [6]. Camenisch, Hohenberger and Pedersen [10] gave multiple-signer batch verifiers for the Waters identity-based signatures [36] and a novel construction. Ferrara, Green, Hohenberger, and Pedersen outlined some techniques for batching pairing-based signatures and also showed how to batch verify other types of signatures, such as group and ring signatures [15]. Blazy, Fuchsbaauer, Izabachéne, Jambert, Sibert and Vergnaud [5] applied batch verification techniques to the Groth-Sahai zero-knowledge proof system as well as group signatures and anonymous credential systems relying on them, obtaining significant savings.

Law and Matt describe methods for identifying invalid signatures in a batch [26], [29], [30].

II. BACKGROUND

A. Signatures

Definition 2.1 (A Digital Signature Scheme): A digital signature scheme is a tuple of probabilistic polynomial-time algorithms (Gen, Sign, Verify) as:

- 1) $\text{Gen}(1^\lambda) \rightarrow (pk, sk)$: the key generation algorithm takes as input the security parameter 1^λ and outputs a pair of keys (pk, sk) .
- 2) $\text{Sign}(sk, m) \rightarrow \sigma$: the signing algorithm takes as input a secret key sk and a message m from the message space and outputs a signature σ .
- 3) $\text{Verify}(pk, m, \sigma) \rightarrow \{0, 1\}$: the verification algorithm takes as input a public key pk , a message m and a purported signature σ , and outputs a bit indicating the validity of the signature.

A scheme is *correct* if for all $\text{Gen}(1^\ell) \rightarrow (pk, sk)$, the algorithm $\text{Verify}(pk, m, \text{Sign}(sk, m)) = 1$.

Goldwasser, Micali and Rivest [17] defined a scheme to be *unforgeable* as follows: Let $\text{Gen}(1^\ell) \rightarrow (pk, sk)$. Suppose (m, σ) is output by a probabilistic polynomial-time adversary with access to a signing oracle $\mathcal{O}_{sk}(\cdot)$ and input pk . Then the probability that m was *not* queried to $\mathcal{O}_{sk}(\cdot)$ and yet $\text{Verify}(pk, m, \sigma) = 1$ is negligible in ℓ .

In this work, we explore signature schemes with two additional properties, which we informally review:

- 1) **Identity-Based Signatures:** The Gen algorithm is executed by a master authority who publishes

pk and uses sk to generate signing keys for users according to their public identity string, such as an email address. To verify a signature on a given message, one only needs to have the pk of the master authority and the public identity string of the purported signer.

- 2) **Ring Signatures:** The signature is associated with a group of users, where verification shows that at least one member of the group signed the message, but it is difficult to tell whom.

B. Batch Verification

In this paper, we will not be concerning ourselves much with the traditional definitions of unforgeability. Rather we are interested in designing batch verification algorithms that accept a set of signatures *if and only if* each signature would have been accepted by its verification algorithm individually. *Thus, if a scheme is unforgeable, then our batching algorithm will preserve this property.* No more, no less.

Specifically, we consider the case where we want to quickly verify a set of signatures on possibly different messages by possibly different signers. The input is $\{(t_1, m_1, \sigma_1), \dots, (t_n, m_n, \sigma_n)\}$, where t_i specifies the verification key against which σ_i is purported to be a signature on message m_i . It is important to understand that here one or more *signers* may be maliciously colluding against the batch verifier.

We recall the definition of Bellare, Garay and Rabin [4] as extended by Camenisch, Hohenberger and Pedersen [10] to deal with multiple signers.

Definition 2.2 (Batch Verification of Signatures):

Let ℓ be the security parameter. Suppose $(\text{Gen}, \text{Sign}, \text{Verify})$ is a signature scheme, $k, n \in \text{poly}(\ell)$, and $(pk_1, sk_1), \dots, (pk_k, sk_k)$ are generated independently according to $\text{Gen}(1^\ell)$. Let $PK = \{pk_1, \dots, pk_k\}$. Then we call probabilistic Batch a batch verification algorithm when the following conditions hold:

- If $pk_{t_i} \in PK$ and $\text{Verify}(pk_{t_i}, m_i, \sigma_i) = 1$ for all $i \in [1, n]$, then $\text{Batch}((pk_{t_1}, m_1, \sigma_1), \dots, (pk_{t_n}, m_n, \sigma_n)) = 1$.
- If $pk_{t_i} \in PK$ for all $i \in [1, n]$ and $\text{Verify}(pk_{t_j}, m_j, \sigma_j) = 0$ for some $j \in [1, n]$, then $\text{Batch}((pk_{t_1}, m_1, \sigma_1), \dots, (pk_{t_n}, m_n, \sigma_n)) = 0$ except with probability negligible in ℓ , taken over the randomness of Batch.

Definition B.1 generalizes beyond signatures to apply to any keyed scheme, or combination of keyed schemes, with a verification algorithm. This includes zero-knowledge proofs, verifiable random functions,

and variants of regular signatures, such as identity-based, attribute-based, ring, group, aggregate, etc. Indeed, when the individual verification procedure involves evaluating a single pairing equation, then applying the small exponents test with security parameter λ to the product of these equations results in a **pairing-based batch verifier** that accepts an invalid batch with probability at most $2^{-\lambda}$ [15].

The above definition requires that signing keys be generated honestly. In practice, users could register their keys and prove some necessary properties of the keys at registration time [2].

C. Algebraic Setting and Group Membership

Bilinear Groups: We recall some of the basics of our algebraic setting as discussed in [10]. Let BSetup be an algorithm that, on input the security parameter 1^ℓ , outputs the parameters for a bilinear map (also called a pairing) as $(q, g, \mathbb{G}, \mathbb{G}_T, e)$, where $\mathbb{G}$ and $\mathbb{G}_T$ are groups of prime order $q \in \Theta(2^\ell)$. The efficient mapping $e : \mathbb{G} \times \mathbb{G} \rightarrow \mathbb{G}_T$ is both: (*bilinear*) for all $g \in \mathbb{G}$ and $a, b \leftarrow \mathbb{Z}_q$, $e(g^a, g^b) = e(g, g)^{ab}$; and (*non-degenerate*) if g generates $\mathbb{G}$, then $e(g, g) \neq 1$.

The above bilinear map is called a *symmetric bilinear map*. A more general version of the bilinear map is the *asymmetric bilinear map* $e : \mathbb{G}_1 \times \mathbb{G}_2 \rightarrow \mathbb{G}_T$, where $\mathbb{G}_1$ and $\mathbb{G}_2$ are distinct groups, possibly without efficient isomorphisms between them. We use this setting in our implementations because it allows for the most compact representations in $\mathbb{G}_1$, as we explain in Section IV.

Testing Membership in Bilinear Groups: When batching a group of signatures, it is critical to test that the elements of each signature are members of the appropriate algebraic group. Indeed, Boyd and Pavlovski [8] demonstrated efficient attacks on batching algorithms for DSA signature verification which omitted a subgroup membership test.

In this paper, we must test membership in bilinear groups. We require that elements of purported signatures are members of $\mathbb{G}$ and *not*, say, members of $E(\mathbb{F}_p) \setminus \mathbb{G}$. Determining whether some data represents a point on a curve is easy. The question is whether it is in the correct subgroup. Assume we have a bilinear map $e : \mathbb{G}_1 \times \mathbb{G}_2 \rightarrow \mathbb{G}_T$ and want to test membership in $\mathbb{G}_1$. ($\mathbb{G}_1$ will contain the smallest group elements, so signature elements will almost exclusively come from $\mathbb{G}_1$.)

If the order of $\mathbb{G}_1$ is a prime q , one option is to verify that an element y is in $\mathbb{G}_1$ by checking that $y^q \bmod q = 1$ [10]. While one might worry that this extra work diminishes the batching savings, it is not a problem in practice for pairing-based schemes, since the cost for a single exponentiation is considerably less than

the cost for computing a pairing. This has been verified experimentally before by Ferrara et al. [15] and our tests confirm this.

III. THE TECHNIQUES BEHIND AUTOBATCH

In this section we summarize the techniques used to programatically generate batch verifiers from a standard signature implementation. A highlevel abstraction was provided in Figure 1. We now give the details. The AutoBatch toolchain begins with a Charm-Python implementation of a signature, and then proceeds as:

1. Parse the program to extract the verification equation. The first phase of the toolchain analyzes the implemented scheme to extract the signature verification equation² as well as the datatypes of the parameter and signature elements. An advantage of using Python for our implementations is that our tools may leverage various capabilities that are built into Python. Specifically, we use introspection to obtain variable types, and we traverse the Python Abstract Syntax Tree (AST) to parse the verification equation. The output of this process is an intermediate representation of the signature verification equation in Scheme Description Language (SDL). An example Python input and its corresponding SDL output is presented in Figure 2.

2. Apply batching techniques and optimize the verification equation. We next apply a set of techniques designed to convert the extracted signature verification equation into a batch verifier. These tools drawn from the summary in [15] re-arrange the verification equation by combining pairing equations and re-arranging the components to minimize the number of expensive pairing operations. To prevent known attacks, we apply the small exponents test of Bellare, Garay and Rabin [4], and optimize the resulting equation to ensure that all signature elements are in the smallest group (typically, $\mathbb{G}_1$). The output of this phase is a modified SDL file, and (optionally) a human-readable proof that the resulting equation is a batch verifier.

3. Evaluate the capabilities of the batch verifier. Given the optimized batching equation produced in the previous step, we estimate the performance of the verifier under various conditions. This is done by counting the operations in the verifier, and deriving a

²To be clear, our Parser only handles extraction for schemes with a single verification equation. This is only a gentle restriction for two reasons. First, many equations can be coalesced into a single equation using the combination step from [15]. Moreover, the AutoBatch process can be started at any stage, so one can start with an SDL representation of a signature scheme with multiple verification equations and proceed automatically from there.

runtime estimate based on the expected cost of each mathematical operation (e.g., pairing, exponentiation, multiplication). The cost of each operation is determined via a set of diagnostic tests conducted when the library is initialized.³

4. Generate code for the resulting batch verifier. Finally, we invert the procedure of step 1 to generate a working verifier implemented in Charm-Python. This verifier implements the SDL-specified batch verification equation as well as the individual verification equation. Based on the calculations of the previous step, the generated code embeds logic to automatically determine *which* verifier is most appropriate for a given dataset (individual or batch). Additionally, the generated code embeds a recursive *divide-and-conquer* strategy to handle cases where batch verification fails due to invalid signatures. A fragment of generated code is shown in the rightmost panel of Figure 2.

We note that processing can begin at any point in the above process. For example, one might begin with a hand-coded SDL representation of a signature scheme, and proceed directly from step 2. We will now describe each the above steps in detail.

A. Code Parsing

The Code Parsing engine extracts meaning from a Charm-Python implementation of a signature scheme. It produces a resulting SDL file that contains the data types and verification equation for the signature. This process is facilitated by two aspects: first, Charm [1] provides a generic class interface for signature schemes. This greatly constrains the code we have to work with, meaning that we only have to focus on Charm-compliant schemes. Secondly, we are assisted by the Python interpreter, which grants programatic access to the Python Abstract Syntax Tree via the `compiler.ast` module.

Code parsing consists of the following stages. First, we parse the entire signature scheme file into a Python AST node. We refer to this as the root node. Next, we identify the AST node of the signature `verify()` method. We then use heuristics to identify one comparison in this function that is fundamentally responsible for the signature verification process.

From this point, we next build a map of variable names, types, structure, and operations. We do this by visiting all assignment statements in the code using Python’s AST NodeVisitor class. For each assignment,

³Obviously these experiments are very specific to the machine and curve parameters on which they are run. Hence, we re-run these experiments whenever the library is initialized with a given set of parameters.

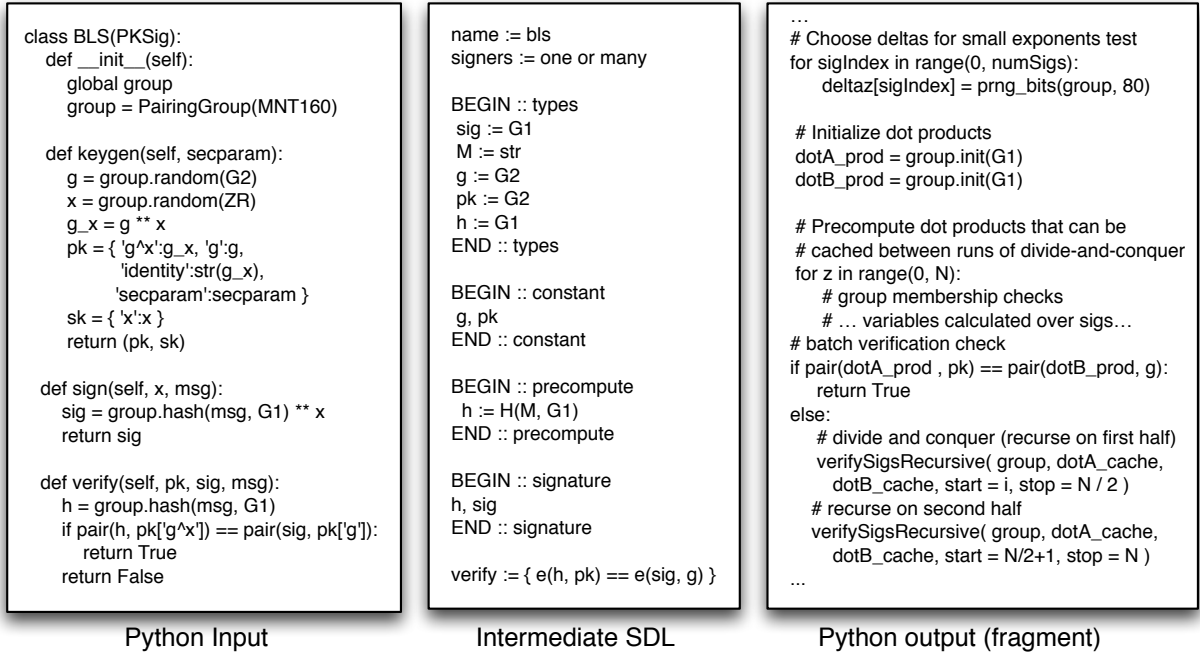


Figure 2. The Boneh-Lynn-Shacham (BLS) signature scheme [6] at various stages in the AutoBatch toolchain. At the left, an initial Charm-Python implementation of the scheme. In the center, an SDL representation of the same scheme, programmatically extracted by the Parsing Engine. At right, a fragment of the resulting batch verifier generated after applying the Batcher and Code Generator.

we check the properties of that assignment using a further set of heuristics, which we store in a database. If we determine that a given assignment is relevant, we extract certain information about it. We use this information to identify any operations that are relevant to the verification equation, even if they are spread throughout the file.

We also use our map to determine the *type* of each variable referenced. To obtain this, we apply known rules to infer type. For example, we know that certain hash calls indicate an element of $\mathbb{G}_1$, a pairing indicates an element in $\mathbb{G}_T$, random element generation calls typically indicate the type of element being generated, and so on.⁴ Our database currently includes signatures for the following types:

- 1) Python’s lambda functions, which may be used to compute dot-product functionality.
- 2) All pairings and their parameters and types.
- 3) All hashes and their parameters and types.
- 4) All Python dictionaries, their key names, their value names, and their types. Charm makes ex-

⁴We believe that this approach may also be useful in the future for static checking and formal verification of dynamically-typed cryptographic implementations.

tensive use of this data structure, so this is particularly important.

5) All constant numbers and strings.

If the assignment expression does not match any of our signatures, we perform a recursive traversal on its elements to determine if any of its sub-elements fit one of our signatures. If we find a match, we return that signatures information.

This process is not perfect. It makes certain assumptions about the structure of an implementation, and therefore is highly dependent on the quality of our heuristic rules. However, the more code we examine, the more powerful our inference becomes.

Finally, we transform the verify equation into an SDL file that the Batcher understands. This requires several straightforward transformations of the verify equation, to produce a simple bilinear-map based representation. In addition, we prepend a list of all constants, precomputed values, variable names and types to the file.

B. Batching and Optimization

Given an SDL file containing the verification equation and variable times, the Batcher applies a series of optimizations to the verification equation in order to derive an efficient batch verifier. Many of these techniques

are drawn from previous works [10], [15]. However, unlike previous works we are able to programmatically identify when they are applicable, and apply them to the verification equation in a consistent way. The Batcher assumes that the input will be a collection of η signatures, possibly on different messages and public keys (or identities). To construct a batch verifier, the Batcher first combines all instances into one equation using the small exponents test, Technique 1. Next, it optimizes the resulting equation using Techniques 2-4.

The order in which Techniques 2-4 are applied varies from scheme to scheme. Our batcher tries various orderings based on heuristics, which examine the equation and attempt to estimate which techniques will result in a successful batch. In some cases a single technique must be applied *multiple* times, generally with one or more different techniques used in between. For some complex schemes, we may specify some guidance on the ordering manually to improve performance. Resolving this is an open problem for our techniques.

Technique 1: Combine equations. Assume we are given η instances that can be verified using the individual verification equation. We then combine all instances into one equation by applying the Combination Step of [15], which employs as a subroutine the small exponents test from [4]. This results in a single verification equation. The correctness of the resulting equation requires that all elements be in the correct subgroup, i.e., that group membership has already been checked. AutoBatch ensures that this check will be explicitly conducted in the final batch verifier program.

As to the security of this step, suppose we use security parameter λ in the small exponents test. Then Ferrara et al. [15, Theorem 3.2] prove that this equation will verify if and only if all individual equations verify, except with probability at most $2^{-\lambda}$. By default in AutoBatch, we set $\lambda = 80$.

Technique 2: Move exponents into the pairing. When a pairing of the form $e(g_i, h_i)^{\delta_i}$ appears, move the exponent δ_i into $e()$. Since elements of $\mathbb{G}_1$ and $\mathbb{G}_2$ are usually smaller than elements of $\mathbb{G}_T$, this gives a noticeable speedup when computing the exponentiation.

Replace $e(g_i, h_i)^{\delta_i}$ with $e(g_i^{\delta_i}, h_i)$

Wherever possible, we move the exponent into the group with the lowest exponentiation cost. We identify this group based on a series of operation microbench-

marks that run automatically at code initialization.⁵

Technique 3: Combine pairings with common elements. When two or more pairings share a common first or second element, they can be combined. For example:

Replace $e(a, g) \cdot e(b, g)$ with $e(ab, g)$

As Ferrara et al. note [15], in rare cases it can be useful to apply this technique in reverse: splitting a single pairing into two pairings, to allow for more efficient batch verification. E.g., this can be applied to the ring signature scheme due to Boyen [9] in order to next apply the technique below.

Technique 4: Optimize the Waters Hash. A variety of bilinear signature schemes employ a hash function by Waters [36], which can be generalized [31], [13]. Assume the identity is a k -bit string $V = v_1 v_2 \dots v_z$ where each v_i is a short string. The hash function is evaluated as $u' \prod_{i=1}^m u_i^{v_i}$.

When batching η equations containing the Waters hash, one will often end up with something of the form $\prod_{j=1}^{\eta} e(g_j, \prod_{i=1}^z u_i^{v_{ij}})$. This can be rewritten to make the number of pairings independent of the number of equations one wants to batch.

Replace $\prod_{j=1}^{\eta} e(g_j, \prod_{i=1}^z u_i^{v_{ij}})$ with $\prod_{i=1}^z e(\prod_{j=1}^{\eta} g_j^{v_{ij}}, u_i)$

C. The Security of AutoBatch

At this point in the narration, we take a short break from our description of how AutoBatch works to argue that that batching and optimization techniques applied in the previous section preserve the security of the original signature scheme, up to a negligible probability of error.

Theorem 3.1 (Security of AutoBatch): Let λ be the security parameter used in the small exponents test during Technique 1. Then the batch verification algorithm resulting from an application of the above techniques, as is done in AutoBatch, is a pairing-based batch verifier that accepts an invalid batch with probability at most $2^{-\lambda}$.

Proof: The truth of this theorem after the initial and only application of Technique 1 follows directly from the proof of this step in [15, Theorem 3.2]. This step introduces a $2^{-\lambda}$ probability of error. After this point, we apply Techniques 2-4 in arbitrary order and

⁵For many common elliptic curves, this is the $\mathbb{G}_1$ base group. However, in some curves the groups $\mathbb{G}_1$ and $\mathbb{G}_2$ have similar operation costs; this may give us some flexibility in modifying the equation.

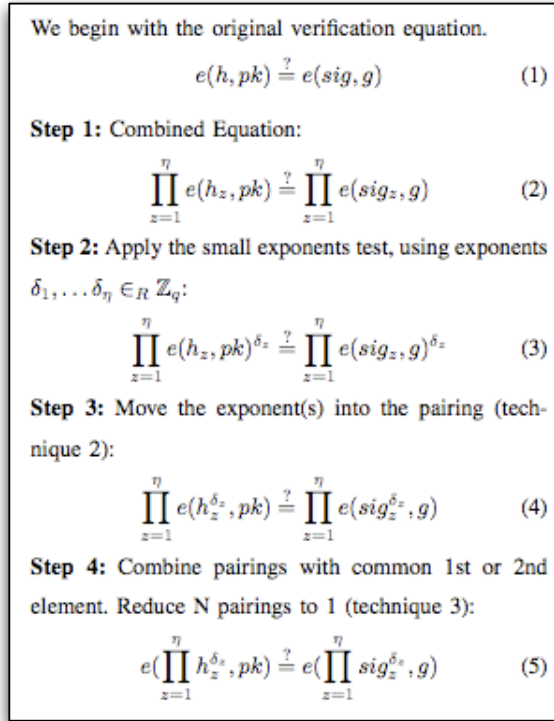


Figure 3. A fragment of the machine-generated security proof of a single-signer batch verifier for the Boneh-Lynn-Shacham (BLS) signature scheme [6]. An early portion of the proof asserted that a group membership test would be done prior to checking the final equation and defined h to be the hash of the message.

potentially multiple times. Each of these techniques involve substituting one equation or value for an equivalent formulation of that equation or value and thus the equation output by our Batcher is equivalent to the equation output after Technique 1. ■

D. Analysis and Machine-Generated Security Proofs

Once the Batcher has produced a final equation for the batch verifier, it counts the number of operations required as a function of the batch size. These operations include point operations, pairings, hashes, as well as random element generation. It then combines this operation count with a database of average operation times that were measured at library initialization. The resulting calculation allows it to determine the “crossover point”, i.e., the batch size where batch verification becomes more efficient than individual verification.

The Batcher produces both an SDL file and, optionally, a human-readable proof of security for the resulting batch verifier. This proof is a LaTeX file that includes the individual and batch verification equations, with an enumeration of the various steps used to convert the

former into the latter. This proof is designed to give users confidence in the correctness of the Batcher’s output. One example of such a proof for the *single-signer* batch verification of the BLS signatures [6] is presented in Figure 3. The resulting batching equation is the same as the one proposed by [6].

A full machine-generated proof appears in Appendix B for the batch verification of the HW signatures [22], which is a novel contribution of this work.

E. Code Generation

The output of the Batcher is a batch verification equation encoded in Scheme Description Language (SDL). This file defines all of the datatypes for the signature, message and public key (or identity and public parameters in the case of an identity-based signature). The Code Generator converts this SDL representation into a useable Python signature class that can operate on real batch inputs.

The Code Generator is essentially a mirror of the Code Parser. However, its design is substantially simpler, since there is less ambiguity in the layout of an SDL description. It translates the individual *and* batch verification equations into Python code, and wraps them with the following additional logic components:

- 1) **Group membership tests.** For each element in the signature (and optionally the public key, if the user requests)⁶ the membership of the group is tested using an exponentiation. Section II-C discusses the importance and details of this test.
- 2) **Pre-computation.** Several values often will be re-used within a verification equation. When this happens, the batch verifier can *pre-compute* certain results once, rather than needlessly compute them several times.
- 3) **Technique selection.** For relatively small batch sizes, it may be *more* efficient to bypass the batch verifier and simply verify the signatures using the individual verification function. For this reason, our Code Generator generates this function as well (the output of the Batcher contains both functions), and adds logic to programmatically choose between batch and individual verification when the batch size is below a certain threshold automatically determined in the Analysis phase.
- 4) **Invalid signature detection.** To handle the presence of invalid signatures in a batch, our batch verifier code includes a recursive *divide-and-conquer* strategy to recover from a batching fail-

⁶In many applications we can assume that the public keys are trusted, thus we can omit group membership testing on these values.

ure (see e.g., [15] for a discussion of this). On failure, this verifier divides the signature collection into two halves and recurses by repeating verification on each half until all of the invalid signatures have been identified.

IV. IMPLEMENTATION AND PERFORMANCE DETAILS

A. Experimental Setup

To evaluate the performance of our techniques we implemented them as part of the Charm prototyping framework [1]. Charm is a Python-based cryptographic prototyping framework, and provides native for bilinear-map based crypto and other useful primitives, e.g., hashing and serialization. We used a version of Charm that implements all bilinear group operations using the C-based MIRACL library [33].⁷ The necessary MIRACL calls are accessed from within our Python code via the C module interface.

To determine the performance of our system in isolation, we first conducted a number of experiments on various components of our code. First, we used the code extraction component to convert several Python signature implementations into our intermediate “SDL” representation. Next, we applied our batcher to the SDL result in order to obtain an optimized equation for a *batch verifier*. We then applied our code generator to convert this representation into a functioning batch verifier program, which we applied to various test data sets.

Hardware configuration. For consistent results we ran all of our experiments on a single hardware platform: a 2 x 2.66 GHz 6-Core Intel Xeon Macintosh Pro running MacOS version 10.7.2 with 12GB of RAM. We ran all of our tests within a single thread, and thus used resources from only a single core of the Intel processor. We instantiated all of our cryptographic implementations using a 160-bit MNT elliptic curve provided with MIRACL.

B. Signature Schemes used as Test Cases

We ran our experiments using three signature schemes, three identity-based signature schemes and one identity-based ring signature scheme, as summarized in Figure 4. All of these schemes are pairing-based schemes. (See Section II for the definitions of these signature types and the background on a pairing.) These

⁷The version of Charm we used (.30) has not been officially released, but can be found in the Charm github repository at www.charm-crypto.com. It uses MIRACL 5.5.4 for bilinear group operations.

schemes were selected because they represent some of the shortest and most practical schemes in the literature.

The batch verification algorithm for the CDH-based scheme due to Hohenberger and Waters [22] is, to our knowledge, the first batching algorithm for this scheme and it was machine-generated by AutoBatch. Prior batch verifiers were known for the remaining schemes (see [15] for a summary) and we note that our machine-generated verifiers match their efficiency.

C. Batch Verification Time: Microbenchmarks

To evaluate the overall efficiency of our approach, we implemented several pairing-based signature schemes in Charm and applied our techniques to extract an SDL-based intermediate representation of the scheme’s verification equation; derive an optimized batch verifier for the scheme; and generate a new Python program implementing the batch verifier. We measured the processing time for each of the above steps. Our timings, averaged over five runs, are presented in Figure 5.

To obtain our microbenchmarks, we ran AutoBatch on several exemplary bilinear signature schemes, including the BLS [6], CHP [10] and HW [22] signature schemes, the Hess [21] and Waters [36] identity-based signatures and the CYH ring signature [14]. We then experimented with these schemes at different batch sizes, to evaluate their raw performance. The results are presented in Figure 6.

Each graph shows the average per-signature verification time for a batch of η signatures, with η increasing from 1 to 100. We conducted these tests by first generating a collection of η keypairs and random messages,⁸ then computing a valid signature over each message. We fed each collection to the batch verifier. ID-based signatures were handled in a similar manner, although we substituted random identities in place of keys. For the CYH ring signature, we constructed a group of twenty signing keys to construct a twenty member ring. In each case, we averaged our results over 20 experimental runs and computed verification time per signature by dividing the total batching time by the number of signatures batched.

D. Batch Verification in Practice

Several previous works have considered the implication of having *invalid* signatures in a batch, e.g., [26], [15], [29], [30]. For the most part, these works estimated raw signature verification times under various conditions, but did not model the implications of these results for building real systems. To evaluate how signature

⁸We used 160-byte random strings for each message.

Scheme	Model	Ind-Verify	Batch-Verify	Techniques-Order
<i>Signatures</i>				
Boyen-Lynn-Shacham (BLS) [7] (same signer)	RO	2η	2	2,3
Camenisch-Hohenberger-Pedersen (CHP) [10] (same time period)	RO	3η	3	2,3
Hohenberger-Waters (HW) [22] (same signer)	plain	2η	4	2,3
<i>ID-based Signatures</i>				
Hess [21]	RO	2η	2	2,3
Cha-Cheon (ChCh) [12]	RO	2η	2	2,3
Waters [36]	plain	3η	$z + 3$	3,2,4,3
<i>ID-based Ring Signatures</i>				
Chow-Yiu-Hui (CYH) [14]	RO	2η	2	2,3

Figure 4. Digital Signature Schemes used as test cases in AutoBatch. RO stands for random oracle. For the verification, we count the total number of pairings needed to process η valid signatures, i.e., the best case for batch verification. For the Waters signatures, we set $z = 5$. The final columns indicates the order of the techniques from Section III that AutoBatch recognized as applicable and applied to obtain the resulting batch verifier.

Process	BLS / CHP / HW / Hess / ChCh / Waters / CYH (ms)
Parse input	67.5
Batch/optimize	133.3
Generate code	83.2
Total	284.0ms

Figure 5. Micro benchmark results: average time required by the Parser, Batcher, and Code Generator to process a variety of signature schemes. Batcher time also includes generating the proof and estimating cross over point between individual and batch verification.

batching might work in real life, we constructed a simulation to determine the resilience of our techniques to various denial of service attacks launched by an adversary.

Basic Model. For this experiment, we simulated a server that verifies incoming signed messages read from a network connection. We believe that this might be a reasonable model for a busy server-side TLS endpoint using client authentication, for example, or for a vehicle-to-vehicle communications base station.

Our server is designed to process as many signatures as possible, and is limited only by its computational resources.⁹ Signatures are drawn off of the “wire” and grouped into batches, with each batch size representing the expected number of signatures that can be verified in one second. Initially this number is simply a guess, which is adjusted upwards or downwards based on the time required to verify each batch.¹⁰ In practice, this approach can lead to some transient errors (batches that require significantly more or less than one second to evaluate) when the initial guess is wrong, or when conditions change. In normal usage, however, this approach converges on an appropriate batch size within

⁹This models a server that either delays, drops or redirects the signatures that it cannot handle (e.g., via load balancing).

¹⁰The adjustment is handled in a relatively naive way: the server simply computes the next batch size by extrapolating based on its time to compute the previous batch.

1-2 seconds.

1) Basic DoS Attacks: A major concern when using a batch verifier is the possibility of *service denial* or degradation, resulting from the presence of some invalid signatures in the batch. As described in §III, each of our generated batch verifiers incorporates a recursive divide-and-conquer strategy for identifying these invalid signatures. In practice, however, this recursion comes at a price; the presence of even a small number of invalid signatures can seriously degrade the performance of a batcher.

To measure this, we simulated an adversary who injects a fraction of invalid signatures into the server’s input stream. We assume that these signatures are well-mixed with the remaining valid signatures.¹¹ Within a single experimental run, the adversary tries various attack strategies, including no attack, a gradual increase in the number of invalid signatures, and sudden bursts of invalid signatures. In all cases we limited the fraction of invalid signatures received by the verifier to 15% of the overall signature stream.

Given this adversary, we measured the verifier’s throughput. The adversary injects no invalid signatures for the first 25 seconds of the experiment, then gradually ramps up its output until the number of invalid signatures received by the verifier reaches approximately

¹¹In practice, this is not a strong assumption, as a server can simply randomize the order of the signatures it receives.

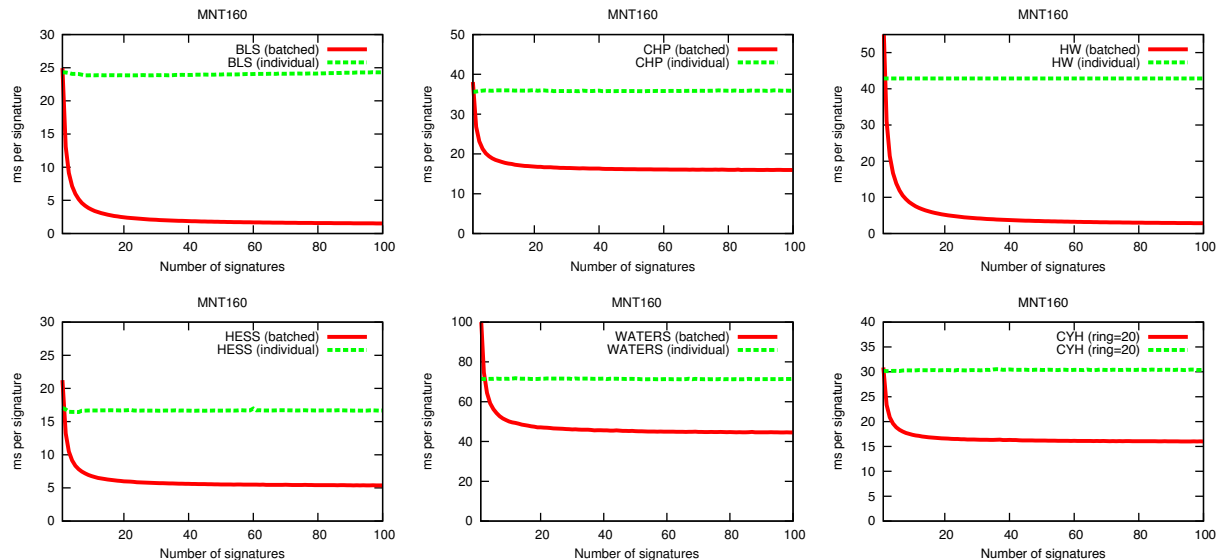


Figure 6. Signature scheme microbenchmarks for BLS (same signer), CHP (same period) and HW (same signer) signatures, Hess and Waters IBS, and CYH ring signature (20 signer ring). Per-signature times were computed by dividing total batch verification time by the number of signatures verified. Note that in the BLS and HW cases, all signatures are formulated by the same signer (as for certificate generation), while for CHP each signature was produced by a different signer but with a special restriction that they be issued in the same time period. All other schemes are without such restrictions. Individual verification times are included for comparison.

15%. At approximately 55 seconds into the experiment, the adversary switches to long bursts of invalid signatures. This measures the verifier’s ability to recover from a short, bursty attack. These strategies are by no means a thorough catalog of the types of attack that a verifier might experience; rather, our effort represents simple experimentation with a few basic attack shapes.

A countermeasure. To handle extended DoS attacks, we also experimented by adding some simple DoS countermeasures within our batching algorithm. Our basic countermeasure uses knowledge of previous batches to estimate the likely number of invalid signatures that will occur in the next batch. Based on this information, the verifier can either (a) switch to individual verification, in cases where recursive batch verification is expected to underperform, or (b) optionally skip some phases of the recursive batch verification process.

For case (b), at each phase of the recursive batch verifier we estimate the probability that the batch verification will fail, and skip the verification process whenever that probability exceeds a threshold (e.g., $3/4$). For example, when we process a batch of η signatures, if we anticipate that there will be even 1 invalid signature in the batch, then verification of the whole batch is unnecessary (i.e., it should fail). Thus we can avoid the work of verifying this batch, and instead move directly to the recursive case as though verification had failed.

We repeat this check at each phase of the process where the batch size > 1 .

Analysis of results. We tested the batch verifier with and without the above countermeasures. Our results are presented in Figure 7. We observe several things. First, though our recursive process is able to deal with invalid signatures, throughput is quite sensitive to even small numbers of invalid signatures in the input stream. However, when comparing our batch verification throughput to the *individual* verification throughput, we note that *even under a significant attack* batch verification dramatically outperforms individual verification.

Thus, while an attacker is able to reduce verifier throughput, the verifier still remains quite efficient compared to the individual verification case. In other words, we are a “victim of our own success”; since batching so dramatically increases throughput in the all-valid case, the drop in throughput caused by invalid signatures makes the verifier *seem* relatively inefficient, even while it still outperforms individual verification.

The impact of our countermeasures is less obvious. Examining the raw numbers closely, we notice a marginal improvement when the rate of invalid signatures is steady, or grows slowly. However, this is offset by a slightly slower return to normal throughput when an attack ends suddenly. We also note that even at these levels, the adversary is not able to push us

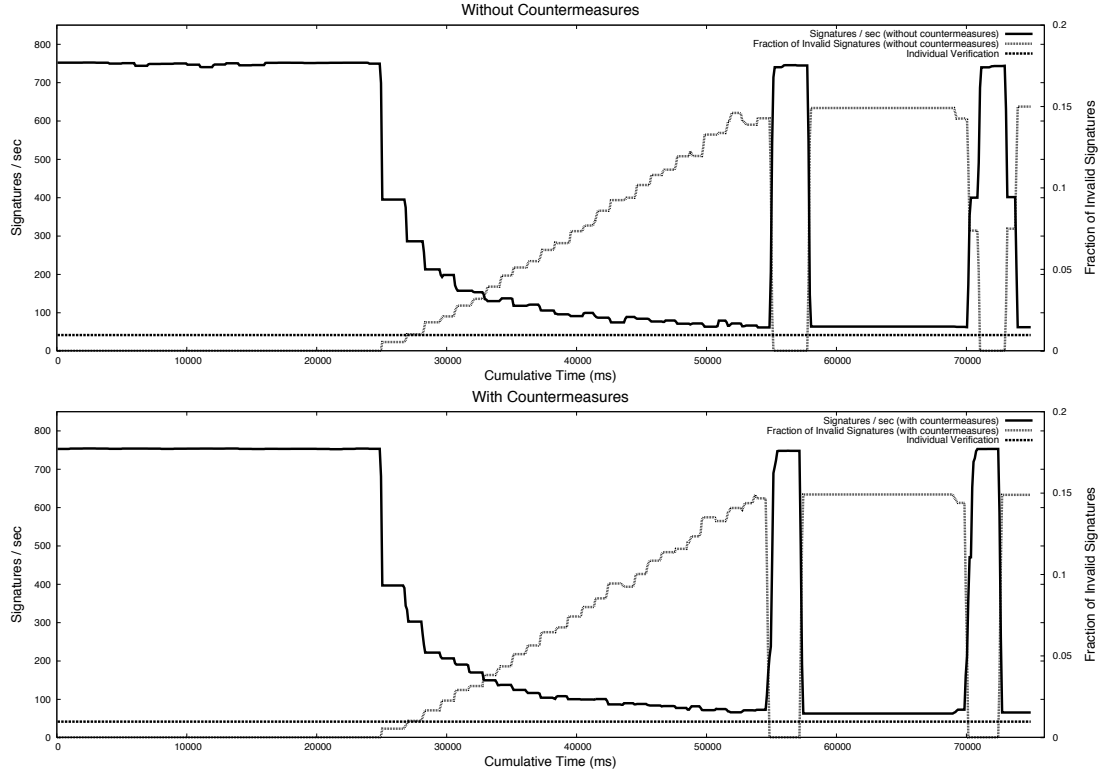


Figure 7. Simulated service denial attacks against a batch verifier (BLS signatures, single signer). The top chart considers a standard batch verifier, and the bottom chart shows the performance of a batch verifier that employs *countermeasures*. In each graph, the black line (left scale) indicates the throughput of the batch verifier measured in signatures/second. The gray line (right scale) represents the average fraction of invalid signatures that the adversary is able to inject into the stream. Note that in both experiments, the adversary varies this fraction between 0% and 15% using a consistent process. Finally, for comparison, the dashed line shows the comparable throughput of an *individual* verifier.

into territory where individual verification becomes a better strategy. Given how poorly the individual verifier performs relative to batch verification, we hypothesize that this countermeasure will not be very useful in practice.

While there is no magic bullet when it comes to batch verification of signatures, we believe that these results are interesting, and that system designers may want to take them into account. At a minimum, designers should build systems to tolerate large swings in verification throughput when an attack is present.

V. CONCLUSION

Batch verification holds great promise for applications where short signatures are a design requirement, yet high signature throughput is required. Where previous works constructed batch verifiers by hand and on a per-scheme basis, we have presented an approach that can programmatically apply batching techniques to a large class of bilinear signature schemes, including

schemes that do not yet exist. We believe that this approach will make it possible for implementers to rapidly determine which schemes can be batched efficiently. Moreover, it will help to eliminate errors that arise when human beings attempt to manually perform such optimizations.

This work leaves several open problems. On the implementation side, the latest versions of the MIRACL library include an elegant new interface for efficiently computing “multipairings” (efficient products of multiple bilinear pairings). Since Charm does not currently provide an API to this new functionality, we did not include it in our optimizations. We hope to rectify this in future versions.

We also believe that there is much to be done in making batch verifiers more resilient to invalid signatures, and therefore more useful in practice. Our work is a first step in this direction. We hope that future work will implement alternative techniques for recognizing invalid signatures in a batch, such as those considered by of Law and Matt [26], [29], [30], along with additional

countermeasure strategies for responding to service denial attacks.

Additionally, we leave open the problem of automatically batching other types of pairing-based equation, including Groth-Sahai proofs [18]. Substantial work has been conducted in this direction by Blazy *et al.* [5]. We believe that these techniques may be extended to the automated setting. Finally, a future batching system might even be capable of batching many *distinct* signature or proof types together.

REFERENCES

- [1] Joseph A. Akinyele, Matthew Green, and Avi Rubin. Charm: A framework for rapidly prototyping cryptosystems. Cryptology ePrint Archive, Report 2011/616, 2011. <http://eprint.iacr.org/>.
- [2] Boaz Barak, Ran Canetti, Jesper Buus Nielsen, and Rafael Pass. Universally composable protocols with relaxed set-up assumptions. In *FOCS*, pages 186–195. IEEE Computer Society, 2004.
- [3] Gilles Barthe, Benjamin Grégoire, Sylvain Heraud, and Santiago Zanella Béguelin. Computer-aided security proofs for the working cryptographer. In *CRYPTO*, pages 71–90, 2011.
- [4] Mihir Bellare, Juan A. Garay, and Tal Rabin. Fast batch verification for modular exponentiation and digital signatures. In *EUROCRYPT '98*, volume 1403 of LNCS, pages 236–250. Springer, 1998.
- [5] Olivier Blazy, Georg Fuchsbauer, Malika Izabachène, Amandine Jambert, Hervé Sibert, and Damien Vergnaud. Batch groth-sahai. In *ACNS '10*, pages 218–235. Springer, 2010.
- [6] Dan Boneh, Ben Lynn, and Hovav Shacham. Short signatures from the Weil pairing. In *ASIACRYPT '01*, volume 2248 of LNCS, pages 514–532, 2001.
- [7] Dan Boneh, Ben Lynn, and Hovav Shacham. Short signatures from the Weil pairing. *Journal of Cryptology*, 17(4):297–319, 2004.
- [8] Colin Boyd and Chris Pavlovski. Attacking and repairing batch verification schemes. In *Advances in Cryptology – ASIACRYPT '00*, volume 1976, pages 58–71, 2000.
- [9] Xavier Boyen. Mesh signatures: How to leak a secret with unwitting and unwilling participants. In *EUROCRYPT*, volume 4515, pages 210–227, 2007.
- [10] Jan Camenisch, Susan Hohenberger, and Michael Østergaard Pedersen. Batch verification of short signatures. In *EUROCRYPT '07*, volume 4515 of LNCS, pages 246–263. Springer, 2007. Full version at <http://eprint.iacr.org/2007/172>.
- [11] Tianjie Cao, Dongdai Lin, and Rui Xue. Security analysis of some batch verifying signatures from pairings. *International Journal of Network Security*, 3(2):138–143, 2006.
- [12] Jae Choon Cha and Jung Hee Cheon. An identity-based signature from gap Diffie-Hellman groups. In *PKC '03*, volume 2567 of LNCS, pages 18–30. Springer, 2003.
- [13] Sanjit Chatterjee and Palash Sarkar. HIBE with short public parameters without random oracle. In *ASIACRYPT '06*, volume 4284 of LNCS, pages 145–160, 2006.
- [14] Sherman S. M. Chow, Siu-Ming Yiu, and Lucas C.K. Hui. Efficient identity based ring signature. In *ACNS*, volume 3531 of LNCS, pages 499–512, 2005.
- [15] Anna Lisa Ferrara, Matthew Green, Susan Hohenberger, and Michael Østergaard Pedersen. Practical short signature batch verification. In *CT-RSA*, volume 5473 of LNCS, pages 309–324, 2009.
- [16] Amos Fiat. Batch RSA. In *Advances in Cryptology – CRYPTO '89*, volume 435, pages 175–185, 1989.
- [17] Shafi Goldwasser, Silvio Micali, and Ronald L. Rivest. A digital signature scheme secure against adaptive chosen-message attacks. *SIAM J. Computing*, 17(2), 1988.
- [18] Jens Groth and Amit Sahai. Efficient non-interactive proof systems for bilinear groups. In *EUROCRYPT '08*, volume 4965 of LNCS, pages 415–432. Springer, 2008.
- [19] Lein Harn. Batch verifying multiple DSA digital signatures. *Electronics Letters*, 34(9):870–871, 1998.
- [20] Lein Harn. Batch verifying multiple RSA digital signatures. *Electronics Letters*, 34(12):1219–1220, 1998.
- [21] Florian Hess. Efficient identity based signature schemes based on pairings. In *Selected Areas in Cryptography*, volume 2595 of LNCS, pages 310–324. Springer, 2002.
- [22] Susan Hohenberger and Brent Waters. Realizing hash-and-sign signatures under standard assumptions. In *EUROCRYPT*, pages 333–350, 2009.
- [23] Min-Shiang Hwang, Cheng-Chi Lee, and Yuan-Liang Tang. Two simple batch verifying multiple digital signatures. In *3rd Information and Communications Security (ICICS)*, pages 233–237, 2001.
- [24] Min-Shiang Hwang, Iuon-Chang Lin, and Kuo-Feng Hwang. Cryptanalysis of the batch verifying multiple RSA digital signatures. *Informatica, Lithuanian Academy of Sciences*, 11(1):15–19, 2000.
- [25] Chi-Sung Lai and Sung-Ming Yen. Improved digital signature suitable for batch verification. *IEEE Transactions on Computers*, 44(7):957–959, 1995.

- [26] Laurie Law and Brian J. Matt. Finding invalid signatures in pairing-based batches. In *Cryptography and Coding*, volume 4887 of LNCS, pages 34–53, 2007.
- [27] Seungwon Lee, Seongje Cho, Jongmoo Choi, and Yookun Cho. Efficient identification of bad signatures in RSA-type batch signature. *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences*, E89-A(1):74–80, 2006.
- [28] C. Lim and P. Lee. Security of interactive DSA batch verification. In *Electronics Letters*, volume 30(19), pages 1592–1593, 1994.
- [29] Brian J. Matt. Identification of multiple invalid signatures in pairing-based batched signatures. In *Public Key Cryptography*, pages 337–356, 2009.
- [30] Brian J. Matt. Identification of multiple invalid pairing-based signatures in constrained batches. In *Pairing*, pages 78–95, 2010.
- [31] D. Naccache. Secure and *practical* identity-based encryption, 2005. Cryptology ePrint Archive: Report 2005/369.
- [32] David Naccache, David M’Raïhi, Serge Vaudenay, and Dan Rappaeli. Can DSA be improved? complexity trade-offs with the digital signature standard. In *Advances in Cryptology – EUROCRYPT ’94*, volume 950, pages 77–85, 1994.
- [33] Michael Scott. Multiprecision Integer and Rational Arithmetic C/C++ Library (MIRACL), Oct. 2007. Published by Shamus Software Ltd., <http://www.shamus.ie/>.
- [34] Hovav Shacham and Dan Boneh. Improving SSL handshake performance via batching. In *Cryptographer’s Track at RSA Conference ’01*, volume 2020, pages 28–43, 2001.
- [35] Martin Stanek. Attacking LCCC batch verification of RSA signatures, 2006. Cryptology ePrint Archive: Report 2006/111.
- [36] Brent Waters. Efficient identity-based encryption without random oracles. In *EUROCRYPT ’05*, volume 3494 of LNCS, pages 320–329. Springer, 2005.
- [37] HyoJin Yoon, Jung Hee Cheon, and Yongdae Kim. Batch verifications with ID-based signatures. In *ICISC, Lecture Notes in Computer Science*, pages 233–248, 2004.
- [38] Fangguo Zhang and Kwangjo Kim. Efficient ID-based blind signature and proxy signature from bilinear pairings. In *8th Information Security and Privacy, Australasian Conference (ACISP)*, volume 2727, pages 312–323, 2003.
- [39] Fangguo Zhang, Reihaneh Safavi-Naini, and Willy Susilo. Efficient verifiably encrypted signature and partially blind signature from bilinear pairings. In *Progress in Cryptology – INDOCRYPT ’03*, volume 2904, pages 191–204, 2003.

APPENDIX A.

MACHINE-GENERATED BATCH VERIFICATION EQUATIONS

In Figure 8, we provide the final batch verification equations output by AutoBatch for each of the signature schemes tested.

APPENDIX B.

A MACHINE-GENERATED PROOF OF SECURITY

The following proof was automatically generated by the Batcher while processing of the HW signature scheme [22]. It has been edited to fit the two column format. This execution was restricted to signatures on a single signing key.

A. Definitions

This document contains a proof that `HW.BatchVerify` is a valid batch verifier for the signature scheme HW. Let g, A, U, V, D, w, z, h be values drawn from the key and/or parameters, and $M, \sigma_1, \sigma_2, r, i$ represent a message (or message hash) and signature. The individual verification equation `HW.Verify` is:

$$e(\sigma_1, g) \stackrel{?}{=} U^M \cdot V^r \cdot D \cdot e(\sigma_2^{\lg(i)}, w) \cdot e(\sigma_2^i, z) \cdot e(\sigma_2, h)$$

Let η be the number of signatures in a batch, and $\delta_1, \dots, \delta_\eta \in_R \mathbb{Z}_q$ be a set of random exponents chosen by the verifier. The batch verification equation `HW.BatchVerify` is:

$$e\left(\prod_{z=1}^{\eta} \sigma_{1z}^{\delta_z}, g\right) \stackrel{?}{=} U^{\sum_{z=1}^{\eta} M_z \cdot \delta_z} \cdot V^{\sum_{z=1}^{\eta} r_z \cdot \delta_z} \cdot D^{\sum_{z=1}^{\eta} \delta_z} \cdot e\left(\prod_{z=1}^{\eta} \sigma_{2z}^{\lg(i_z) \cdot \delta_z}, w\right) \cdot e\left(\prod_{z=1}^{\eta} \sigma_{2z}^{i_z \cdot \delta_z}, z\right) \cdot e\left(\prod_{z=1}^{\eta} \sigma_{2z}^{\delta_z}, h\right)$$

We will now formally define a batch verifier and demonstrate that `HW.BatchVerify` is a secure batch verifier for the HW signature scheme.

Definition B.1 (Pairing-based Batch Verifier): Let $\text{BSetup}(1^\tau) \rightarrow (q, g, \mathbb{G}, \mathbb{G}_T, e)$. For each $j \in [1, \eta]$, where $\eta \in \text{poly}(\tau)$, let $X^{(j)}$ be a generic pairing-based claim and let `Verify` be a pairing based verifier. We define *pairing-based batch verifier* for `Verify` a probabilistic $\text{poly}(\tau)$ -time algorithm which outputs *accept* if $X^{(j)}$ holds for all $j \in [1, \eta]$ whereas it outputs *reject* if $X^{(j)}$ does not hold for any $j \in [1, \eta]$ except with negligible probability.

Theorem B.2: `HW.BatchVerify` is a batch verifier for the HW signature scheme.

Scheme	Batch Verification Equation output by AutoBatch
<i>Signatures</i>	
BLS [7] (same signer)	$e(\prod_{z=1}^{\eta} h_z^{\delta_z}, pk) \stackrel{?}{=} e(\prod_{z=1}^{\eta} sig_z^{\delta_z}, g)$
CHP [10] (same time period)	$e(\prod_{z=1}^{\eta} sig_z^{\delta_z}, g) \stackrel{?}{=} e(a, \prod_{z=1}^{\eta} pk_z^{\delta_z}) \cdot e(h, \prod_{z=1}^{\eta} pk_z^{b_z \cdot \delta_z})$
HW [22] (same signer)	$e(\prod_{z=1}^{\eta} \sigma_{1z}^{\delta_z}, g) \stackrel{?}{=} U^{\sum_{z=1}^{\eta} M_z \cdot \delta_z} \cdot V^{\sum_{z=1}^{\eta} r_z \cdot \delta_z} \cdot D^{\sum_{z=1}^{\eta} \delta_z}$ $\cdot e(\prod_{z=1}^{\eta} \sigma_{2z}^{\lg(i_z) \cdot \delta_z}, w) \cdot e(\prod_{z=1}^{\eta} \sigma_{2z}^{i_z \cdot \delta_z}, z) \cdot e(\prod_{z=1}^{\eta} \sigma_{2z}^{\delta_z}, h)$
<i>ID-based Signatures</i>	
Hess [21]	$e(\prod_{z=1}^{\eta} S_{2z}^{\delta_z}, g_2) \stackrel{?}{=} e(\prod_{z=1}^{\eta} pk_z^{a_z \cdot \delta_z}, P_{pub}) \cdot \prod_{z=1}^{\eta} S_{1z}^{\delta_z}$
ChCh [12]	$e(\prod_{z=1}^{\eta} S_{2z}^{\delta_z}, g_2) \stackrel{?}{=} e(\prod_{z=1}^{\eta} (S_{1z} \cdot pk_z^{a_z})^{\delta_z}, P_{pub})$
Waters [36]	$e(\prod_{z=1}^{\eta} S_{1z}^{\delta_z}, g_2) \cdot e(\prod_{z=1}^{\eta} S_{2z}^{\delta_z}, u_1) \cdot \prod_{i=1}^l e(\prod_{z=1}^{\eta} S_{2z}^{\delta_z \cdot k_{i,z}} \cdot S_{3z}^{\delta_z \cdot m_{i,z}}, \hat{u}_i)$ $\cdot e(\prod_{z=1}^{\eta} S_{3z}^{\delta_z}, u_2) \stackrel{?}{=} A^{\sum_{z=1}^{\eta} \delta_z}$
<i>ID-based Ring Signatures</i>	
CYH [14]	$e(\prod_{z=1}^{\eta} \prod_{y=1}^l u_{y,z} \cdot pk_{y,z}^{h_{y,z} \cdot \delta_z}, P) \stackrel{?}{=} e(\prod_{z=1}^{\eta} S_z^{\delta_z}, g)$

Figure 8. These are the final batch verification equations output by AutoBatch. Due to space, we do not include the full schemes or further describe the elements of the signature or our shorthand for them, such as setting $h = H(M)$ in BLS. However, a reader could retrace our steps by applying the techniques in Section III to the original verification equation in the order specified in Figure 4.

B. Proof

Proof: Via a series of steps, we will show that if HW is a secure signature scheme, then BatchVerify is a secure batch verifier. Recall our batch verification software will perform a group membership test to ensure that each group element of the signature is a member of the proper subgroup, so here we will assume this fact. We begin with the original verification equation.

$$U^M \cdot V^r \cdot D \cdot e(\sigma_2^{\lg(i)}, w) \cdot e(\sigma_2^i, z) \cdot e(\sigma_2, h) \stackrel{?}{=} e(\sigma_1, g) \quad (1)$$

Step 1: Combined Equation:

$$\prod_{z=1}^{\eta} e(\sigma_{1z}, g) \stackrel{?}{=} \prod_{z=1}^{\eta} U^{M_z} \cdot V^{r_z} \cdot D \cdot e(\sigma_{2z}^{\lg(i_z)}, w) \cdot e(\sigma_{2z}^{i_z}, z) \cdot e(\sigma_{2z}, h)$$

Step 2: Apply the small exponents test, using exponents

$\delta_1, \dots, \delta_{\eta} \in_R \mathbb{Z}_q$:

$$\prod_{z=1}^{\eta} (e(\sigma_{1z}, g))^{\delta_z} \stackrel{?}{=} \prod_{z=1}^{\eta} U^{M_z \cdot \delta_z} \cdot \prod_{z=1}^{\eta} V^{r_z \cdot \delta_z} \cdot \prod_{z=1}^{\eta} D^{\delta_z} \cdot \prod_{z=1}^{\eta} (e(\sigma_{2z}^{\lg(i_z)}, w))^{\delta_z} \cdot \prod_{z=1}^{\eta} (e(\sigma_{2z}^{i_z}, z))^{\delta_z} \cdot \prod_{z=1}^{\eta} (e(\sigma_{2z}, h))^{\delta_z} \quad (2)$$

Step 3: Move the exponent(s) into the pairing (tech-

nique 2):

$$\prod_{z=1}^{\eta} e(\sigma_{1z}^{\delta_z}, g) \stackrel{?}{=} \prod_{z=1}^{\eta} U^{M_z \cdot \delta_z} \cdot \prod_{z=1}^{\eta} V^{r_z \cdot \delta_z} \cdot \prod_{z=1}^{\eta} D^{\delta_z} \cdot \prod_{z=1}^{\eta} e(\sigma_{2z}^{\lg(i_z) \cdot \delta_z}, w) \cdot \prod_{z=1}^{\eta} e(\sigma_{2z}^{i_z \cdot \delta_z}, z) \cdot \prod_{z=1}^{\eta} e(\sigma_{2z}^{\delta_z}, h) \quad (3)$$

Step 4: Combine pairings with common 1st or 2nd element. Reduce N pairings to 1 (technique 3):

$$e(\prod_{z=1}^{\eta} \sigma_{1z}^{\delta_z}, g) \stackrel{?}{=} U^{\sum_{z=1}^{\eta} M_z \cdot \delta_z} \cdot V^{\sum_{z=1}^{\eta} r_z \cdot \delta_z} \cdot D^{\sum_{z=1}^{\eta} \delta_z} \cdot e(\prod_{z=1}^{\eta} \sigma_{2z}^{\lg(i_z) \cdot \delta_z}, w) \cdot e(\prod_{z=1}^{\eta} \sigma_{2z}^{i_z \cdot \delta_z}, z) \cdot e(\prod_{z=1}^{\eta} \sigma_{2z}^{\delta_z}, h) \quad (4)$$

Steps 1 and 2 form the Combination Step in [15], which was proven to result in a secure batch verifier in [15, Theorem 3.2]. We observe that the remaining steps are merely reorganizing terms within the same equation. Hence, the final verification equation (4) is also batch verifier for HW. ■

Detecting Dangerous Queries: A New Approach for Chosen Ciphertext Security

Susan Hohenberger*
Johns Hopkins University

Allison Lewko[†]
University of Texas at Austin

Brent Waters[‡]
University of Texas at Austin

January 4, 2012

Abstract

We present a new approach for creating chosen ciphertext secure encryption. The focal point of our work is a new abstraction that we call *Detectable Chosen Ciphertext Security* (DCCA). Intuitively, this notion is meant to capture systems that are not necessarily chosen ciphertext attack (CCA) secure, but where we can detect whether a certain query CT can be useful for decrypting (or distinguishing) a challenge ciphertext CT*.

We show how to build chosen ciphertext secure systems from DCCA security. We motivate our techniques by describing multiple examples of DCCA systems including creating them from 1-bit CCA secure encryption — capturing the recent Myers-shelat result (FOCS 2009). Our work identifies DCCA as a new target for building CCA secure systems.

1 Introduction

A central goal of public key cryptography is to design encryption systems that are secure against chosen ciphertext attacks. Public key encryption systems that are chosen ciphertext attack (CCA) secure are robust against powerful adversaries that are able to leverage interaction with a decryptor. Such an attacker is modeled by allowing him to query for the decryption of any ciphertext except a challenge ciphertext for which he is trying to break. This includes ciphertexts derived from the challenge ciphertext¹. Due to its robustness against powerful attackers, chosen ciphertext security has become the accepted goal for building secure encryption. For this reason, building chosen ciphertext secure systems has been a central pursuit of cryptographers for over twenty years and we have seen many distinct approaches to achieving CCA security.

*Sponsored by the Defense Advanced Research Projects Agency (DARPA) and the Air Force Research Laboratory (AFRL) under contract FA8750-11-2-0211, the Office of Naval Research under contract N00014-11-1-0470, a Microsoft Faculty Fellowship and a Google Faculty Research Award. Applying to all authors, the views expressed are those of the authors and do not reflect the official policy or position of the Department of Defense or the U.S. Government.

[†]Sponsored by a Microsoft Research Ph.D. Fellowship.

[‡]Supported by NSF CNS-0915361 and CNS-0952692, AFOSR Grant No: FA9550-08-1-0352, DARPA PROCEED, DARPA N11AP20006, Google Faculty Research award, the Alfred P. Sloan Fellowship, and Microsoft Faculty Fellowship, and Packard Foundation Fellowship.

¹We use “CCA” and “CCA2” interchangeably in this paper.

Early pioneering work in chosen ciphertext security [24, 14, 27] introduced the technique of leveraging Non-Interactive Zero Knowledge Proofs (NIZKs) [5] to build CCA-secure encryption systems from chosen plaintext secure encryption systems. Roughly, a NIZK is used to prove that a ciphertext is “well-formed” or legal. Later Cramer and Shoup [12, 13] introduced the first practical CCA-secure systems that were built on specific number theoretic assumptions such as Decisional Diffie Hellman. These techniques implicitly embed a certain form of designated verifier Non-Interactive Zero Knowledge proofs in them. More recently, different methods for building chosen ciphertext security from Identity-Based Encryption [7] and Lossy Trapdoor Functions [26] have emerged. In addition, Myers and shelat [23] described general methods for amplifying CCA encryption of 1 bit to many bits.

In this work, we introduce a new approach to obtaining chosen ciphertext secure systems. The focal point of our work is a new abstraction that we call *Detectable Chosen Ciphertext Security* (DCCA). Intuitively, this notion is meant to capture systems that are not necessarily CCA secure, but where we can detect whether a certain query CT can be useful for decrypting (or distinguishing) a challenge ciphertext CT^* .

A system that is DCCA secure will be associated with a boolean function F that takes in three inputs: a public key pk , a challenge ciphertext CT^* and a query ciphertext CT. The function will output 1 if the query CT is “dangerous” for the an attacker wishing to distinguish CT^* . A DCCA secure system must have the following two properties stated here informally:

- **Unpredictability** Without seeing CT^* it should be hard to find a ciphertext CT such that $F(pk, CT^*, CT) = 1$. In other words, an attacker must first see a challenge ciphertext in order to discover a dangerous query for it.
- **Indistinguishability** The system will be secure under a detectable chosen ciphertext attack *if* the attacker is limited to decryption queries of ciphertexts CT where $F(pk, CT^*, CT) = 0$ for challenge ciphertext CT^* . I.e. the system is CCA secure if the attacker does not make dangerous queries.

The goal of our work will be to construct fully chosen ciphertext secure systems from detectable CCA-secure systems. We first motivate this goal by observing multiple DCCA systems that naturally occur:

- **Many bit encryption from 1-bit CCA** Suppose we have a 1-bit CCA-secure system and we wish to encrypt multiple bits by concatenating multiple 1-bit encryptions together. The resulting system is no longer chosen ciphertext secure, but is DCCA secure. The detecting function F is 1 iff any of the 1-bit ciphertext components between CT^* and CT are equal. This scenario is akin to the problem of showing that bit encryption is complete considered by Myers and shelat [23], where they worried about such “quoting” attacks.
- **Tag-Based Encryption Systems** MacKenzie, Reiter and Yang [22] and Kiltz [20] define a tag-based encryption scheme as an encryption scheme that takes in an additional “tag” parameter on encryption and decryption. The security game allows an attacker to make decryption queries with any tag parameter t , except for the tag t^* that the challenge ciphertext is encrypted under. Several examples of tag-based schemes exist. Kiltz [20] gave a direct construction from the linear assumption. The CCA1-secure encryption variant of the Canetti, Halevi and Katz [7] construction where the tag is an IBE identity is an additional example.

One can also view the CCA1-secure variant of Peikert and Waters [26] as a tag-based scheme, where the tag is the “branch” in an all-but-one encryption scheme.

Most of the above examples of tag-based encryption can be proven selectively secure, where an attacker must commit to the tag of the challenge ciphertext before seeing the public key. However, if we are willing to utilize complexity leveraging arguments, we can argue that these are adaptively secure. In addition, the CHK-lite transformation will be an adaptively secure tag-based scheme if used with an adaptively secure Identity-Based Encryption system.

We observe that adaptively-secure tag-based encryption immediately gives rise to DCCA-secure encryption. A ciphertext of the DCCA-secure system consists of a random tag t plus a tag-based encryption of the message under the tag t . Decryption follows analogously and the function F simply tests if two ciphertexts have the same tag. Unpredictability follows from having a large tag space. Although it is already possible to transform tag-based encryption into CCA-secure encryption using a strongly unforgeable signature [20], these examples demonstrate natural DCCA systems. We detail this argument in Appendix D.

- **“Sloppy” CCA Encryption** One can envision that in practice an encryption system is CCA secure, but an implementation of it is not due to certain nuances. For instance, suppose a number theoretic library had a slack bit in its representation of group elements (e.g. a bit that was always supposed to be 0, but if set to 1 does not affect any computations.) A CCA attacker could exploit this weakness in an implementation, however, it is possible that the system would still be DCCA secure. Thus, one might use our techniques as a hedge against such problems. This is somewhat analogous to recent work [2] on applying deterministic encryption as a hedge against faulty random bit generation.

In addition to the examples listed above, we believe that it is useful to identify DCCA security as a new “target” for achieving chosen ciphertext security.

Overview of Our Techniques We now give an overview of our construction and proof. Our construction will build a chosen ciphertext secure system from three components: a chosen plaintext secure system, 1-bounded CCA-secure system ², and a detectable CCA-secure system. Since DCCA security (trivially) implies CPA, and we can build 1-bounded CCA from CPA encryption [25, 11, 10], it follows that all components are realizable from DCCA as a building block.

A public key from our system consists of three components. An “inner” public key PK_{in} which is a DCCA public key and two “outer” keys PK_A, PK_B respectively from 1-bounded CCA and CPA secure systems. To encrypt a message M , one first chooses the randomness r_A, r_B to be used for the outer encryptions and then encrypts the tuple (r_A, r_B, M) under the inner (detectable) key to compute an inner ciphertext CT_{in} . Next, the encryption algorithm encrypts CT_{in} under the outer public key PK_A using randomness r_A to get CT_A . It then analogously creates CT_B as the encryption of CT_{in} under key PK_B and randomness r_B . The output ciphertext is $CT = (CT_A, CT_B)$.

The structure of our ciphertexts is that the two outer ciphertexts both encrypt the same message — the inner ciphertext. This ciphertext itself encrypts the message and the randomness used to create the outer ciphertexts. Thus, the outer ciphertexts indirectly encrypt their own randomness. ³

²A 1-bounded CCA-secure encryption system is secure against one chosen ciphertext query.

³This construction implicitly assumes that the length of the random string needed for encryption is dependent only on the security parameter and is independent (or at least smaller than) the message size of the outer ciphertexts.

The decryption algorithm will receive $CT = (CT_A, CT_B)$ and first decrypt CT_A to get CT'_{in} and decrypt this to get (r'_A, r'_B, M') using the appropriate secret keys. Finally, it will check that the ciphertext is well formed by itself encrypting CT'_{in} under PK_A, PK_B and the respective randomness r'_A, r'_B and validating that the output matches CT_A and CT_B before accepting M' as the message. Our encryption system has elements both of the Naor-Yung [24] two key method for our two outer keys and the Myers-shelat [23] method of embedding outer randomness in inner ciphertexts.

Security of our system depends on the premise that no attacker is able to learn the message encrypted in the inner ciphertext. This will follow from the Detectable CCA security *if* we are able to guarantee that an attacker is unable to make any ciphertext queries CT_A, CT_B where the decryption of CT_A , denoted CT_{in} , is related to the inner component of our challenge ciphertext CT^*_{in} according to the DCCA function F . Intuitively, we hope to achieve this from the combination of two features of our system. First, the 1-bounded CCA security of PK_A will (hopefully) make it difficult to create an encryption under PK_A related to CT^*_{in} . Second, the embedded randomness will allow us to check that ciphertexts are well formed and thus answer multiple ciphertext queries under the Naor-Yung two key type manner.

The trickiness in proving security lies with the embedded randomness which is a two-edge sword. On one hand, forcing the attacker queries to embed randomness allows a reduction algorithm to decrypt if it knows either one of the two outer keys. On the other hand, it is not clear how such a reduction can create valid ciphertexts while playing the 1-bounded CCA game, since a reduction algorithm will not know the randomness r_A to embed. Thus, this circularity creates a fundamental barrier similar to difficulties encountered in attempts to create trapdoor functions from encryption [15].

We deal with this by arguing security in an indirect way that steps around this barrier. We first define a security game specific to our construction called nested indistinguishability. In this game, an attacker will receive a public key and is allowed to make decryption queries. The attacker at some point submits a single message M . The challenger will flip a coin z . If $z = 0$, the challenger creates a valid encryption of M ; otherwise, if $z = 1$ the challenger creates an encryption where the innermost message is all 0's — it neither includes the message nor the embedded randomness. The attacker continues to make decryption queries (other than the challenge ciphertext) and wins if it is successfully able to guess z . It follows that if no attacker is successful in this game, then our system is chosen ciphertext secure.

To prove security of this nested indistinguishability game, we begin by defining a “bad event”. The bad event is defined to be when the attacker submits a query (CT_A, CT_B) such that $CT_A \neq CT^*_A$ where CT^*_A is from the challenge ciphertext and the decryption of CT_A gives a ciphertext that is related to the inner challenge ciphertext according to F . If we can argue that such bad events only occur with negligible probability, then security of the nested indistinguishability game follows straightforwardly from DCCA security.

The crux of our proof is how we eliminate the possibility of a bad event. We do so in an indirect manner. We begin by arguing this event cannot happen in the case where $z = 1$, which is where all 0's are encrypted and the randomness is not embedded. In this case, we get the best of both worlds. We are able to require that the attacker's queries have the randomness embedded in them,

We can justify this assumption with the common technique of using a seed to a (variable length) Pseudo Random Generator (PRG) as the input to each encryption algorithm. The PRG can then extend the randomness to whatever length is required by the underlying encryption system. By using this justified assumption in our definitions, we are able to simplify the presentation of our construction and proofs. In contrast, Myers and shelat [23] explicitly carry the PRG technique through their exposition. This choice gives our exposition and proof an advantage in simplicity.

so that we can check ciphertext well-formedness, however, *the challenge ciphertext is not required to embed the outer randomness*. We argue that the bad event does not happen by applying a set of hybrid experiments. First, we change CT_B^* to be an encryption of all 1's. Next, we change the decryption algorithm to decrypt using the secret key for PK_B . Finally, we change CT_A^* to be an encryption of all 1's. In each experiment we argue that the chance of a bad event must be very close to that of the prior experiment. For the last step we leverage the 1-bounded CCA property of the first component. Finally, we note that in the last experiment the probability of a bad event is negligible since the inner challenge ciphertext CT_{in}^* is replaced by all 1's and is not even present.

One interesting question is why is 1-bounded CCA security needed for the PK_A since at the last step in the proof we can use the secret key SK_B to execute decryption. While this is true, it is actually possible for the bad event to occur on a malformed ciphertext that will not decrypt. We need the 1-bounded CCA property to detect the occurrence of the bad event in this case during the security reduction.

We are not able to argue the lack of a bad event in a similar manner for the $z = 0$ (embedded randomness) case due to the aforementioned circularity problems. Instead, we can infer this from the lack of event in the $z = 1$ case along with DCCA security. To prove this, we can create an algorithm that plays the DCCA indistinguishability game while simulating the nested indistinguishability game to the attacker. The simulator will choose the outer keys and outer randomness for the challenge ciphertext itself. It submits the message and outer randomness as one inner message and the 0's string as another. Then it will be able to decrypt all ciphertext queries until a bad event happens using its keys in addition to the DCCA decryption oracle. Once a bad event query is made though, it is stuck. However, it need not go any further! The fact that the attacker was able to create a bad event at all must mean that the message and randomness were embedded. It can then break the DCCA distinguishing game. Thus, we can infer that the bad event happens with negligible probability in either case. The remainder of the proof follows straightforwardly.

Comparison to Myers-shelat Myers and shelat [23] showed how to achieve many-bit chosen ciphertext security from 1-bit chosen ciphertext security and motivated us to explore the notion of detectability. They created a system using an inner/outer structure where the inner ciphertext encrypted the outer random coins. Their inner scheme, built from 1-bit CCA, is what they call “unquoteable” secure. Their concept is roughly analogous to a specific instance of a DCCA scheme. Encryptions of many-bit messages are concatenations of 1-bit encryptions; the system is chosen ciphertext secure as long as queries do not copy a 1-bit ciphertext component of the underlying scheme. For the outer scheme, they use a notion of security that is an amalgam of unquoteability and non-malleability. Their outer construction follows a specific adaptation of the Choi et. al. [10] methods applied to the 1-bit primitive. (No two key structure is used.) Their proof relies on defining quoting attacks on *both* the inner and outer layers and then establishing a certain order that outer quoting attacks must happen before inner quoting attacks.

We believe our methods offer benefits in terms of generality, simplicity, and efficiency. First, our general notion of Detectable Chosen Ciphertext Security can be realized by multiple systems. These include the 1-bit to many-bit examples, the tag-based encryption class and future systems that can leverage this as a new target path for creating CCA secure encryption.

Another key difference is that the outer layer of our scheme is built from simple 1-bounded CCA and CPA-secure parts. We argue these provide simpler concepts and are easier to work with. In addition, one can instantiate them from *any* 1-bounded encryption system. For instance, we

can apply any candidate 1-bounded CCA-secure system and do not need to work through the Choi et. al. [10] construction. Instead we can apply the 1-bounded CCA system of Cramer et. al. [11], which is significantly more efficient and simpler than the non-malleable systems of either PSV [25] or Choi et. al. [10]. We also regard avoiding a combination security definition between 1-bounded CCA (or non-malleability) and detection as a benefit for simplicity. This simplification will also improve efficiency in the case where there is a candidate CPA primitive that is more efficient than the candidate DCCA primitive, since we can build the 1-bounded scheme out of the CPA primitive.

Our choice of abstractions and structure allow us to have a simple proof. We can eliminate the possibility of a bad event using a basic Naor-Yung two key argument. Then once we are able to eliminate this, the rest of the proof follows in a straightforward manner.

Why not CCA1? One intriguing possibility is to try to leverage our techniques to build full chosen ciphertext security from CCA1 security. A natural direction would be to use a CCA1 system for the inner component in place of the detectable encryption scheme. The intuitive rationale would be if the outer keys are 1-bounded CCA or non-malleable then the queries produced by the attacker should not be related to the inner challenge ciphertext and thus CCA1 might suffice. Unfortunately, we were able to create an attack oracle which breaks full CCA security in our scheme, yet does not perturb the 1-bounded CCA or CCA1 primitives, giving evidence that this approach may not work. However, the oracle we use is quite strong and “exotic”. This suggests that there might be primitives that lie somewhere in between DCCA and CCA1. One interesting example is the CCA-1 secure “Cramer-Shoup lite” [12] cryptosystem. There exists a malleability attack on a challenge CT^* that produces a query ciphertext which has the same distribution as a fresh encryption of a random message. Hence the CS-lite system is not CCA secure. However, it would be very interesting and surprising if there existed attack algorithms that matched the above oracle.

We describe these issues in more detail in Section 5.

1.1 Related Work

Relaxations of CCA Multiple relaxations of chosen ciphertext security have been proposed in the literature.

One interesting class of relaxations is the notion of Replayable Chosen Ciphertext Security [8] and other similar works [30, 1]. These works aim to capture the concept that some malleability attacks might intuitively be benign. In particular, consider a cryptosystem where an attacker is only able to maul a ciphertext CT encrypting a message M into a different ciphertext C' that encrypts the *same* message M . If an application (or user) makes all decisions based on the decrypted plaintexts as opposed to the representation of the ciphertext such notions might be sufficient.

The primary goal of RCCA is to formally capture a form of “good enough” security under ciphertext attacks. In contrast, Detectable CCA inherently does not have good enough security on its own. In DCCA systems, it may be possible to maul ciphertexts to be encryptions of different messages or even create attack ciphertexts that each target a single bit of a target ciphertext. Thus, our primary focus is to create CCA security from a fundamentally less secure DCCA building block.

We observe that DCCA does not imply RCCA. In [8], the authors gave an example of an RCCA scheme that could not be publicly detected. Conversely, not all DCCA schemes will be RCCA secure. Our bit encryption instance serves as an example. We also note that [8] discusses a notion of detectability and introduces a definition that combines replayable and detectable properties. This combined definition is a particular instance of DCCA. However, they do not explore the notion of

detectability in isolation or how to build CCA security from it. Canetti, Krawczyk, and Nielsen [8] do show how to create CCA security from RCCA security using the KEM/DEM framework.

Finally, Hofheinz and Kiltz [17] introduce a notion they call Constrained CCA security particular to developing Key Encapsulation Mechanisms. In their definition an attacker must include a predicate p along with each query ciphertext CT. The challenger will only answer the query if the predicate evaluated on the decrypted key of the ciphertext is true and the predicate is false for all but a negligible fraction of possible KEM keys. While this notion is weaker than CCA security, they show that when combined with a (symmetric) authenticated encryption scheme, the resulting system is CCA secure.

Other Related Work Goldwasser and Micali [16] gave the first formal definition of security for public key encryption systems. Naor and Yung [24] and Rackoff and Simon [27] extended this to include chosen ciphertext attacks.

Naor and Yung [24] initiated the approach of leveraging NIZKs to build chosen ciphertext security by introducing their “two key” method. A NIZK would guarantee the integrity of the ciphertext by giving a proof that the same message was encrypted to two keys. While their system gave security against lunchtime or CCA1 attacks, Dolev, Dwork and Naor [14] showed how to achieve full CCA2 security. In addition, they introduced the fundamental concept of non-malleability. Sahai [29] introduced a concept of simulation sound NIZKs that could be used to achieve CCA security through the NY two key structure. Bellare and Sahai [4] gave relations between non-malleability [14] chosen ciphertext security.

Since then, different approaches to achieving CCA security have been proposed. Cramer and Shoup [12, 13] showed techniques for proving ciphertexts were well-structured and abstracted this into projective hash functions. Several other novel cryptosystems make use of specific number-theoretic techniques (e.g. [20, 9, 18]). Boneh, Canetti, Halevi and Katz [6] showed a generic method of achieving chosen ciphertext security from IBE systems. Peikert and Waters [26] gave a new avenue for achieving CCA security with the introduction of Lossy Trapdoor Functions (TDFs). Notably, this gave the first chosen ciphertext secure systems from lattice-based assumptions. Subsequently, various refinements of weaker conditions on the trapdoor functions were introduced [28, 21].

The above techniques are proven secure in the standard model. Bellare and Rogaway [3] show that in the random oracle model chosen ciphertext security can be built from chosen plaintext security.

2 Detectable Chosen Ciphertext Security

In this section, we define *detectable chosen ciphertext security*. An encryption scheme satisfying this definition is called a *detectable* encryption system. Our discussions assume a familiarity with CPA, CCA1 and CCA2 security as well as bounded CCA security and non-malleability. A reader wishing to review these definitions can find them in Appendix A.

2.1 Detectable Encryption

We define a *detectable* encryption scheme as having the usual algorithms (KeyGen, Enc, Dec), as defined in Definition A.1, together with an efficiently-computable boolean function F . Informally, F tests for a “detectable” relationship between two ciphertexts. The security game will mirror

that of CCA2 security, except that decryption queries in the second phase will not be answered for ciphertexts detectably-related to the challenge ciphertext. Our formal definition follows below.

Definition 2.1 (Detectable Encryption System) *A detectable encryption system is a tuple of probabilistic polynomial-time algorithms $(\text{KeyGen}, \text{Enc}, \text{Dec}, F)$ such that:*

1. $(\text{KeyGen}, \text{Enc}, \text{Dec})$ satisfy Definition A.1, where we sometimes denote $\text{Enc}(pk, m; r)$ as a deterministic function of the public key pk , the message m and randomness r , and
2. $F(pk, c', c) \rightarrow \{0, 1\}$: the detecting function F takes as input a public key pk and two ciphertexts c' and c , and outputs a bit.

Correctness is the same as a regular encryption system.

A detectable encryption system must have two properties, which we now define.

Unpredictability of the Detecting Function F . Informally, given the description of F and a public key pk , for an unknown ciphertext c , it should be hard to find a second ciphertext c' that is “related” to c ; i.e., such that $F(pk, c', c) = 1$. We consider both a basic and a strong formalization.

Basic Unpredictability Experiment. Consider the following experiment $\text{Exp}_{\mathcal{A}, \Pi}^{\text{predict.basic}}(\lambda)$ defined for a detectable encryption scheme $\Pi = (\text{KeyGen}, \text{Enc}, \text{Dec}, F)$ and an adversary $\mathcal{A}$:

1. Setup: $\text{KeyGen}(1^\lambda)$ is run to obtain keys (pk, sk) .
2. Queries: Adversary $\mathcal{A}$ is given pk and access to a decryption oracle $\text{Dec}(sk, \cdot)$. The adversary outputs a message m in the message space associated with pk and a ciphertext c in the ciphertext space associated with pk .
3. Challenge: A ciphertext $c^* \leftarrow \text{Enc}(pk, m)$ is computed.
4. Output: The output of the experiment is defined to be 1 if $F(pk, c^*, c)$, and 0 otherwise.

We also define a stronger variant $\text{Exp}_{\mathcal{A}, \Pi}^{\text{predict.strong}}(\lambda)$ of the unpredictability experiment where the adversary is additionally given sk . We observe that strong unpredictability implies basic unpredictability since the adversary can simulate the decryption oracle using the secret key.

Indistinguishability of Encryptions. Next, we formalize the confidentiality guarantee. Consider the following experiment $\text{Exp}_{\mathcal{A}, \Pi}^{\text{indist}}(\lambda)$ defined for a detectable encryption scheme $\Pi = (\text{KeyGen}, \text{Enc}, \text{Dec}, F)$ and an adversary $\mathcal{A}$:

1. Setup: $\text{KeyGen}(1^\lambda)$ is run to obtain keys (pk, sk) .
2. Phase 1: Adversary $\mathcal{A}$ is given pk and access to a decryption oracle $\text{Dec}(sk, \cdot)$. $\mathcal{A}$ outputs a pair of messages m_0, m_1 of the same length in the message space associated with pk .
3. Challenge: A random bit $b \leftarrow \{0, 1\}$ is chosen, and then a ciphertext $c^* \leftarrow \text{Enc}(pk, m_b)$ is computed and given to $\mathcal{A}$. We call c^* the challenge ciphertext.
4. Phase 2: $\mathcal{A}$ continues to have access to $\text{Dec}(sk, \cdot)$, but may not request a decryption of a ciphertext c such that $F(pk, c^*, c) = 1$. Finally, $\mathcal{A}$ outputs a bit b' .
5. Output: The output of the experiment is defined to be 1 if $b' = b$, and 0 otherwise.

Definition 2.2 (Detectable Chosen Ciphertext Security) A detectable encryption scheme $\Pi = (\text{KeyGen}, \text{Enc}, \text{Dec}, F)$ has an unpredictable detecting function and indistinguishable encryptions under a detectable chosen-ciphertext attack (or is DCCA-secure) if for all probabilistic polynomial-time adversaries $\mathcal{A}$ there exists a negligible function negl such that:

1. (F is unpredictable:) $\Pr[\text{Exp}_{\mathcal{A}, \Pi}^{\text{predict.basic}}(\lambda) = 1] \leq \text{negl}(\lambda)$ and
2. (Encryptions are indistinguishable:) $\Pr[\text{Exp}_{\mathcal{A}, \Pi}^{\text{indist}}(\lambda) = 1] \leq \frac{1}{2} + \text{negl}(\lambda)$.

2.2 Facts about DCCA Security

For space reasons, we omit the simple proofs of the first two claims. We conjecture that the converse of Claim 2.4 is not true. Indeed if the DDH assumption holds, then the CCA-1 secure Cramer-Shoup lite system would separate these two notions as discussed in the introduction.

Claim 2.3 (CCA2 $\implies$ DCCA) If $\Pi = (\text{KeyGen}, \text{Enc}, \text{Dec})$ is a CCA2-secure encryption scheme, then $\Pi' = (\text{KeyGen}, \text{Enc}, \text{Dec}, F)$ is a DCCA-secure encryption scheme where F outputs 0 on all inputs except those of the form $(\cdot, c, c)$.

Claim 2.4 (DCCA $\implies$ CCA1) If $\Pi = (\text{KeyGen}, \text{Enc}, \text{Dec}, F)$ is a DCCA-secure encryption scheme, then $\Pi' = (\text{KeyGen}, \text{Enc}, \text{Dec})$ is a CCA1-secure encryption scheme.

We also claim that one-bit DCCA-secure encryption implies arbitrary-length DCCA-secure encryption. Say $\Pi = (\text{KeyGen}, \text{Enc}, \text{Dec}, F)$ is a detectable encryption system with plaintext space $\{0, 1\}$. We can construct a new scheme $\Pi' = (\text{KeyGen}, \text{Enc}', \text{Dec}', F')$ with plaintext space $\{0, 1\}^*$ by defining Enc' as follows:

$$\text{Enc}'(pk, m) = \text{Enc}(pk, m_1), \dots, \text{Enc}(pk, m_n)$$

where $m = m_1 \dots m_n$. The decryption algorithm Dec' decrypts each ciphertext piece using Dec . The function F' performs n^2 invocations of F , testing each ciphertext piece of C with each ciphertext piece of C' , and outputting 1 if any invocation of F returned 1, and 0 otherwise.

Lemma 2.5 (1-bit DCCA encryption implies arbitrary-length DCCA encryption) Let Π and Π' be as above. If Π is DCCA-secure, then so is Π' .

We prove this lemma in Appendix B.

3 The Construction: CCA2 Security from DCCA Security

An overview of the techniques used for our construction is provided in Section 1.

The Construction Description We now construct a CCA2-secure public-key encryption scheme $\Pi = (\text{KeyGen}, \text{Enc}, \text{Dec})$ using three building blocks⁴:

⁴A 1-bounded CCA-secure encryption system is secure if an attacker makes at most one decryption query. One-bounded CCA security can be constructed from CPA security [25, 10]. See Appendix A. CPA security is trivially implied by DCCA security. Thus, there is really only one necessary building block: a DCCA-secure system.

1. a DCCA-secure encryption scheme, denoted $\Pi_{\text{dcca}} = (\text{KeyGen}_{\text{dcca}}, \text{Enc}_{\text{dcca}}, \text{Dec}_{\text{dcca}}, F)$.
2. a 1-bounded CCA-secure encryption scheme with perfect correctness, denoted $\Pi_{1\text{b-cca}} = (\text{KeyGen}_{1\text{b-cca}}, \text{Enc}_{1\text{b-cca}}, \text{Dec}_{1\text{b-cca}})$.
3. a CPA-secure encryption scheme with perfect correctness, denoted $\Pi_{\text{cpa}} = (\text{KeyGen}_{\text{cpa}}, \text{Enc}_{\text{cpa}}, \text{Dec}_{\text{cpa}})$.

We assume that the message space of each system is $\{0, 1\}^*$ and that messages of the form (x, y, z) can be uniquely and efficiently encoded as strings in $\{0, 1\}^*$, where the encoding length is the same for all inputs of the same length. We assume that λ bits will be sufficient randomness for the encryption algorithm of each system, where 1^λ is the security parameter. We assume that $\Pi_{1\text{b-cca}}$ and Π_{cpa} have perfect correctness for decryption. Finally, we assume that for Π_{dcca} the ciphertext length is a deterministic function of the security parameter and the message length. We discuss the justification behind these assumptions in Section B.

KeyGen(1^λ) Run $\text{KeyGen}_{\text{dcca}}(1^\lambda)$ to produce $(\text{PK}_{\text{in}}, \text{SK}_{\text{in}})$. Run $\text{KeyGen}_{1\text{b-cca}}(1^\lambda)$ to produce $(\text{PK}_A, \text{SK}_A)$. Run $\text{KeyGen}_{\text{cpa}}(1^\lambda)$ to produce $(\text{PK}_B, \text{SK}_B)$. Set the public key as $\text{PK} := (\text{PK}_{\text{in}}, \text{PK}_A, \text{PK}_B)$ and the secret key as $\text{SK} := (\text{SK}_{\text{in}}, \text{SK}_A, \text{SK}_B)$.

Enc(PK, M) The encryption algorithm first chooses three random strings $r_{\text{in}}, r_A, r_B \in \{0, 1\}^\lambda$. Next, it computes the ciphertext $\text{CT}_{\text{in}} := \text{Enc}_{\text{dcca}}(\text{PK}_{\text{in}}, (r_A, r_B, M); r_{\text{in}})$. It then treats this ciphertext as the message and computes $\text{CT}_A := \text{Enc}_{1\text{b-cca}}(\text{PK}_A, \text{CT}_{\text{in}}; r_A)$ and $\text{CT}_B := \text{Enc}_{\text{cpa}}(\text{PK}_B, \text{CT}_{\text{in}}; r_B)$. Finally, it outputs the encryption as $(\text{CT}_A, \text{CT}_B)$.

Dec(SK, CT) The decryption algorithm takes a ciphertext $\text{CT} := (\text{CT}_A, \text{CT}_B)$. It decrypts the first ciphertext as $\text{CT}_{\text{in}} := \text{Dec}_{1\text{b-cca}}(\text{SK}_A, \text{CT}_A)$. It then decrypts this output as $(r_A, r_B, M) := \text{Dec}_{\text{dcca}}(\text{SK}_{\text{in}}, \text{CT}_{\text{in}})$. It then checks that

$$\text{CT}_A = \text{Enc}_{1\text{b-cca}}(\text{PK}_A, \text{CT}_{\text{in}}; r_A) \text{ and } \text{CT}_B = \text{Enc}_{\text{cpa}}(\text{PK}_B, \text{CT}_{\text{in}}; r_B).$$

If all checks pass, it outputs M ; otherwise, it outputs $\perp$.

4 Proof of Security

We will now argue that the Section 3 construction is CCA2 secure, assuming the respective security properties of the underlying building blocks. To do so, it will be easier to consider a slight variant of the CCA2 security game, which we call *nested indistinguishability*, where the challenger either encrypts one of the two challenge messages or encrypts a string of zeros. The experiment involves three encryption schemes and combines them in the same manner as our main construction.

Nested Indistinguishability. Consider the experiment $\text{Exp}_{\mathcal{A}, \Pi_{\text{dcca}}, \Pi_{1\text{b-cca}}, \Pi_{\text{cpa}}}^{\text{nested}}(\lambda)$ defined for detectable encryption scheme Π_{dcca} , encryption schemes $\Pi_{1\text{b-cca}}$, Π_{cpa} and an adversary $\mathcal{A}$:

1. Setup: Run $\text{KeyGen}_{\text{dcca}}$, $\text{KeyGen}_{1\text{b-cca}}$ and $\text{KeyGen}_{\text{cpa}}$ to obtain key pairs $(\text{PK}_{\text{in}}, \text{SK}_{\text{in}})$, $(\text{PK}_A, \text{SK}_A)$ and $(\text{PK}_B, \text{SK}_B)$ respectively. Set $pk := (\text{PK}_{\text{in}}, \text{PK}_A, \text{PK}_B)$ and $sk := (\text{SK}_{\text{in}}, \text{SK}_A, \text{SK}_B)$.

2. Phase 1: Adversary $\mathcal{A}$ is given pk and access to a decryption oracle $\text{Dec}(sk, \cdot)$, which executes the decryption algorithm as defined in Section 3. $\mathcal{A}$ outputs a pair of messages m_0, m_1 of the same length in the message space associated with pk .
3. Challenge: Randomness $\beta, z \leftarrow \{0, 1\}$ and $r_A, r_B \leftarrow \{0, 1\}^\lambda$ are chosen. Let ℓ denote the length of the encoding of (r_A, r_B, m_β) . Then compute:

$$\text{CT}_{\text{in}}^* := \begin{cases} \text{Enc}_{\text{dcca}}(\text{PK}_{\text{in}}, (r_A, r_B, m_\beta)) & \text{if } z = 0; \\ \text{Enc}_{\text{dcca}}(\text{PK}_{\text{in}}, 0^\ell) & \text{if } z = 1. \end{cases} \quad (1)$$

Next compute $\text{CT}_A^* := \text{Enc}_{\text{1b-cca}}(\text{PK}_A, \text{CT}_{\text{in}}^*; r_A)$ and $\text{CT}_B^* := \text{Enc}_{\text{cpa}}(\text{PK}_B, \text{CT}_{\text{in}}^*; r_B)$. Return to $\mathcal{A}$ the ciphertext $\text{CT}^* := (\text{CT}_A^*, \text{CT}_B^*)$.

4. Phase 2: $\mathcal{A}$ continues to have access to $\text{Dec}(sk, \cdot)$, but may not request a decryption of the challenge ciphertext CT^* . Finally, $\mathcal{A}$ outputs a bit z' .
5. Output: The output of the experiment is defined to be 1 if $z' = z$, and 0 otherwise.

Definition 4.1 (Nested Indistinguishability) A tuple of systems $(\Pi_{\text{dcca}}, \Pi_{\text{1b-cca}}, \Pi_{\text{cpa}})$ has nested indistinguishable encryptions under a chosen-ciphertext attack if for all probabilistic polynomial-time adversaries $\mathcal{A}$ there exists a negligible function negl such that:

$$\Pr[\text{Exp}_{\mathcal{A}, \Pi_{\text{dcca}}, \Pi_{\text{1b-cca}}, \Pi_{\text{cpa}}}^{\text{nested}}(\lambda) = 1] \leq \frac{1}{2} + \text{negl}(\lambda).$$

It is important to observe that the nested indistinguishability experiment combines $\Pi_{\text{dcca}}, \Pi_{\text{1b-cca}}, \Pi_{\text{cpa}}$ in exactly the same manner as the Section 3 construction. When $z = 1$, it encrypts “properly” and when $z = 0$, it encrypts all zeros.

With a goal of proving CCA2 security, our main task is to argue that our Section 3 construction provides nested indistinguishability. To do this, we must first establish that a certain event does not happen, except with negligible probability. We define this event as follows.

Definition 4.2 (The Bad Query Event) Let $\Pi_{\text{dcca}}, \Pi_{\text{1b-cca}}$, and Π_{cpa} be the schemes parameterizing the experiment $\text{Exp}^{\text{nested}}$. Let PK_{in} be the public key output by running $\text{KeyGen}_{\text{det}}$ during the course of the experiment. We say that a bad query event has occurred during an execution of this experiment if in Phase 2, the adversary $\mathcal{A}$ makes a decryption query of the form $\text{CT} := (\text{CT}_A, \text{CT}_B)$ such that

- (Query inner is “related” to challenge inner:) $F(\text{PK}_{\text{in}}, \text{CT}_{\text{in}}^*, \text{Dec}_{\text{1b-cca}}(\text{SK}_A, \text{CT}_A)) = 1$, and
- (Query ciphertext differs from challenge ciphertext in first half): $\text{CT}_A^* \neq \text{CT}_A$.

where $\text{CT}^* := (\text{CT}_A^*, \text{CT}_B^*)$ is the challenge ciphertext and CT_A^* is an encryption of CT_{in}^* . We note that this event is well defined in both the cases where $z = 0$ and $z = 1$.

4.1 Proof that Bad Query Event Does Not Happen

Claim 4.3 (No Bad Query Event when $z = 1$ (all zeros encrypted)) Suppose that Π_{dcca} is DCCA secure, $\Pi_{\text{1b-cca}}$ is 1-bounded CCA secure, and Π_{cpa} is CPA secure, all with perfect correctness. Then for all probabilistic polynomial-time adversaries $\mathcal{A}$, during a run of experiment $\text{Exp}_{\mathcal{A}, \Pi_{\text{dcca}}, \Pi_{\text{1b-cca}}, \Pi_{\text{cpa}}}^{\text{nested}}(\lambda)$ with $z = 1$, a bad query event does not take place except with negligible probability in λ where the probability is taken over the coins of the adversary and the experiment.

Proof. We proceed via a series of hybrids. Let BQE denote a bad query event.

Step 1: $\Pr[\text{BQE in Nested}] \sim \Pr[\text{BQE in Right-Erased}]$ from CPA-security of Π_{cpa} . We first define a variation of the nested indistinguishability experiment with $z = 1$, which we call the *right-erased* experiment. In this experiment, CT_B^* is formed as $\text{CT}_B^* := \text{Enc}_{\text{cpa}}(\text{PK}_B, 1^k; r_B)$ where k denotes the length of CT_{in}^* . CT_A^* is formed the same as in the nested indistinguishability experiment with $z = 1$. We suppose there exists a PPT adversary $\mathcal{A}$ for the nested indistinguishability experiment which causes the bad query event to occur with non-negligibly different probability in the usual experiment with $z = 1$ compared to the right-erased experiment. We construct a PPT algorithm $\mathcal{B}$ which violates the CPA-security of Π_{cpa} .

$\mathcal{B}$ is given PK_B . $\mathcal{B}$ then runs $\text{KeyGen}_{\text{dcca}}$ and $\text{KeyGen}_{1\text{b-cca}}$ for itself to produce $\text{PK}_{\text{in}}, \text{SK}_{\text{in}}$ and PK_A, SK_A respectively. It gives $\mathcal{A}$ $pk = (\text{PK}_{\text{in}}, \text{PK}_A, \text{PK}_B)$. $\mathcal{B}$ can simulate the decryption oracle $\text{Dec}(sk, \cdot)$ for $\mathcal{A}$ by running the usual decryption algorithm (note that this does not require SK_B).

The adversary $\mathcal{A}$ outputs a pair of messages m_0, m_1 of the same length in the message space associated with pk . $\mathcal{B}$ chooses $r_A \in \{0, 1\}^\lambda$ and computes $\text{CT}_{\text{in}}^* = \text{Enc}_{\text{dcca}}(\text{PK}_{\text{in}}, 0^\ell)$, where ℓ is the length of the encoding of (r_A, r_A, m_0) . It then computes $\text{CT}_A^* = \text{Enc}_{1\text{b-cca}}(\text{PK}_A, \text{CT}_{\text{in}}^*; r_A)$. It submits CT_{in}^* and 1^k to its challenger as its two messages. It receives CT_B^* as the ciphertext. It gives $\text{CT}^* := (\text{CT}_A^*, \text{CT}_B^*)$ to $\mathcal{A}$.

To respond to remaining decryption queries $\mathcal{A}$ makes, $\mathcal{B}$ runs the usual decryption algorithm (after checking that the query is not equal to the challenge ciphertext). In addition, $\mathcal{B}$ checks for the bad query event by first checking if $\text{CT}_A \neq \text{CT}_A^*$ and then computing $F(\text{PK}_{\text{in}}, \text{CT}_{\text{in}}^*, \text{Dec}_{1\text{b-cca}}(\text{SK}_A, \text{CT}_A))$. We recall that $\mathcal{B}$ generated SK_A, PK_A for itself, so it can compute $\text{Dec}_{1\text{b-cca}}(\text{SK}_A, \text{CT}_A)$.

If CT_B^* is an encryption of CT_{in}^* , then $\mathcal{B}$ has properly simulated the usual experiment with $z = 1$. If it is instead an encryption of 1^k , then $\mathcal{B}$ has properly simulated the right-erased experiment. We note that the bad query event occurs in the simulation if and only if it is detected by $\mathcal{B}$.

We let ϵ denote the probability that the bad query event occurs in the usual experiment with $z = 1$ and δ denote this probability in the right-erased experiment. We suppose $\epsilon - \delta$ is positive and non-negligible (the opposite case is analogous). Now, if $\mathcal{B}$ detects the bad query event, it guesses that CT_A^* is an encryption of CT_{in}^* . Otherwise, it guesses the opposite. $\mathcal{B}$'s probability of guessing correctly in the CPA security game for Π_{cpa} is then equal to $\frac{\epsilon}{2} + \frac{1}{2}(1 - \delta) = \frac{1}{2} + \frac{1}{2}(\epsilon - \delta)$. The quantity $\epsilon - \delta$ is non-negligible, so $\mathcal{B}$ violates the CPA-security of Π_{cpa} . Hence we may conclude that the probability of the bad query event happening in the usual experiment with $z = 1$ is the same (up to a negligible difference) as the probability of the bad query event happening in the right-erased experiment for any PPT adversary.

Step 2: $\Pr[\text{BQE in Full-Erased}]$ is negligible from the unpredictability of the detecting function of Π_{dcca} . We now define an additional variation of the experiment, which we call the *full-erased* experiment. This is like the right-erased experiment, except that CT_A^* is also an encryption of 1^k , instead of an encryption of CT_{in}^* . We claim that in the full-erased experiment, the bad query event can only occur with negligible probability. To see this, we suppose we have a PPT adversary $\mathcal{A}$ which causes the bad query event to occur with non-negligible probability in the full-erased experiment. We will build a PPT adversary $\mathcal{B}$ for the basic unpredictability experiment which violates unpredictability of the detecting function for Π_{dcca} .

$\mathcal{B}$ is given PK_{in} and access to a decryption oracle $\text{Dec}(\text{SK}_{\text{in}}, \cdot)$. It runs $\text{KeyGen}_{1\text{b-cca}}$ and $\text{KeyGen}_{\text{cpa}}$ for itself to produce PK_A, SK_A and PK_B, SK_B . It gives $(\text{PK}_{\text{in}}, \text{PK}_A, \text{PK}_B)$ to $\mathcal{A}$. $\mathcal{B}$ can simulate the decryption oracle for $\mathcal{A}$ using SK_A and its own decryption oracle. $\mathcal{A}$ outputs m_0, m_1 . $\mathcal{B}$ then computes $\text{CT}_A^* = \text{Enc}_{1\text{b-cca}}(\text{PK}_A, 1^k)$ and $\text{CT}_B^* = \text{Enc}_{\text{cpa}}(\text{PK}_B, 1^k)$ and gives

$CT^* = (CT_A^*, CT_B^*)$ to $\mathcal{A}$. We let q denote the number of Phase 2 queries made by $\mathcal{A}$. $\mathcal{B}$ can respond to these queries as before. $\mathcal{B}$ chooses a random $i \in \{1, 2, \dots, q\}$ and a random bit $b \in \{0, 1\}$. It takes the i^{th} Phase 2 query of $\mathcal{A}$, denoted by (CT_A^i, CT_B^i) , and computes $CT_{in}^i = \text{Dec}_{1b-cca}(SK_A, CT_A^i)$. It submits m_b and CT_{in}^i to its challenger. Then, the distribution of $c^* = \text{Enc}_{dcca}(PK_{in}, m_b)$ in the basic unpredictability experiment is precisely the distribution of CT_{in}^* . Hence, the bad query event for query i corresponds to an output of 1 for basic unpredictability experiment. Thus, if the bad query event occurs with some non-negligible probability ϵ , $\mathcal{B}$ will cause an output of 1 in the basic unpredictability experiment with probability at least $\frac{\epsilon}{q}$, which is non-negligible.

Step 3: $\Pr[\text{BQE in Right-Erased}] \sim \Pr[\text{BQE in Full-Erased}]$ from the 1-bounded CCA security of Π_{1b-cca} . We now return to considering a PPT adversary $\mathcal{A}$ in the right-erased experiment. We let q denote the number of Phase 2 queries made by $\mathcal{A}$. We suppose that $\mathcal{A}$ causes the bad query event with non-negligible probability. Then there exists some index $i \in \{1, \dots, q\}$ such that $\mathcal{A}$ causes the bad query event to occur with non-negligible probability on its i^{th} Phase 2 query. In other words, if there exists a PPT adversary $\mathcal{A}$ for which the bad query event occurs with non-negligible probability in the right-erased experiment, then for each value of the security parameter, there exists an index i such that $\mathcal{A}$ causes the BQE to occur on its i^{th} Phase 2 query with non-negligible probability. We note that for *any* i , the probability that $\mathcal{A}$ causes the BQE to occur on its i^{th} Phase 2 query in the full-erased experiment is negligible, as we proved above.

We fix such an i , and we define a PPT algorithm $\mathcal{B}$ which violates the 1-bounded CCA security of Π_{1b-cca} . $\mathcal{B}$ receives PK_A from its challenger. It runs KeyGen_{dcca} and KeyGen_{cpa} for itself to produce PK_{in}, SK_{in} and PK_B, SK_B . It gives (PK_{in}, PK_A, PK_B) to $\mathcal{A}$ as the public key.

$\mathcal{B}$ simulates the decryption oracle for $\mathcal{A}$ as follows. Upon receiving a ciphertext (CT_A, CT_B) , $\mathcal{B}$ decrypts CT_B using Dec_{cpa} with SK_B , and we let CT_{in} denote the output. It then decrypts CT_{in} using Dec_{dcca} with SK_{in} , and parses the output as r_A, r_B, M . It checks if $CT_A = \text{Enc}_{1b-cca}(PK_A, CT_{in}; r_A)$ and if $CT_B = \text{Enc}_{cpa}(PK_B, CT_{in}; r_B)$. If both checks pass, it outputs M . Else, it outputs $\perp$.

We claim that this matches the output of the usual decryption algorithm, even though $\mathcal{B}$ is first decrypting CT_B instead of CT_A . To see this, note that the outputs are clearly the same whenever $\text{Dec}_{1b-cca}(CT_A, SK_A) = \text{Dec}_{cpa}(CT_B, SK_B)$. Whenever these are unequal, both decryption methods will output $\perp$. This is because $CT_A = \text{Enc}_{1b-cca}(PK_A, CT_{in}; r_A)$ and $CT_B = \text{Enc}_{cpa}(PK_B, CT_{in}; r_B)$ imply that $\text{Dec}_{1b-cca}(CT_A, SK_A) = CT_{in} = \text{Dec}_{cpa}(CT_B, SK_B)$. (Recall here that we have assumed Π_{1b-cca} and Π_{cpa} have perfect correctness.)

At some point, $\mathcal{A}$ outputs m_0, m_1 . $\mathcal{B}$ forms $CT_{in}^* = \text{Enc}_{dcca}(PK_{in}, 0^\ell)$ and $CT_B^* = \text{Enc}_{cpa}(PK_B, 1^k)$. It outputs the messages CT_{in}^* and 1^k to its challenger, and receives a ciphertext which it sets as CT_A^* . It gives the ciphertext (CT_A^*, CT_B^*) to $\mathcal{A}$. It can then respond to $\mathcal{A}$'s Phase 2 decryption queries in the same way as before. When it receives the i^{th} Phase 2 query of $\mathcal{A}$, denoted by (CT_A^i, CT_B^i) , $\mathcal{B}$ checks for the bad query event by first checking if $CT_A^i \neq CT_A^*$ and if so, submitting CT_A^i as its one decryption query to its decryption oracle for PK_A . It can compute $F(PK_{in}, CT_{in}^*, \text{Dec}(SK_A, CT_A^i))$. This equals 1 if and only if the bad query event has occurred for query i , and in this case $\mathcal{B}$ guesses that CT_A^* is an encryption of CT_{in}^* . Otherwise, $\mathcal{B}$ guesses the opposite.

We observe that when CT_A^* is an encryption of CT_{in}^* , then $\mathcal{B}$ has properly simulated the right-erased experiment, and when CT_A^* is an encryption of 0^k , then $\mathcal{B}$ has properly simulated the full-erased experiment. We let ϵ denote the non-negligible probability that $\mathcal{A}$ causes the bad query event to occur on (Phase 2) query i in the right-erased experiment, and we let δ denote the corresponding probability for the full-erased experiment. We know that δ must be negligible, therefore $\epsilon - \delta$ is

positive and non-negligible. The probability that $\mathcal{B}$ guesses correctly is: $\frac{1}{2}(1-\delta) + \frac{1}{2}\epsilon = \frac{1}{2} + \frac{1}{2}(\epsilon - \delta)$, so $\mathcal{B}$ achieves a non-negligible advantage in the 1-bounded CCA security game for Π_{1b-cca} .

Thus, it must be the case that for all PPT algorithms $\mathcal{A}$, the BQE occurs with only negligible probability in the right-erased experiment, and hence also in the nested experiment with $z = 1$. $\square$

Claim 4.4 (No Bad Query Event when $z = 0$ (real message encrypted)) *As a consequence of Claim 4.3 and the DCCA security of Π_{dcca} , it holds that for all probabilistic polynomial-time adversaries $\mathcal{A}$, during a run of experiment $\text{Exp}_{\mathcal{A}, \Pi_{dcca}, \Pi_{1b-cca}, \Pi_{cpa}}^{\text{nested}}(\lambda)$ with $z = 0$, a bad query event does not take place except with negligible probability in λ where the probability is taken over the coins of the adversary and the experiment.*

Proof. In Claim 4.3, we established that bad query events happen with at most negligible probability when $z = 1$. We will use this fact to argue that they cannot happen much more frequently when $z = 0$. Suppose to the contrary that there exists a PPT adversary $\mathcal{A}$ that forces bad query events to happen with non-negligible probability ϵ when $z = 0$. We create an PPT adversary $\mathcal{B}$ who interacts with $\mathcal{A}$ in a run of the nested indistinguishability experiment to break the DCCA security of Π_{dcca} with detecting function F with probability negligibly-close to $\frac{1}{2} + \frac{\epsilon}{2}$ as follows:

1. Setup: $\mathcal{B}$ obtains PK_{in} from the $\text{Exp}^{\text{indist}}$ challenger. It runs KeyGen_{1b-cca} to obtain $(\text{PK}_A, \text{SK}_A)$ and KeyGen_{cpa} to obtain $(\text{PK}_B, \text{SK}_B)$.
2. Phase 1: $\mathcal{B}$ gives to $\mathcal{A}$ the public key $\text{PK} = (\text{PK}_{\text{in}}, \text{PK}_A, \text{PK}_B)$. When $\mathcal{A}$ queries the decryption oracle on CT, $\mathcal{B}$ can simulate the normal decryption algorithm using SK_A and the phase 1 oracle $\text{Dec}(\text{SK}_{\text{in}}, \cdot)$. Eventually, $\mathcal{A}$ outputs a pair of messages m_0, m_1 .
3. Challenge: Choose random $\beta \in \{0, 1\}$ and $r_A, r_B \in \{0, 1\}^\lambda$. Send to the $\text{Exp}^{\text{indist}}$ challenger the messages $M_0 = (r_A, r_B, m_\beta)$ and $M_1 = 0^{|M_0|}$, and obtain from this challenger the ciphertext CT_{in}^* . Compute $\text{CT}_A^* := \text{Enc}_{1b-cca}(\text{PK}_A, \text{CT}_{\text{in}}^*; r_A)$ and $\text{CT}_B^* := \text{Enc}_{cpa}(\text{PK}_B, \text{CT}_{\text{in}}^*; r_B)$. Return $\text{CT}^* := (\text{CT}_A^*, \text{CT}_B^*)$ to $\mathcal{A}$.
4. Phase 2: When $\mathcal{A}$ queries the decryption oracle on $\text{CT} := (\text{CT}_A, \text{CT}_B)$, compute $\text{CT}_{\text{in}} := \text{Dec}_{1b-cca}(\text{SK}_A, \text{CT}_A)$. If
 - (a) Case 1 (a bad query event): $\text{CT}_A \neq \text{CT}_A^*$ and yet $F(\text{PK}_{\text{in}}, \text{CT}_{\text{in}}^*, \text{CT}_{\text{in}}) = 1$, then abort and output the bit 0.
 - (b) Case 2 (partial match with challenge): $\text{CT}_A = \text{CT}_A^*$, then return $\perp$ to $\mathcal{A}$.

Otherwise, query the phase 2 oracle, $\text{Dec}(\text{SK}_{\text{in}}, \cdot)$, to decrypt CT_{in} , and return its response to $\mathcal{A}$.

5. Output: When $\mathcal{A}$ outputs a bit, $\mathcal{B}$ echos the bit as its output.

Analysis. We begin our analysis by arguing that $\mathcal{B}$ correctly answers all decryption queries except when it aborts. First, we show that a partial match with the challenge, causing the $\perp$ response in Case 2, is correct because that query must be invalid. Since a decryption query on the challenge is forbidden by the experiment, if $\text{CT}_A = \text{CT}_A^*$, then $\text{CT}_B \neq \text{CT}_B^*$. However, we argue that this must be an invalid ciphertext, i.e., one on which the main construction's decryption algorithm would return $\perp$. We see this as follows. Since decryption is deterministic, we have

$T := \text{Dec}_{1b-cca}(\text{SK}_A, \text{CT}_A) = \text{Dec}_{1b-cca}(\text{SK}_A, \text{CT}_A^*)$ and $(r_A, r_B, m) := \text{Dec}_{dcca}(\text{SK}_{in}, T)$. By the checks enforced by the main construction's decryption algorithm, there is only one "second half" that matches $\text{CT}_A = \text{CT}_A^*$, that is $\text{Enc}_{cpa}(\text{PK}_B, T; r_B)$. Since the challenge is a valid ciphertext, CT_B^* must be this value and CT_B must cause an error.

When neither Case 1 or Case 2 applies in phase 2, the inner decryption query will succeed since the ciphertext is not detectably related to the challenge. This allows $\mathcal{B}$ to respond correctly.

When a bad query event occurs in Phase 2, $\mathcal{B}$ cannot query $\text{Exp}^{\text{indist}}$'s decryption oracle to decrypt the ciphertext. At first glance, one seems stuck. However, we assumed bad query events happen only when $z = 0$ with all but negligible probability. Thus, $\mathcal{B}$ can guess that $\mathcal{A}$ thinks $z = 0$, which corresponds to M_0 being encrypted in our reduction. Thus, $\mathcal{B}$ can abort and guess 0 at this point.

When $\mathcal{B}$ aborts, it causes the $\text{Exp}^{\text{indist}}$ experiment to output 1 with high probability. When $\mathcal{B}$ does not abort, it causes $\text{Exp}^{\text{indist}}$ experiment to output 1 with probability $\frac{1}{2}$. Since $\mathcal{B}$ aborts with non-negligible probability ϵ when $z = 0$, then $\mathcal{B}$ causes the experiment's output to be 1 with probability non-negligibly greater than $\frac{1}{2}$. $\square$

4.2 Putting the Proof of the Main Theorem Together

Theorem 4.5 (Main Construction is Nested Indistinguishable) *Our main construction in Section 3, comprised of the three building blocks Π_{dcca} , Π_{1b-cca} , Π_{cpa} , has nested indistinguishable encryptions under a chosen-ciphertext attack under the assumptions that Π_{dcca} is DCCA secure, Π_{1b-cca} is 1-bounded CCA secure, and Π_{cpa} is CPA secure, all with perfect correctness.*

Proof of Theorem 4.5 is given in Appendix C. The crux of the argument is that bad query events do not happen (except with negligible probability). This was already established in Claims 4.3 and 4.4. Armed with this fact, we can prove the nested indistinguishability of the main construction based on the indistinguishability property of the DCCA-security of Π_{dcca} . The reduction and its analysis are similar to those in the proof of Claim 4.4.⁵

The following corollary follows from Theorem 4.5. Informally, if the adversary cannot distinguish an encryption of a message from an encryption of zeros, then she also cannot distinguish between the encryptions of two different messages.

Corollary 4.6 (Main Construction is CCA2 Secure) *Our main construction in Section 3, comprised of the three building blocks Π_{dcca} , Π_{1b-cca} , Π_{cpa} , is CCA2 secure under the assumptions that Π_{dcca} is DCCA secure, Π_{1b-cca} is 1-bounded CCA secure, and Π_{cpa} is CPA secure, all with perfect correctness.*

5 Why not use CCA1?

We now consider using an arbitrary CCA1-secure scheme in place of the DCCA-secure scheme in our construction in Section 3. To give intuition about why we *believe* this approach fails in general

⁵We note that we alternatively could have merged the proofs of Claim 4.4 and Theorem 4.5. However, we chose to keep the bad event analysis separate for pedagogical purposes at the expense of some redundancy in the description of the related reductions.

to provide a CCA2-secure scheme, we define the following oracle. This oracle enables a CCA2 attack on the construction, without *appearing* to break the CCA1 security of the inner scheme or the 1-bounded CCA security/CPA security of the outer schemes.

Oracle The oracle takes in the public key (consisting of the three public keys for the three building blocks) and a ciphertext CT . It runs the decryption algorithm of the construction on CT . If the decryption algorithm outputs a message M , then the oracle runs the encryption algorithm to produce a new ciphertext $\widetilde{CT}$ encrypting M . If the decryption algorithm outputs $\perp$ (indicating that the ciphertext was malformed), the oracle encrypts a string of 0's of the appropriate length to produce the new ciphertext $\widetilde{CT}$. (The length of the 0 string is chosen so that the inner ciphertext has the same size as CT_{in} for CT .) The oracle outputs $\widetilde{CT}$.

A Chosen Ciphertext Attack on the System. Using the oracle, an attacker can violate the CCA2 security of the construction as follows. Upon receiving the challenge ciphertext CT^* , the attacker sends this to the oracle to obtain a ciphertext $\widetilde{CT}^*$ encrypting the same message. Since $\widetilde{CT}^* \neq CT^*$, it can query $\widetilde{CT}^*$ to its decryption oracle in the CCA2 security game. It receives the message, thereby violating security.

Why Security of the Underlying Primitives Remains. Intuitively, the oracle should only be useful to an attacker who still has access to the decryption oracle in the security game. This does not violate CCA1 security of the inner scheme, since in the CCA1 security game the attacker loses access to the decryption oracle completely after receiving the challenge ciphertext. It is important to note here that the oracle's output does not give away whether the ciphertext it received was malformed. The oracle also does not break the 1-bounded CCA and CPA security guarantees of the outer encryption schemes, because even though the attacker may use its one query for the 1-bounded CCA-secure scheme after seeing the challenge ciphertext, it will not know the randomness used in the challenge encryption, and hence cannot create from it a well-formed ciphertext. Since the oracle runs the usual decryption algorithm which checks that the ciphertext is well-formed, it will not be useful to the attacker attempting to break security of the outer schemes.

Open Questions. This oracle is quite strong, and this leaves some remaining questions. First, might there be a useful notion between CCA1 security and DCCA security for the inner building block that would suffice for the outer scheme to imply CCA2 security for our construction? Also, it would be interesting to construct a more concrete counterexample to CCA2 security for our construction with a CCA1-secure inner scheme.

Acknowledgments

The authors thank Steven Myers and the anonymous reviewers for helpful comments.

References

- [1] Jee Hea An, Yevgeniy Dodis, and Tal Rabin. On the security of joint signature and encryption. In *EUROCRYPT*, volume 2332, pages 83–107, 2002.

- [2] Mihir Bellare, Zvika Brakerski, Moni Naor, Thomas Ristenpart, Gil Segev, Hovav Shacham, and Scott Yilek. Hedged public-key encryption: How to protect against bad randomness. In *ASIACRYPT*, pages 232–249, 2009.
- [3] Mihir Bellare and Phillip Rogaway. Random oracles are practical: A paradigm for designing efficient protocols. In *ACM Conference on Computer and Communications Security*, pages 62–73, 1993.
- [4] Mihir Bellare and Amit Sahai. Non-malleable encryption: Equivalence between two notions, and an indistinguishability-based characterization. In *CRYPTO*, pages 519–536, 1999.
- [5] Manuel Blum, Paul Feldman, and Silvio Micali. Non-interactive zero-knowledge and its applications (extended abstract). In *STOC*, pages 103–112, 1988.
- [6] Dan Boneh, Ran Canetti, Shai Halevi, and Jonathan Katz. Chosen-ciphertext security from identity-based encryption. *SIAM J. Comput.*, 36(5):1301–1328, 2007.
- [7] Ran Canetti, Shai Halevi, and Jonathan Katz. Chosen-ciphertext security from identity-based encryption. In *EUROCRYPT*, pages 207–222, 2004.
- [8] Ran Canetti, Hugo Krawczyk, and Jesper Buus Nielsen. Relaxing chosen-ciphertext security. In *CRYPTO*, volume 2729, pages 565–582, 2003.
- [9] David Cash, Eike Kiltz, and Victor Shoup. The twin diffie-hellman problem and applications. In *EUROCRYPT*, pages 127–145, 2008.
- [10] Seung Geol Choi, Dana Dachman-Soled, Tal Malkin, and Hoeteck Wee. Black-box construction of a non-malleable encryption scheme from any semantically secure one. In *TCC*, pages 427–444, 2008.
- [11] Ronald Cramer, Goichiro Hanaoka, Dennis Hofheinz, Hideki Imai, Eike Kiltz, Rafael Pass, Abhi Shelat, and Vinod Vaikuntanathan. Bounded cca2-secure encryption. In *ASIACRYPT*, pages 502–518, 2007.
- [12] Ronald Cramer and Victor Shoup. A practical public key cryptosystem provably secure against adaptive chosen ciphertext attack. In *CRYPTO*, pages 13–25, 1998.
- [13] Ronald Cramer and Victor Shoup. Universal hash proofs and a paradigm for adaptive chosen ciphertext secure public-key encryption. In *EUROCRYPT*, pages 45–64, 2002.
- [14] Danny Dolev, Cynthia Dwork, and Moni Naor. Non-malleable cryptography (extended abstract). In *STOC*, pages 542–552, 1991.
- [15] Yael Gertner, Tal Malkin, and Omer Reingold. On the impossibility of basing trapdoor functions on trapdoor predicates. In *FOCS*, pages 126–135, 2001.
- [16] Shafi Goldwasser and Silvio Micali. Probabilistic encryption. *J. Comput. Syst. Sci.*, 28(2):270–299, 1984.
- [17] Dennis Hofheinz and Eike Kiltz. Secure hybrid encryption from weakened key encapsulation. In *CRYPTO*, volume 4622, pages 553–571, 2007.

- [18] Dennis Hofheinz and Eike Kiltz. Practical chosen ciphertext secure encryption from factoring. In *EUROCRYPT*, pages 313–332, 2009.
- [19] Jonathan Katz and Yehuda Lindell. *Introduction to Modern Cryptography*. Chapman & Hall, CRC, 2007.
- [20] Eike Kiltz. Chosen-ciphertext security from tag-based encryption. In *TCC*, 2006.
- [21] Eike Kiltz, Payman Mohassel, and Adam O’Neill. Adaptive trapdoor functions and chosen-ciphertext security. In *EUROCRYPT*, pages 673–692, 2010.
- [22] Philip D. MacKenzie, Michael K. Reiter, and Ke Yang. Alternatives to non-malleability: Definitions, constructions, and applications (extended abstract). In *TCC*, pages 171–190, 2004.
- [23] Steven Myers and Abhi Shelat. Bit encryption is complete. In *FOCS*, pages 607–616, 2009.
- [24] Moni Naor and Moti Yung. Public-key cryptosystems provably secure against chosen ciphertext attacks. In *STOC*, pages 427–437, 1990.
- [25] Rafael Pass, Abhi Shelat, and Vinod Vaikuntanathan. Construction of a non-malleable encryption scheme from any semantically secure one. In *CRYPTO*, pages 271–289, 2006.
- [26] Chris Peikert and Brent Waters. Lossy trapdoor functions and their applications. In *STOC*, pages 187–196, 2008.
- [27] Charles Rackoff and Daniel R. Simon. Non-interactive zero-knowledge proof of knowledge and chosen ciphertext attack. In *CRYPTO*, pages 433–444, 1991.
- [28] Alon Rosen and Gil Segev. Chosen-ciphertext security via correlated products. In *TCC*, pages 419–436, 2009.
- [29] Amit Sahai. Non-malleable non-interactive zero knowledge and adaptive chosen-ciphertext security. In *FOCS*, pages 543–553, 1999.
- [30] Victor Shoup. A proposal for an iso standard for public key encryption. Cryptology ePrint Archive, Report 2001/112, 2001. <http://eprint.iacr.org/>.

A Background on Security Definitions for Encryption

There is a rich body of literature on how to formalize the confidentiality guarantee of encryption as we discussed in Section 1.1. We define several of these concepts here.

First, we recall the algorithms comprising an encryption system.

Definition A.1 (Encryption System) *An encryption system is a tuple of probabilistic polynomial-time algorithms $(\text{KeyGen}, \text{Enc}, \text{Dec})$ such that:*

1. $\text{KeyGen}(1^\lambda) \rightarrow (pk, sk)$: the key generation algorithm takes as input the security parameter 1^λ and outputs a pair of keys (pk, sk) .

2. $\text{Enc}(pk, m) \rightarrow c$: the encryption algorithm takes as input a public key pk and a message m from some underlying plaintext space and outputs a ciphertext c . Enc is a probabilistic algorithm, although we will sometimes cast it as a deterministic algorithm with an explicit random input, r , by writing

$$c := \text{Enc}(pk, m; r).$$

3. $\text{Dec}(sk, c) \rightarrow m$: the decryption algorithm takes as input a secret key sk and a ciphertext c , and outputs a message m or a special symbol $\perp$ denoting failure. Wlog, we assume that this algorithm is deterministic and write $m := \text{Dec}(sk, c)$.

For the system to be correct, we require that $\text{Dec}(sk, \text{Enc}(pk, m)) = m$, except with negligible probability over (pk, sk) output by $\text{KeyGen}(1^\lambda)$ and any randomness used by Enc .

CCA Security Experiment. We now recall the definition of CCA Security. Consider the following experiment $\text{Exp}_{\mathcal{A}, \Pi}^{\text{cca}}(\lambda)$ defined for public-key encryption scheme $\Pi = (\text{KeyGen}, \text{Enc}, \text{Dec})$ and an adversary $\mathcal{A}$:

1. Setup: $\text{KeyGen}(1^\lambda)$ is run to obtain keys (pk, sk) .
2. Phase 1: Adversary $\mathcal{A}$ is given pk and access to a decryption oracle $\text{Dec}(sk, \cdot)$. The adversary outputs a pair of messages m_0, m_1 of the same length in the message space associated with pk .
3. Challenge: A random bit $b \leftarrow \{0, 1\}$ is chosen, and then a ciphertext $c^* \leftarrow \text{Enc}(pk, m_b)$ is computed and given to $\mathcal{A}$. We call c^* the challenge ciphertext.
4. Phase 2: $\mathcal{A}$ continues to have access to $\text{Dec}(sk, \cdot)$ provided he does not request a decryption of c^* . Finally, $\mathcal{A}$ outputs a bit b' .
5. Output: The output of the experiment is defined to be 1 if $b' = b$, and 0 otherwise.

Definition A.2 (CCA Security [27]) A public-key encryption scheme $\Pi = (\text{KeyGen}, \text{Enc}, \text{Dec})$ has indistinguishable encryptions under a chosen-ciphertext attack (or is CCA-secure) if for all probabilistic polynomial-time adversaries $\mathcal{A}$ there exists a negligible function negl such that:

$$\Pr[\text{Exp}_{\mathcal{A}, \Pi}^{\text{cca}}(\lambda) = 1] \leq \frac{1}{2} + \text{negl}(\lambda).$$

We also consider the following variants of the above definition:

1. **Chosen Plaintext Attack (CPA) Security [16]:** same as above, except that $\mathcal{A}$ is not given access to the decryption oracle in phase 1 or phase 2.
2. **Lunchtime or CCA1 Security [24]:** same as above, except that $\mathcal{A}$ is not given access to the decryption oracle in phase 2.
3. **q -Bounded CCA Security [11]:** same as above, except that the total number of decryption queries made by $\mathcal{A}$ in phase 1 and phase 2 is at most q . In this work, we will use 1-bounded CCA (a.k.a., “one-time CCA”) security, where $\mathcal{A}$ can make a single decryption query.

Realizations We note that CPA security implies 1-bounded CCA security [25, 11, 10]. Actually, the above works allow for a stronger notion of one parallel query of many ciphertexts. This is related to the notion of non-malleability [14, 4].

B Plaintext, Randomness and Ciphertext Spaces

For this paper, we assume that detectable schemes allow arbitrary-length messages and that the encryption randomness will always be of the length of the security parameter and therefore independent of the message length. We also assume that the ciphertext size is a deterministic function of the security parameter and the message length.

This will be useful for a property of our systems where we will implicitly encrypt our own randomness by having it nested inside of another ciphertext. Having short randomness is actually more important for the one-time CCA-secure schemes used as a building block in our construction, but we argue generally that this is not a limiting assumption for encryption schemes here.

We justify our main assumptions with two lemmas.

First, we use Lemma 2.5, which asserts that one-bit DCCA-secure encryption implies arbitrary-length DCCA-secure encryption. Recall the construction. Say $\Pi = (\text{KeyGen}, \text{Enc}, \text{Dec}, F)$ is a detectable encryption system with plaintext space $\{0, 1\}$. We can construct a new scheme $\Pi' = (\text{KeyGen}, \text{Enc}', \text{Dec}', F')$ with plaintext space $\{0, 1\}^*$ by defining Enc' as follows:

$$\text{Enc}'(pk, m) = \text{Enc}(pk, m_1), \dots, \text{Enc}(pk, m_n)$$

where $m = m_1 \dots m_n$. The decryption algorithm Dec' decrypts each ciphertext piece using Dec . The function F' performs n^2 invocations of F , testing each ciphertext piece of C with each ciphertext piece of C' , and outputting 1 if any invocation of F returned 1, and 0 otherwise. We now prove Lemma 2.5.

Proof. Recall that our construction defined the function F' to perform n^2 invocations of F , testing each ciphertext piece of C with each ciphertext piece of C' , and outputting 1 if any invocation of F returned 1, and 0 otherwise. Clearly F' is efficiently computable if F is.

Unpredictability of F' . We suppose there exists a PPT adversary $\mathcal{A}$ that causes the output of the basic unpredictability experiment with Π' to be 1 with non-negligible probability ϵ . We construct a PPT adversary $\mathcal{B}$ against the basic unpredictability experiment with Π . The experiment for $\mathcal{B}$ begins by a run of KeyGen producing pk, sk . $\mathcal{B}$ is given pk and access to a decryption oracle $\text{Dec}(sk, \cdot)$. It gives pk to $\mathcal{A}$. To simulate a decryption oracle for $\mathcal{A}$, $\mathcal{B}$ takes a ciphertext query from $\mathcal{A}$ in the form $c_1, \dots, c_n$ and separately queries each of $c_1, \dots, c_n$ to its decryption oracle. It returns the ordered n -tuple of replies to $\mathcal{A}$.

When $\mathcal{A}$ outputs a message $m = m_1 \dots m_k$ and a ciphertext $c_1, \dots, c_\ell$, $\mathcal{B}$ chooses two random indices $i \in [k]$, $j \in [\ell]$ and outputs m_i, c_j . The experiment then proceeds to the challenge phase, computing $c_i^* \leftarrow \text{Enc}(pk, m_i)$. This is distributed identically to the i^{th} piece of the challenge ciphertext that would be created in the experiment for $\mathcal{A}$. We let K and L be polynomial-size bounds such that $\mathcal{A}$ chooses $k \leq K$ and $\ell \leq L$ with all but negligible probability. Then the probability that the outcome of $\mathcal{B}$'s is 1 is negligibly close to $\frac{\epsilon}{KL}$.

To see this, we consider simulating the full experiment for $\mathcal{A}$ by also computing $c_{i'}^* \leftarrow \text{Enc}(pk, m_{i'})$ for all $i' \neq i$. We note that when $\mathcal{A}$ succeeds, there must be some indices i^*, j^* such that

$F(pk, c_{j^*}, \text{Enc}(pk, m_{i^*})) = 1$. Fixing all the randomness except for the choice of i, j by $\mathcal{B}$, $\mathcal{B}$ will now succeed in its experiment as long as it chooses $i = i^*$ and $j = j^*$: this occurs with probability $\frac{1}{KL}$, which is $\geq \frac{1}{KL}$ with all but negligible probability. We note that the choices of i, j by $\mathcal{B}$ are made independently of all other random choices occurring in this simulated experiment for $\mathcal{A}$. Hence, $\mathcal{B}$ causes the output of the basic unpredictability experiment with Π' to be 1 with non-negligible probability. We note that the same proof (with trivial adjustments) works for the strong unpredictability experiment.

Indistinguishability of Encryptions. It remains to show a PPT adversary must have a negligible advantage in the indistinguishability of encryptions experiment. We suppose there exists a PPT adversary $\mathcal{A}$ that causes the output of the indistinguishability of encryptions experiment with Π' to equal 1 with probability non-negligibly greater than $\frac{1}{2}$. We construct a PPT adversary $\mathcal{B}$ for the indistinguishability of encryptions experiment with Π . We employ a hybrid argument. We let k denote an upper bound of the length of the messages m_0, m_1 produced by $\mathcal{A}$ as its output for Phase I of the experiment. We then define experiments 1 through k as follows. Each experiment i is like the indistinguishability of encryptions experiment except for how the challenge ciphertext is created. In experiment i , the first $i - 1$ pieces of the ciphertext are created by encrypting the first $i - 1$ bits of m_0 , the i^{th} piece is created by encrypting the i^{th} bit of m_b , and the remaining pieces are encryptions of the bits of m_1 , starting from the $i + 1$ bit. The output of each experiment is still defined to be 1 when $b = b'$.

We observe that there must exist some i^* such that $\mathcal{A}$ causes the output of experiment i^* with Π' to be 1 with probability non-negligibly greater than $\frac{1}{2}$. The experiment for $\mathcal{B}$ begins by a run of KeyGen producing pk, sk . $\mathcal{B}$ is given pk and access to a decryption oracle $\text{Dec}(sk, \cdot)$. It forwards pk to $\mathcal{A}$. To simulate a decryption oracle for $\mathcal{A}$, $\mathcal{B}$ takes a ciphertext query from $\mathcal{A}$ in the form $c_1, \dots, c_n$ and separately queries each of $c_1, \dots, c_n$ to its decryption oracle. It returns the ordered n -tuple of replies to $\mathcal{A}$.

$\mathcal{A}$ produces two messages m_0, m_1 of the same length in the message space associated with pk . $\mathcal{B}$ produces the ciphertext c^* as follows. It encrypts the first $i^* - 1$ bits of m_0 to form the first $i^* - 1$ pieces of c^* , and submits the i^* th bits of m_0, m_1 as its messages to the challenger for the indistinguishability of encryptions experiment for Π . It uses the ciphertext it receives in return as the i^* piece of c^* . It forms the remaining pieces by encrypting the final bits of m_1 , (starting from the $i^* + 1$ bit). It gives c^* to $\mathcal{A}$.

$\mathcal{A}$ may continue to make decryption queries, as long as none of the pieces of these queries are “related” to pieces of c^* in the sense defined by F (this is just a restatement of the definition for F'). This restriction allows $\mathcal{B}$ to simulate the decryption oracle on these queries as before, by submitting them separately to its own decryption oracle. Finally, $\mathcal{A}$ will output a bit b' , which $\mathcal{B}$ copies as its own output. This will equal b with probability non-negligibly greater than $\frac{1}{2}$, since $\mathcal{A}$ accomplishes this in experiment i^* . $\square$

Next, we rely on the common trick of replacing the randomness for encryption with the output of a pseudorandom generator. Thus the “real” randomness needed for encryption is just a seed of the length of the security parameter. Say $\Pi = (\text{KeyGen}, \text{Enc}, \text{Dec}, F)$ is a detectable encryption system with randomness s of length $\ell = \ell(\lambda)$, where λ is the security parameter and ℓ is a polynomial. Let $G : \{0, 1\}^\lambda \rightarrow \{0, 1\}^{\ell(\lambda)}$ be a pseudorandom generator; if pseudorandom generators exist, then such a pseudorandom generator must exist [19, p. 76]. We can construct a new scheme

$\Pi' = (\text{KeyGen}', \text{Enc}', \text{Dec}, F)$ with randomness s' of length λ . The first step of KeyGen' and Enc' is to run $t := G(s')$. Then they operate exactly as KeyGen and Enc using the appropriate bits of t .

Lemma B.1 (Short Randomness is Sufficient) *Let Π and Π' be as above. If Π is DCCA-secure, then so is Π' .*

We omit the straightforward proof of the above lemma. It expands in the obvious way to CPA and CCA2-secure systems.

Finally, we assume that the ciphertext size can be considered a deterministic function of the security parameter and message length. Informally, the length of the ciphertext cannot be predictably related to the *message value*, because this would break the indistinguishability of encryptions property. It is possible that some systems might, for example, have variable-length ciphertexts by appending a variable-length random string to the end of the encryption. However, we focus our attention on schemes that do not allow this.

C Proof of Theorem 4.5 (Main Construction is Nested Indistinguishable)

Proof. Given that bad query events occur with only negligible probability, we use this fact to prove the nested indistinguishability of our main construction based on the indistinguishability property of the DCCA-security of Π_{dcca} .

The Reduction Algorithm. Let 1^λ be the security parameter. Suppose there exists a PPT adversary $\mathcal{A}$ that causes the output of the nested experiment with the main construction to output 1 with probability $\frac{1}{2} + \epsilon$. We construct a PPT adversary $\mathcal{B}$ against the indistinguishability experiment of the DCCA security of Π_{dcca} with detecting function F .

1. Setup: $\mathcal{B}$ obtains PK_{in} from the $\text{Exp}^{\text{indist}}$ challenger. It runs $\text{KeyGen}_{1b\text{-cca}}$ to obtain $(\text{PK}_A, \text{SK}_A)$ and $\text{KeyGen}_{\text{cpa}}$ to obtain $(\text{PK}_B, \text{SK}_B)$.
2. Phase 1: $\mathcal{B}$ gives to $\mathcal{A}$ the public key $\text{PK} = (\text{PK}_{\text{in}}, \text{PK}_A, \text{PK}_B)$. When $\mathcal{A}$ queries the decryption oracle on CT , $\mathcal{B}$ can simulate the normal decryption algorithm using SK_A and the phase 1 oracle $\text{Dec}(\text{SK}_{\text{in}}, \cdot)$. Eventually, $\mathcal{A}$ outputs a pair of messages m_0, m_1 .
3. Challenge: Choose random $\beta \in \{0, 1\}$ and $r_A, r_B \in \{0, 1\}^\lambda$. Send to the $\text{Exp}^{\text{indist}}$ challenger the messages $M_0 = (r_A, r_B, m_\beta)$ and $M_1 = 0^{|M_0|}$, and obtain from this challenger the ciphertext CT_{in}^* . Compute $\text{CT}_A^* := \text{Enc}_{1b\text{-cca}}(\text{PK}_A, \text{CT}_{\text{in}}^*; r_A)$ and $\text{CT}_B^* := \text{Enc}_{\text{cpa}}(\text{PK}_B, \text{CT}_{\text{in}}^*; r_B)$. Return $\text{CT}^* := (\text{CT}_A^*, \text{CT}_B^*)$ to $\mathcal{A}$.
4. Phase 2: When $\mathcal{A}$ queries the decryption oracle on $\text{CT} := (\text{CT}_A, \text{CT}_B)$, compute $\text{CT}_{\text{in}} := \text{Dec}_{1b\text{-cca}}(\text{SK}_A, \text{CT}_A)$. If
 - (a) Case 1 (a bad query event): $\text{CT}_A \neq \text{CT}_A^*$ and yet $F(\text{PK}_{\text{in}}, \text{CT}_{\text{in}}^*, \text{CT}_{\text{in}}) = 1$, then abort and take a random guess.
 - (b) Case 2 (partial match with challenge): $\text{CT}_A = \text{CT}_A^*$, then return $\perp$ to $\mathcal{A}$.

Otherwise, query the phase 2 oracle, $\text{Dec}(\text{SK}_{\text{in}}, \cdot)$, to decrypt CT_{in} , and return its response to $\mathcal{A}$.

5. Output: When $\mathcal{A}$ outputs a bit, $\mathcal{B}$ echoes the bit as its output.

Analysis. We begin our analysis by arguing that $\mathcal{B}$ correctly answers all decryption queries except when it aborts. Through Claims 4.3 and 4.4, we have already established that a bad query event, causing the abort in Case 1, happens with only negligible probability assuming that Π_{dcca} is DCCA secure, $\Pi_{\text{1b-cca}}$ is 1-bounded CCA secure, and Π_{cpa} is CPA secure.

Next, we show that a partial match with the challenge, causing the $\perp$ response in Case 2, is correct because that query must be invalid. Since a decryption query on the challenge is forbidden by the experiment, if $\text{CT}_A = \text{CT}_A^*$, then $\text{CT}_B \neq \text{CT}_B^*$. However, we argue that this must be an invalid ciphertext, i.e., one on which the main construction’s decryption algorithm would return $\perp$. We see this as follows. Since decryption is deterministic, we have $T := \text{Dec}_{\text{1b-cca}}(\text{SK}_A, \text{CT}_A) = \text{Dec}_{\text{1b-cca}}(\text{SK}_A, \text{CT}_A^*)$ and $(r_A, r_B, m) := \text{Dec}_{\text{dcca}}(\text{SK}_{\text{in}}, T)$. By the checks enforced by the main construction’s decryption algorithm, there is only one “second half” that matches $\text{CT}_A = \text{CT}_A^*$, that is $\text{Enc}_{\text{cpa}}(\text{PK}_B, T; r_B)$. Since the challenge is a valid ciphertext, CT_B^* must be this value and CT_B must cause an error.

When neither Case 1 or Case 2 applies in phase 2, the inner decryption query will succeed since the ciphertext is not detectably related to the challenge. This allows $\mathcal{B}$ to respond correctly.

Finally, $\mathcal{B}$ causes all inputs to $\mathcal{A}$ to have the same distribution as the nested experiment. When $\mathcal{B}$ aborts, it causes the $\text{Exp}^{\text{indist}}$ experiment to output 1 with probability $\frac{1}{2}$. When $\mathcal{B}$ does not abort, all decryption queries are answered correctly and this causes the $\text{Exp}^{\text{indist}}$ experiment to output 1 with probability $\frac{1}{2} + \epsilon$. Since $\mathcal{B}$ does not abort with high probability, if ϵ is non-negligible, then $\mathcal{B}$ causes the experiment’s output to be 1 with probability non-negligibly greater than $\frac{1}{2}$. $\square$

D Detectable CCA from (Adaptive) Tag-Based Encryption

MacKenzie, Reiter and Yang [22] define a tag-based encryption scheme as an encryption scheme that takes in an additional “tag” parameter on encryption and decryption. We recall this definition.

Consider the following experiment $\text{Exp}_{\mathcal{A}, \Pi}^{\text{tbe-atag-cca}}(\lambda)$ defined for a tag-based encryption scheme $\Pi = (\text{TBKeyGen}, \text{TBEnc}, \text{TBD ec})$ and an adversary $\mathcal{A}$:

1. Setup: $\text{TBKeyGen}(1^\lambda)$ is run to obtain keys (pk, sk) .
2. Phase 1: Adversary $\mathcal{A}$ is given pk and access to a decryption oracle $\text{TBD ec}(sk, \cdot, \cdot)$. $\mathcal{A}$ outputs a pair of messages m_0, m_1 of the same length in the message space associated with pk and a target tag t^* from the tag space.
3. Challenge: A random bit $b \leftarrow \{0, 1\}$ is chosen, and then a ciphertext $c^* \leftarrow \text{TBEnc}(pk, t^*, m_b)$ is computed and given to $\mathcal{A}$. We call c^* the challenge ciphertext.
4. Phase 2: $\mathcal{A}$ continues to have access to $\text{TBD ec}(sk, \cdot, \cdot)$, but may not request a decryption of a ciphertext with tag t such that $t \neq t^*$. Finally, $\mathcal{A}$ outputs a bit b' .
5. Output: The output of the experiment is defined to be 1 if $b' = b$, and 0 otherwise.

Definition D.1 (Adaptive Tag-Based Security) *A tag-based encryption scheme $\Pi = (\text{TBKeyGen}, \text{TBEnc}, \text{TBD ec})$ has indistinguishable encryptions under a tag-based chosen-ciphertext attack if for all probabilistic polynomial-time adversaries $\mathcal{A}$ there exists a negligible function negl such that:*

$$\Pr[\text{Exp}_{\mathcal{A}, \Pi}^{\text{tbe-atag-cca}}(\lambda) = 1] \leq \frac{1}{2} + \text{negl}(\lambda).$$

Kiltz [20] considered a *selective* variant of this definition where the target tag t^* used in the challenge ciphertext must be output by the adversary before the public key is created.

We now show that any tag-based scheme satisfying Definition D.1 gives rise to a DCCA-secure system. Let $\Pi = (\text{TBKeyGen}, \text{TBEnc}, \text{TBDec})$ be a tag-based encryption system with tag space T . We create a detectable encryption system, $\Pi' = (\text{KeyGen}, \text{Enc}, \text{Dec}, F)$, as:

1. $\text{KeyGen}(1^\lambda)$: Run $\text{TBKeyGen}(1^\lambda)$ and output the resulting key pair.
2. $\text{Enc}(pk, m)$: Select a random tag $t \leftarrow T$ and compute $d \leftarrow \text{TBEnc}(pk, t, m)$. Output the ciphertext $c := (t, d)$.
3. $\text{Dec}(sk, c)$: Parse c as (t, d) . Output the result of $\text{TBDec}(sk, t, d)$.
4. $F(pk, c, c')$: Parse c as (t, d) and c' as (t', d') . Output 1 if $t = t'$ and 0 otherwise.

Lemma D.2 (DCCA from Tag-Based Encryption) *If Π is an adaptively-secure tag-based system according to Definition D.1 with an exponentially-large tag space T , then Π' is a DCCA-secure detectable system according to Definition 2.2.*

Proof. This proof involves two parts. First, we argue that the detecting function F is unpredictable. In the basic unpredictability game, the adversary must fix a tag $t \in T$ as part of the ciphertext c . After this is fixed, the challenger chooses a random tag $t^* \in T$ for the challenge ciphertext. The detecting function F outputs 1, causing the experiment to output 1, if and only if $t = t^*$. This happens with probability exactly $1/|T|$. Since we conditioned that T is exponentially-large in the security parameter, we can conclude that the adversary causes the basic unpredictability experiment to output 1 will only negligible probability.

Second, we argue that the encryptions of Π' are indistinguishable under a detectable chosen ciphertext attack. Suppose this is false and there exists a PPT adversary $\mathcal{A}$ with probability $1/2$ plus a non-negligible advantage ϵ , then we use this adversary to construct a PPT adversary $\mathcal{B}$ for the tag-based experiment as follows. $\mathcal{B}$ passes the public key pk to $\mathcal{A}$. When $\mathcal{A}$ makes a decryption query on ciphertext $c := (t, d)$, $\mathcal{B}$ passes this query to its decryption oracle with tag t and ciphertext d and returns the answer. When $\mathcal{A}$ outputs a pair of messages m_0, m_1 , $\mathcal{B}$ chooses a random $t^* \in T$ and outputs (m_0, m_1, t^*) . The tag-based challenger responds with a ciphertext $c^* := (t^*, d^*)$, which $\mathcal{B}$ passes to $\mathcal{A}$. When $\mathcal{A}$ makes a decryption query it must obey the detecting predicate F which is defined to forbid any ciphertext $c := (t, d)$ such that $t = t^*$, thus $\mathcal{B}$ will be able to pass the query on to its decryption oracle and return the answer. When $\mathcal{A}$ outputs a bit, $\mathcal{B}$ outputs the same bit. It is straightforward to see that $\mathcal{B}$ is able to simulate the DCCA game for $\mathcal{A}$ exactly and will succeed in the tag-based experiment with probability $1/2 + \epsilon$, thereby breaking the security of Π . $\square$

New Definitions and Separations for Circular Security

David Cash*

Matthew Green†

Susan Hohenberger‡

Abstract

Traditional definitions of encryption security guarantee secrecy for any plaintext that can be computed by an outside adversary. In some settings, such as anonymous credential or disk encryption systems, this is not enough, because these applications encrypt messages that depend on the secret key. A natural question to ask is do standard definitions capture these scenarios? One area of interest is *n-circular security* where the ciphertexts $E(pk_1, sk_2), E(pk_2, sk_3), \dots, E(pk_{n-1}, sk_n), E(pk_n, sk_1)$ must be indistinguishable from encryptions of zero. Acar et al. (Eurocrypt 2010) provided a CPA-secure public key cryptosystem that is not 2-circular secure due to a distinguishing attack.

In this work, we consider a natural relaxation of this definition. Informally, a cryptosystem is *n-weak circular secure* if an adversary given the cycle $E(pk_1, sk_2), E(pk_2, sk_3), \dots, E(pk_{n-1}, sk_n), E(pk_n, sk_1)$ has no significant advantage in the regular security game, (e.g., CPA or CCA) where ciphertexts of chosen messages must be distinguished from ciphertexts of zero. Since this definition is sufficient for some practical applications and the Acar et al. counterexample no longer applies, the hope is that it would be easier to realize, or perhaps even implied by standard definitions. We show that this is unfortunately not the case: even this weaker notion is not implied by standard definitions. Specifically, we show:

- For symmetric encryption, under the minimal assumption that one-way functions exist, *n*-weak circular (CPA) security is not implied by CCA security, for any *n*. In fact, it is not even implied by authenticated encryption security, where ciphertext integrity is guaranteed.
- For public-key encryption, under a number-theoretic assumption, 2-weak circular security is not implied by CCA security.

In both of these results, which also apply to the stronger circular security definition, *we actually show for the first time an attack in which the adversary can recover the secret key of an otherwise-secure encryption scheme after an encrypted key cycle is published*. These negative results are an important step in answering deep questions about which attacks are prevented by commonly-used definitions and systems of encryption. They say to practitioners: if key cycles may arise in your system, then even if you use CCA-secure encryption, your system may break catastrophically; that is, a passive adversary might be able to recover your secret keys.

Keywords: Encryption, Definitions, Circular Security, Counterexamples

1 Introduction

Encryption is one of the most fundamental cryptographic primitives. Most definitions of encryption security [21, 18, 34] follow the seminal notion of Goldwasser and Micali which guarantees indistinguishability of

*IBM T.J. Watson Research Center, 1101 Kitchawan Road, Yorktown Heights, N.Y. 10598, cdc@ucsd.edu. This work was performed at University of California, San Diego, supported in part by NSF grant CCF-0915675.

†Johns Hopkins University, 3400 N. Charles St., Baltimore, MD 21218. Supported in part by the Defense Advanced Research Projects Agency (DARPA) and the Air Force Research Laboratory (AFRL) under contract FA8750-11-2-0211, the Office of Naval Research under contract N00014-11-1-0470, NSF grant CNS-1010928 and HHS 90TR0003/01. Its contents are solely the responsibility of the authors and do not necessarily represent the official views of the HHS mgreen@cs.jhu.edu

‡Johns Hopkins University. Supported in part by the Defense Advanced Research Projects Agency (DARPA) and the Air Force Research Laboratory (AFRL) under contract FA8750-11-2-0211, the Office of Naval Research under contract N00014-11-1-0470, NSF CNS 1154035, a Microsoft Faculty Fellowship and a Google Faculty Research Award. Applying to all authors, the views expressed are those of the authors and do not reflect the official policy or position of the Department of Defense or the U.S. Government.

encryptions for messages chosen by the adversary [21]. However, Goldwasser and Micali wisely warned to be careful when using a system proven secure within this framework on messages that the adversary cannot derive himself.

Over the past several years, there has been significant interest in designing schemes secure against *key-dependent message attacks*, e.g., [15, 11, 30, 3, 26, 28, 13, 14, 5, 2], where the system must remain secure even when the adversary is allowed to obtain encryptions of messages that depend on the secret keys themselves. In this work, we are particularly interested in circular security [15]. A public-key cryptosystem is *n-circular secure* if the ciphertexts $E(pk_1, sk_2), E(pk_2, sk_3), \dots, E(pk_{n-1}, sk_n), E(pk_n, sk_1)$, as well as ciphertexts of chosen messages, cannot be distinguished from encryptions of zero, for independent key pairs. Either by design or accident, these key cycles naturally arise in many applications, including storage systems such as BitLocker [13], anonymous credentials [15], the study of “axiomatic security” [30, 3] and more. See [13] for a discussion of the applications.

Until recently, few positive or negative results regarding circular security were known outside of the random oracle model. On one hand, no *n-circular* secure cryptosystems were known for $n > 1$. On the other hand, no counterexamples existed for $n > 1$ to separate the definitions of circular and CPA security; that is, as far as anyone knew the CPA-security definition already captured circular security for any cycle larger than a self-loop.

Recently, this gap has been closing in two ways. On the positive side, several circular-secure schemes have been proposed [13, 5, 14]. The focus of the current work is on negative results – namely, investigating whether standard notions of encryption are “safe” for circular applications.

In 2008, Boneh, Halevi, Hamburg and Ostrovsky proved, by counterexample, that *one-way* security does not imply circular security [13]. Recently, Acar, Beleniky, Bellare and Cash [2] proved that, under an assumption in bilinear groups, CPA-security does not imply circular security.

Our Results We narrow this gap even further by studying the extent to which standard definitions (e.g., CPA, CCA) imply a *weak* form of circular security. Our results are primarily negative.

1. Relaxing the Circular Security Notion. Perhaps the current formulation of circular security is “too strong”; that is, perhaps there is a relaxed notion of this definition which simultaneously satisfies many practical applications and yet is also *already* captured by standard security notions. This is an area worth investigating. We begin by proposing a natural relaxation called *weak circular security* where the adversary is handed an encrypted cycle $E(pk_1, sk_2), E(pk_2, sk_3), \dots, E(pk_{n-1}, sk_n), E(pk_n, sk_1)$ along with the public keys and then proceeds to play the CPA or CCA security game as normal (where these ciphertexts are also off-limits for the decryption oracle). We stress here that the encrypted cycle is *always* generated as described, and is never changed to encryptions of zero. This definition is intriguing, and perhaps of independent interest, for two reasons.

First, the Acar et al. [2] counterexample does *not* apply to it. That construction uses the bilinear map to test whether a sequence of ciphertexts contain a cycle or zeros. Here the adversary knows he’s getting an encrypted cycle, but then must extract some knowledge from this that helps him distinguish two messages of his choosing.

Second, this definition appears sufficient for some practical settings. Using a weak circular secure encryption scheme, Alice and Bob could exchange keys with each other over an insecure channel knowing that: (1) Eve can detect that they did so, but (2) Eve cannot learn anything about their other messages. Similarly, an adversary scanning over a user’s BitLocker storage may detect that her drive contains an encrypted cycle, but cannot read anything on her drive. In an anonymous credential system of Camenisch and Lysyanskaya [15], a user has multiple keys. To participate in the system, the user must encrypt them in a cycle, provide this cycle to the other users, and prove that she has done this correctly. Then, if she shares one key, she automatically shares all her keys. In their application, *detection* of a cycle is actually desirable, provided that subsequent encryptions remain secure.

2. Symmetric-Key Counterexamples. In the symmetric setting, we show that standard notions do not imply *n-circular* security for any positive n . Specifically, given any $n \geq 1$, we show how to construct a

secure authenticated encryption scheme (which is necessarily CCA-secure; see Section 2) that is not n -weak circular secure, under the minimal assumption that secure authenticated encryption schemes exist, which are equivalent to one-way functions.

The main technical ingredient in our counterexample is a lemma showing that it is provably hard for an adversary to compute an encrypted key cycle itself, assuming that the symmetric scheme under attack is a secure authenticated encryption scheme (or CCA secure). We stress that this lemma does not hold if the encryption scheme is only CPA secure.

Our lemma gives us leverage in constructing a counterexample because it means the adversary is given strictly more power in the weak circular security game than in the standard security game. Specifically, the adversary is given an encrypted key cycle in the weak circular security game that it could not have computed itself, and we design a scheme to help such an adversary without affecting regular security.

3. Public-Key Counterexamples. We show that neither CPA nor CCA-security imply (even) weak circular security for cycles of size 2. That is, we show secure systems that are totally compromised when the independently-generated ciphertexts $E(pk_A, sk_B)$ and $E(pk_B, sk_A)$ are released. This is a difficult task, because the system must remain secure if either one, but only one, of these ciphertexts are released. Moreover, this counterexample requires new ideas. We cannot use the common trick in self-loop counterexamples that test if the message is the secret key corresponding to the public key, since there is no way for the encryption algorithm with public key pk_A to distinguish, say, sk_B from any other valid message. Specifically, we show that:

If there exists an algebraic setting where the Symmetric External Diffie-Hellman ¹ (SXDH) assumption holds, then there exists a CPA-secure cryptosystem which is *not* 2-weak circular secure. The proposed scheme is particularly interesting in that it breaks *catastrophically* in the presence of a 2-cycle — revealing the secret keys of both users.

Moreover, if simulation-sound non-interactive zero-knowledge (NIZK) proof systems exist for NP and there exists an algebraic setting where the Symmetric External Diffie-Hellman (SXDH) assumption holds, then there exists a CCA-secure cryptosystem which is *not* 2-weak circular secure. This is also the first separation of CCA security and (regular) circular security.

These results deepen our understanding of how to define “secure” encryption and which practical attacks are captured by the standard definitions. They also provide additional justification for the ongoing effort, e.g. [13, 14, 5], to develop cryptosystems which are provably circular secure.

1.1 Related Work

In 2001, Camenisch and Lysyanskaya [15] introduced the notion of *circular security* and used it in their anonymous credential system to discourage users from delegating their secret keys. They also showed how to construct a circular-secure cryptosystem from any CPA-secure cryptosystem in the random oracle model. Independently, Abadi and Rogaway [1] and Black, Rogaway, Shrimpton [11] introduced the more general notion of *key-dependent message* (KDM) security, where the encrypted messages might depend on an arbitrary function of the secret keys. Black et al. showed how to realize this notion in the random oracle model.

Halevi and Krawczyk [26] extended the work of Black et al. to look at KDM security for deterministic secret-key functions such as pseudorandom functions (PRFs), tweakable blockciphers, and more. They give both positive and negative results, including some KDM-secure constructions in the standard model for PRFs. In the symmetric setting, Hofheinz and Unruh [28] showed how to construct circular-secure cryptosystems in the standard model under relaxed notions of security. Backes, Pfizmann and Scedrov [7] presented stronger notions of KDM security (some in the random oracle model) and discussed the relationships among these notions.

¹The SXDH assumption states that there is a bilinear setting $e : \mathbb{G}_1 \times \mathbb{G}_2 \rightarrow \mathbb{G}_T$ where the Decisional Diffie-Hellman (DDH) assumption holds in both $\mathbb{G}_1$ and $\mathbb{G}_2$. It has been extensively studied and used e.g., [20, 38, 31, 12, 8, 6, 23, 9, 24], perhaps most notably as a setting of the Groth-Sahai NIZK proof system [24].

In the public-key setting, Boneh, Halevi, Hamburg and Ostrovsky [13] presented the first cryptosystem which is simultaneously CPA-secure and n -circular-secure (for any n) in the standard model, based on either the DDH or Decision Linear assumptions. As mentioned earlier, Boneh et al. [13] also proved, by counterexample, that *one-way* security does not imply circular security. One-way encryption is a very weak notion, which informally states that given $(pk, E(pk, m))$, the adversary should not be able to recover m . Given any one-way encryption system, they constructed a one-way encryption system that is not n -circular secure (for any n). Their system generates two key pairs from the original and sets $PK = pk_1$ and $SK = (sk_1, sk_2)$. A message (m_1, m_2) is encrypted as $(m_1, E(pk_1, m_2))$. In the event of a 2-cycle, the values $\text{Enc}(pk_A, sk_B) = (sk_{B,1}, E(pk_{A,1}, sk_{B,2}))$ and $\text{Enc}(pk_B, sk_A) = (sk_{A,1}, E(pk_{B,1}, sk_{A,2}))$ provide the critical secret key information $(sk_{B,1}, sk_{A,1})$ in the clear.

Subsequently, Applebaum, Cash, Peikert and Sahai [5] adapted the circular-secure construction of [13] into the lattice setting. Camenisch, Chandran and Shoup [14] extended [13] to the first cryptosystem which is simultaneously CCA-secure and n -circular-secure (for any n) in the standard model, by applying the “double encryption” paradigm of Naor and Yung [33]. (Interestingly, we use this same approach in Section 4.4 to extend our public-key counterexample from CPA to CCA security.)

Haitner and Holenstein [25] recently provided strong impossibility results for KDM-security *with respect to 1-key cycles* (a.k.a., self-loops.) They study the problem of building an encryption scheme where it is secure to release $E(k, g(k))$ for various functions g . First, they show that there exists no fully-black-box reduction from a KDM-secure encryption scheme to one-way permutations (or even some families of trapdoor permutations) if the adversary can obtain encryptions of $g(k)$, where g is a $\text{poly}(n)$ -wise independent hash function. Second, there exists no reduction from an encryption scheme secure against key-dependent messages to, essentially, any cryptographic assumption, if the adversary can obtain an encryption of $g(k)$ for an *arbitrary* g , as long as the security reduction treats both the adversary and the function g as black boxes. These results address the possibility of achieving strong single-user KDM-security via reductions to cryptographic assumptions. The results in this paper study a version of KDM security that is in one sense weaker – we only allow a narrow class of functions g – but also stronger because it considers multiple users. Our results also address a different question regarding KDM security. We study whether or not KDM security is always implied by regular security while Haitner and Holenstein study the possibility of achieving strong single-user KDM security via specialized constructions.

Most closely related to our work, Acar et al. [2] demonstrated both public and private key encryption systems that are provably CPA-secure and yet also demonstrably *not* 2-circular secure. Their counterexample does not apply to CCA or weak circular security.

Subsequent to the original posting of this work, Rothblum [36] studied the circular security of bit encryption. In particular, using n -linear maps, for large n , where DDH is assumed hard in every pre-image group, he constructs a CPA (or CCA) secure bit-encryption scheme that is not circular secure; that is, where it is not “safe” to encrypt the secret key sk bit-by-bit using the corresponding public key pk . This approach is conceptually similar to extending either the Acar et al. [2] or our 2-circular counterexample in Section 4 to an n -circular counterexample using n -linear maps. Unfortunately, there are no candidate implementations for n -linear maps where $n > 2$ and even the discrete logarithm problem is believed to be hard in one of the pre-image groups. Thus, it remains an open problem to resolve these two fascinating questions relating to circular security.

There is also a relationship to recent work on *leakage resilient* and *auxiliary input* models of encryption, which mostly falls into the “self-loop” category. In leakage resilient models, such as those of Akavia, Goldwasser and Vaikuntanathan [4] and Naor and Segev [32], the adversary is given some function h of the secret key, not necessarily an encryption, such that it is *information theoretically* impossible to recover sk . The auxiliary input model, introduced by Dodis, Kalai and Lovett [17], relaxes this requirement so that it only needs to be difficult to recover sk .

Self-Loops In sharp contrast to all $n \geq 2$, the case of 1-circular security is fairly well understood. A folklore counterexample shows that CPA-security does not directly imply 1-circular security. Given any encryption scheme (G, E, D) , one can build a second scheme (G, E', D') as follows: (1) $E'(pk, m)$ outputs

$\text{IND-CPA}(\Pi, \mathcal{A}, \lambda)$	$\text{AE}(\Pi, \mathcal{A}, \lambda)$
$b \xleftarrow{r} \{0, 1\}$	$b \xleftarrow{r} \{0, 1\}$
$(pk, sk) \leftarrow \text{KeyGen}(1^\lambda)$	$K \leftarrow \text{KeyGen}(1^\lambda)$
$(m_0, m_1, z) \leftarrow \mathcal{A}_1(pk)$	$\hat{b} \leftarrow \mathcal{A}_{K,b}^{\mathcal{E}_{K,b}(\cdot, \cdot), \mathcal{D}_{K,b}^{\text{ae}}(\cdot)}(1^\lambda)$
$y \leftarrow \text{Enc}(pk, m_b)$	Output $(\hat{b} \stackrel{?}{=} b)$.
$\hat{b} \leftarrow \mathcal{A}_2(y, z)$	
Output $(\hat{b} \stackrel{?}{=} b)$	

Figure 1: Experiments for Definitions 2.1 and 2.3.

$E(pk, m) \parallel 0$ if $m \neq sk$ and $m \parallel 1$ otherwise, (2) $D'(sk, c \parallel b)$ outputs $D(sk, m)$ if $b = 0$ and sk otherwise. It is easy to show that if (G, E, D) is CPA-secure, then (G, E', D') is CPA-secure. When $E'(pk, sk) = sk \parallel 1$ is exposed, then there is a complete break. Conversely, given any CPA-secure system, one can build a 1-circular secure scheme in the standard model [13].

2 Definitions of Security

A *public-key encryption system* Π is a tuple of algorithms $(\text{KeyGen}, \text{Enc}, \text{Dec})$, where KeyGen is a key-generation algorithm that takes as input a security parameter λ and outputs a public/secret key pair (pk, sk) ; $\text{Enc}(pk, m)$ encrypts a message m under public key pk ; and $\text{Dec}(sk, c)$ decrypts ciphertext c with secret key sk . A *symmetric-key encryption system* is a public-key encryption system, except that it always outputs $pk = \perp$, and the encryption algorithm computes ciphertexts using sk , i.e. by running $\text{Enc}(sk, m)$. In the symmetric case we will sometimes write K instead of sk . As in most other works, we assume that all algorithms implicitly have access to shared public parameters establishing a common algebraic setting.

Our definitions of security will associate a message space, denoted M , with each encryption scheme. Throughout this paper, we assume that the space of possible secret keys output by KeyGen is a subset of the message space M and thus any secret key can be encrypted using any public key. For symmetric encryption schemes we will always have $M \subset \{0, 1\}^*$.

By $\nu(k)$ we denote some *negligible* function, i.e., one such that, for all $c > 0$ and all sufficiently large k , $\nu(k) < 1/k^c$. We abbreviate probabilistic polynomial time as PPT.

2.1 Standard Security Definitions

Public-key encryption We recall the standard notion of indistinguishability of encryptions under a chosen-plaintext attack due to Goldwasser and Micali [21].

Definition 2.1 (IND-CPA) Let $\Pi = (\text{KeyGen}, \text{Enc}, \text{Dec})$ be a public-key encryption scheme for the message space M . For $b \in \{0, 1\}$, $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ and $\lambda \in \mathbb{N}$, let the random variable $\text{IND-CPA}(\Pi, \mathcal{A}, \lambda)$ be defined by the probabilistic algorithm described on the left side of Figure 1. We denote the IND-CPA advantage of $\mathcal{A}$ by $\text{Adv}_{\Pi, \mathcal{A}}^{\text{cpa}}(\lambda) = 2 \cdot \Pr[\text{IND-CPA}(\Pi, \mathcal{A}, \lambda) = 1] - 1$. We say that Π is IND-CPA secure if $\text{Adv}_{\Pi, \mathcal{A}}^{\text{cpa}}(\lambda)$ is negligible for all PPT $\mathcal{A}$.

We also consider the indistinguishability of encryptions under chosen-ciphertext attacks [33, 34, 18].

Definition 2.2 (IND-CCA) Let $\Pi = (\text{KeyGen}, \text{Enc}, \text{Dec})$ be a public-key encryption scheme for the message space M . Let the random variable $\text{IND-CCA}(\Pi, \mathcal{A}, \lambda)$ be defined by an algorithm identical to $\text{IND-CPA}(\Pi, \mathcal{A}, \lambda)$ above, except that both $\mathcal{A}_1$ and $\mathcal{A}_2$ have access to an oracle $\text{Dec}(sk, \cdot)$ that returns the output of the decryption algorithm and $\mathcal{A}_2$ cannot query this oracle on input y . We denote the IND-CCA advantage of $\mathcal{A}$ by $\text{Adv}_{\Pi, \mathcal{A}}^{\text{cca}}(\lambda) = 2 \cdot \Pr[\text{IND-CCA}(\Pi, \mathcal{A}, \lambda) = 1] - 1$. We say that Π is IND-CCA secure if $\text{Adv}_{\Pi, \mathcal{A}}^{\text{cca}}(\lambda)$ is negligible for all PPT $\mathcal{A}$.

$\text{IND-CIRC-CPA}^n(\Pi, \mathcal{A}, \lambda)$ $b \xleftarrow{r} \{0, 1\}$ For $i = 1$ to n : $(pk_i, sk_i) \leftarrow \text{KeyGen}(1^\lambda)$ If $b = 1$ then $\mathbf{y} \leftarrow \text{EncCycle}(\mathbf{pk}, \mathbf{sk})$ Else $\mathbf{y} \leftarrow \text{EncZero}(\mathbf{pk}, \mathbf{sk})$ $\hat{b} \leftarrow \mathcal{A}(\mathbf{pk}, \mathbf{y})$ Output $(\hat{b} \stackrel{?}{=} b)$	$\text{IND-WCIRC-CPA}^n(\Pi, \mathcal{A}, \lambda)$ $b \xleftarrow{r} \{0, 1\}$ For $i = 1$ to n : $(pk_i, sk_i) \leftarrow \text{KeyGen}(1^\lambda)$ $\mathbf{y} \leftarrow \text{EncCycle}(\mathbf{pk}, \mathbf{sk})$ $(j, m_0, m_1, z) \leftarrow \mathcal{A}_1(\mathbf{pk}, \mathbf{y})$ $y \leftarrow \text{Enc}(pk_j, m_b)$ $\hat{b} \leftarrow \mathcal{A}_2(y, z)$ Output $(\hat{b} \stackrel{?}{=} b)$	$\text{EncCycle}(\mathbf{pk}, \mathbf{sk})$ For $i = 1$ to n $y_i \leftarrow \text{Enc}(pk_i, sk_{(i \bmod n)+1})$ Output $\mathbf{y}$ $\text{EncZero}(\mathbf{pk}, \mathbf{sk})$ For $i = 1$ to n $y_i \leftarrow \text{Enc}(pk_i, 0^{ sk_{(i \bmod n)+1} })$ Output $\mathbf{y}$

Figure 2: Experiments for Definitions 2.4 and 2.5. Each is defined with respect to a message space M , and we assume that $m_0, m_1 \in M$ always. We write $\mathbf{pk}$, $\mathbf{sk}$, and $\mathbf{y}$ for $(pk_1, \dots, pk_n)$, $(sk_1, \dots, sk_n)$ and $(y_1, \dots, y_n)$ respectively.

Symmetric-key authenticated encryption We recall the definition of secure authenticated (symmetric-key) encryption due to [35], except that we will not require pseudorandom ciphertexts. Bellare and Namprempre [10] showed that AE implies IND-CCA, and is in fact strictly stronger. For our counterexample, we target this very strong definition of security in order strengthen our results by showing that even this does not imply weak circular security.

Definition 2.3 (AE) Let $\Pi = (\text{KeyGen}, \text{Enc}, \text{Dec})$ be a symmetric-key encryption scheme for the message space M . Let the random variable $\text{AE}(\Pi, \mathcal{A}, \lambda)$ be defined by the probabilistic algorithm described on the right side of Figure 1. In the experiment, the oracle $\mathcal{E}_{K,b}^{\text{ae}}(\cdot, \cdot)$ takes as input a pair of equal-length messages (m_0, m_1) and computes $\text{Enc}(K, m_b)$. The oracle $\mathcal{D}_{K,b}^{\text{ae}}(\cdot)$ takes as input a ciphertext c and computes $\text{Dec}(K, c)$ if $b = 1$ and always returns $\perp$ if $b = 0$. The adversary is not allowed to submit any ciphertext to $\mathcal{D}_{K,b}^{\text{ae}}(\cdot)$ that was previously returned by $\mathcal{E}_{K,b}^{\text{ae}}(\cdot, \cdot)$. We denote the AE advantage of $\mathcal{A}$ by $\text{Adv}_{\Pi, \mathcal{A}}^{\text{ae}}(\lambda) = 2 \cdot \Pr[\text{AE}(\Pi, \mathcal{A}, \lambda) = 1] - 1$. We say that Π is AE secure if $\text{Adv}_{\Pi, \mathcal{A}}^{\text{ae}}(\lambda)$ is negligible for all PPT $\mathcal{A}$.

2.2 Circular Security Definitions

We next give definitions for circular security of public-key and symmetric-key encryption. These definitions are variants of the Key-Dependent Message (KDM) security notion of Black et al. [11]. By restricting the adversary's power, we make it significantly harder for us to devise a counterexample and thus prove a stronger negative result.²

Definition 2.4 (IND-CIRC-CPAⁿ) Let $\Pi = (\text{KeyGen}, \text{Enc}, \text{Dec})$ be a public-key encryption scheme for the message space M . For $b \in \{0, 1\}$, integer $n > 0$, adversary $\mathcal{A}$ and $\lambda \in \mathbb{N}$, let the random variable $\text{IND-CIRC-CPA}^n(\Pi, \mathcal{A}, \lambda)$ be defined by the probabilistic algorithm on the left side of Figure 2. We denote the IND-CIRC-CPA^n advantage of $\mathcal{A}$ by

$$\text{Adv}_{\Pi, \mathcal{A}}^{n\text{-circ-cpa}}(\lambda) = 2 \cdot \Pr[\text{IND-CIRC-CPA}^n(\Pi, \mathcal{A}, \lambda) = 1] - 1.$$

We say that Π is IND-CIRC-CPA^n secure if $\text{Adv}_{\Pi, \mathcal{A}}^{n\text{-circ-cpa}}(\lambda)$ is negligible for all PPT $\mathcal{A}$.

One could augment this definition by modifying the IND-CIRC-CPA^n experiment to allow for a challenge “left-or-right” query as in IND-CPA. While this is a quite natural modification, it only strengthens the definition, and we are interested in studying the weakest notions for which we can give a separation. Next we give a definition of *weak* circular security of public-key encryption.

²If we allowed the adversary to obtain encryptions of any affine function of the secret keys, as is done in [26, 13], then we could devise a trivial counterexample where the adversary uses 1-cycles to break the system.

Definition 2.5 (IND-WCIRC-CPAⁿ) Let $\Pi = (\text{KeyGen}, \text{Enc}, \text{Dec})$ be a public-key encryption scheme for the message space M . For $b \in \{0, 1\}$, integer $n > 0$, adversary $\mathcal{A}$ and $\lambda \in \mathbb{N}$, let the random variable $\text{IND-WCIRC-CPA}^n(\Pi, \mathcal{A}, \lambda)$ be defined by probabilistic algorithm on the center of Figure 2. We denote the IND-WCIRC-CPA^n advantage of $\mathcal{A}$ by

$$\text{Adv}_{\Pi, \mathcal{A}}^{n\text{-wcirc-cpa}}(\lambda) = 2 \cdot \Pr[\text{IND-WCIRC-CPA}^n(\Pi, \mathcal{A}, \lambda) = 1] - 1.$$

We say that Π is IND-WCIRC-CPA^n secure if the function $\text{Adv}_{\Pi, \mathcal{A}}^{n\text{-wcirc-cpa}}(\lambda)$ is negligible for all PPT $\mathcal{A}$.

Finally, we give a definition of weak circular security for symmetric encryption. We will abuse notation and also call this IND-WCIRC-CPA^n security, since it will be clear from the context whether or not we mean public-key and symmetric-key.

Definition 2.6 (IND-WCIRC-CPAⁿ) Let $\Pi = (\text{KeyGen}, \text{Enc}, \text{Dec})$ be a symmetric-key encryption scheme for the message space M . For $b \in \{0, 1\}$, integer $n > 0$, adversary $\mathcal{A}$ and $\lambda \in \mathbb{N}$, let $\text{IND-WCIRC-CPA}^n(\Pi, \mathcal{A}, \lambda)$ be defined by the following probabilistic algorithm:

$\begin{array}{l} \text{IND-WCIRC-CPA}_b^n(\Pi, \mathcal{A}, \lambda) \\ \hline b \xleftarrow{\$} \{0, 1\} \\ \text{For } i = 1 \text{ to } n: \\ \quad K_i \leftarrow \text{KeyGen}(1^\lambda) \\ \mathbf{y} \leftarrow \text{EncCycle}(\mathbf{K}) \\ \hat{b} \leftarrow \mathcal{A}^{\widetilde{\text{Enc}(\cdot, \cdot, \cdot)}}(\mathbf{y}) \\ \text{Output } (\hat{b} \stackrel{?}{=} b) \end{array}$	$\begin{array}{l} \text{EncCycle}(\mathbf{K}) \\ \hline \text{For } i = 1 \text{ to } n \\ \quad y_i \leftarrow \text{Enc}(K_i, K_{(i \bmod n)+1}) \\ \text{Output } \mathbf{y} \\ \hline \widetilde{\text{Enc}}(j, m_0, m_1) \\ \hline \text{Return } \text{Enc}(K_j, m_b) \end{array}$

We denote the IND-WCIRC-CPA^n advantage of $\mathcal{A}$ by

$$\text{Adv}_{\Pi, \mathcal{A}}^{n\text{-wcirc-cpa}}(\lambda) = 2 \cdot \Pr[\text{IND-WCIRC-CPA}^n(\Pi, \mathcal{A}, \lambda) = 1] - 1.$$

We say that Π is IND-WCIRC-CPA^n secure if $\text{Adv}_{\Pi, \mathcal{A}}^{n\text{-wcirc-cpa}}(\lambda)$ is negligible for all PPT $\mathcal{A}$.

Discussion In both the IND-CPA and IND-CIRC-CPA notions, the adversary must distinguish an encryption (or encryptions) of a special message from the encryption of zero. This choice of the message zero is arbitrary. We keep it in the statement of our definition to be consistent with [13]; however, it is important to note, for systems such as ours where zero is not in the message space, that zero can be replaced by any constant message for an equivalent definition. Acar et al. [2] use an equivalent definition where zero is replaced by a fresh random message.

We will not need to define a notion of security to withstand *circular and chosen-ciphertext attacks*, because we are able to show a stronger negative result. In Section 4.4, we provide an IND-CCA -secure cryptosystem, which is provably not IND-CIRC-CPA -secure. In other words, we are able to devise a peculiar cryptosystem: one that withstands all chosen-ciphertext attacks, and yet breaks under a weak circular attack which does not require a decryption oracle.

3 Counterexample for Symmetric Encryption

Encryption Scheme Π_{ae} Let $\Pi'_{\text{ae}} = (\text{KeyGen}', \text{Enc}', \text{Dec}')$ be a secure authenticated encryption scheme. To simplify our results, we assume that $\text{KeyGen}'(1^\lambda)$ outputs a uniformly random key K in $\{0, 1\}^\lambda$, that the message space $M' = \{0, 1\}^*$, and that ciphertexts output by $\text{Enc}'(K, m)$ are always in $\{0, 1\}^{p(|m|)}$, where p is some polynomial that depends on λ . We also assume that the first λ bits of a ciphertext are *never* equal to K . All of these assumptions can be removed via straightforward and standard modifications to our arguments below.

Fix a positive integer n . We now construct our counterexample scheme, denoted $\Pi_{\text{ae}} = (\text{KeyGen}, \text{Enc}, \text{Dec})$. We will take $\text{KeyGen} = \text{KeyGen}'$, i.e., Π_{ae} also uses keys randomly chosen from $\{0, 1\}^\lambda$. The message-space of Π_{ae} will consist of $M = \{0, 1\}^\lambda \cup \{0, 1\}^{np(\lambda)}$, bit strings of length either λ or $np(\lambda)$. The algorithms Enc and Dec are defined as follows.

$\frac{\text{Enc}(K, m)}{\begin{array}{l} \text{If } \text{IsCycle}(K, m) \text{ then} \\ \quad \text{Output } K \parallel m \\ \text{Else} \\ \quad \text{Output } \text{Enc}'(K, m) \end{array}}$	$\frac{\text{IsCycle}(K, m)}{\begin{array}{l} \text{If } m \neq np(\lambda) \\ \quad \text{Return false} \\ \text{Parse } m \text{ as } (c_1, \dots, c_n) \\ K_2 \leftarrow \text{Dec}'(K, c_1) \\ \text{For } i = 2 \text{ to } n \\ \quad K_{i \bmod n+1} \leftarrow \text{Dec}'(K_i, c_i) \\ \text{Return } (K_1 \stackrel{?}{=} K) \end{array}}$
$\frac{\text{Dec}(K, c)}{\begin{array}{l} \text{If } c = K \parallel \tilde{m} \text{ then} \\ \quad \text{Output } \tilde{m} \\ \text{Else} \\ \quad \text{Output } \text{Dec}'(K, c) \end{array}}$	

Decryption is always correct. This follows from our assumption that Enc' will never output a ciphertext that contains K as a prefix. We first establish the AE security of our scheme.

Theorem 3.1 *Encryption scheme Π_{ae} is AE secure whenever Π'_{ae} is AE secure. (Proof in Appendix A.2.)*

The proof proceeds by showing that computing an encrypted key-cycle during the AE game is equivalent to recovering the secret key. From there we can reduce the security of Π_{ae} to Π'_{ae} easily.

Curiously, Theorem 3.1 is no longer true if one replaces AE security with a symmetric version of IND-CPA security for both Π_{ae} and Π'_{ae} . Namely, some type of chosen-ciphertext security is required on Π'_{ae} to prove even chosen-plaintext security of Π_{ae} . Intuitively, this is because it might be possible for an adversary to compute an encrypted key-cycle on its own if the scheme is only IND-CPA-secure, but *not* if the scheme is AE-secure. In fact, the work of Boneh et al. [13] gives an explicit example of a scheme where the adversary can compute a cycle himself.

The Attack We now show that Π_{ae} is not circular-secure for n cycles, even in a weak sense.

Theorem 3.2 Π_{ae} is not IND-WCIRC-CPAⁿ secure.

Proof. We give an explicit adversary $\mathcal{A}$ that has advantage negligibly close to 1. The adversary takes as input the encrypted key-cycle $\mathbf{y}$ in the IND-WCIRC-CPAⁿ game. It queries $\widetilde{\text{Enc}}(1, m_0, m_1)$, where $m_0 = \mathbf{y}$ and m_1 is a random message of the same length. Let y be the ciphertext returned by the oracle.

At this point, there are many ways to proceed; perhaps the simplest is to observe that the *length* of y depends on the challenge bit b . This is because, if $b = 0$, then $m_0 = \mathbf{y}$ was encrypted, resulting in $y = K \parallel \mathbf{y}$, which is $\lambda + np(\lambda)$ bits long. If $b = 1$ then y was computed by running $\text{Enc}'(K, m_1)$, which will be $p(|m_1|) = p(np(\lambda))$ bits long *if* $\text{IsCycle}(K, m_1)$ returns false. Thus, as long as $\text{IsCycle}(K, m_1)$ returns false, $\mathcal{A}_2$ can compute the value of b by measuring y 's length.

But why should $\text{IsCycle}(K, m_1)$ return false? This follows from the AE security of Π'_{ae} . Let us parse m_1 into $(c_1, \dots, c_n)$, where each $c_i \in \{0, 1\}^{p(\lambda)}$ is random. When $\text{IsCycle}(K, m_1)$ returns true, it must be that $\text{Dec}'(K, c_1)$ did not return $\perp$. But if this happens, then we can construct an adversary to break the AE security of Π'_{ae} . The adversary simply queries $\mathcal{D}_{K,b}^{\text{ae}}(\cdot)$ at a random point, observes if it returns $\perp$ or not, and outputs $\hat{b} = 0$ or 1 depending on this observation. $\square$

We note that we could design an encryption scheme that does not have this type of ciphertext-length behavior by giving a different attack that abuses the fact that K is present in the ciphertext in one case, but not the other. We have chosen to present the attack this way for simplicity only.

4 Counterexamples for Public-Key Encryption

4.1 Preliminaries and Algebraic Setting

Bilinear Groups We work in a bilinear setting where there exists an efficient mapping function $e : \mathbb{G}_1 \times \mathbb{G}_2 \rightarrow \mathbb{G}_T$ involving groups of the same prime order p . Two algebraic properties required are that: (1) if g generates $\mathbb{G}_1$ and h generates $\mathbb{G}_2$, then $e(g, h) \neq 1$ and (2) for all $a, b \in \mathbb{Z}_p$, it holds that $e(g^a, h^b) = e(g, h)^{ab}$.

Decisional Diffie-Hellman Assumption (DDH) Let $\mathbb{G}$ be a group of prime order $p \in \Theta(2^\lambda)$. For all PPT adversaries $\mathcal{A}$, the following probability is $1/2$ plus an amount negligible in λ :

$$\Pr \left[\begin{array}{l} g, z_0 \leftarrow \mathbb{G}; a, b \leftarrow \mathbb{Z}_p; z_1 \leftarrow g^{ab}; d \leftarrow \{0, 1\}; \\ d' \leftarrow \mathcal{A}(g, g^a, g^b, z_d) : d = d' \end{array} \right].$$

Strong External Diffie-Hellman Assumption (SXDH): Let $e : \mathbb{G}_1 \times \mathbb{G}_2 \rightarrow \mathbb{G}_T$ be bilinear groups. The SXDH assumption states that the DDH problem is hard in both $\mathbb{G}_1$ and in $\mathbb{G}_2$. This implies that there does *not* exist an efficiently computable isomorphism between these two groups. The SXDH assumption appears in many prior works, such as [20, 38, 31, 12, 8, 6, 23, 9, 24, 2].

Indistinguishability and Pseudorandom Generators

Definition 4.1 (Indistinguishability) Two ensembles of probability distributions $\{X_k\}_{k \in \mathbb{N}}$ and $\{Y_k\}_{k \in \mathbb{N}}$ with index set $\mathbb{N}$ are said to be computationally indistinguishable if for every polynomial-size circuit family $\{D_k\}_{k \in \mathbb{N}}$, there exists a negligible function ν such that

$$|\Pr[x \leftarrow X_k : D_k(x) = 1] - \Pr[y \leftarrow Y_k : D_k(y) = 1]|$$

is less than $\nu(k)$. We denote such sets $\{X_k\}_{k \in \mathbb{N}} \stackrel{c}{\approx} \{Y_k\}_{k \in \mathbb{N}}$.

Definition 4.2 (Pseudorandom Generator [29]) Let U_x denote the uniform distribution over $\{0, 1\}^x$. Let $\ell(\cdot)$ be a polynomial and let G be a deterministic polynomial-time algorithm such that for any input $s \in \{0, 1\}^n$, algorithm G outputs a string of length $\ell(n)$. We say that G is a pseudorandom generator if the following two conditions hold:

- (Expansion:) For every n , it holds that $\ell(n) > n$.
- (Pseudorandomness:) For every n , $\{U_{\ell(n)}\}_n \stackrel{c}{\approx} \{s \leftarrow U_n : G(s)\}_n$.

The constructions of Section 4.2 use a PRG where the domain of the function is an exponentially-sized cyclic group.

4.2 Encryption Scheme Π_{cpa}

We now describe an encryption scheme $\Pi_{\text{cpa}} = (\text{KeyGen}, \text{Enc}, \text{Dec})$. It is set in asymmetric bilinear groups $e : \mathbb{G}_1 \times \mathbb{G}_2 \rightarrow \mathbb{G}_T$ of prime order p where we assume that the groups $\mathbb{G}_1$ and $\mathbb{G}_2$ are distinct and that the DDH assumption holds in both. We assume that a single set of group parameters $(e, p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, g, h)$, where $\mathbb{G}_1 = \langle g \rangle, \mathbb{G}_2 = \langle h \rangle$, will be shared across all keys generated at a given security level and are implicitly provided to all algorithms.

The message space is $\mathcal{M} = \{0, 1\} \times \mathbb{Z}_p^* \times \mathbb{Z}_p^*$. Let $\text{encode} : \mathcal{M} \rightarrow \{0, 1\}^{\ell(\lambda)}$ and $\text{decode} : \{0, 1\}^{\ell(\lambda)} \rightarrow \mathcal{M}$ denote an invertible encoding scheme where $\ell(\lambda)$ is the polynomial length of the encoded message. Let $F : \mathbb{G}_T \rightarrow \{0, 1\}^{\ell(\lambda)}$ be a pseudorandom generator secure under the Decisional Diffie Hellman assumption. (Recall that pseudorandom generators can be constructed from any one-way function [27].)

KeyGen(1^λ). The key generation algorithm selects a random bit $\beta \leftarrow \{0, 1\}$ and random values $a_1, a_2 \leftarrow \mathbb{Z}_p^*$. The secret key is set as $sk = (\beta, a_1, a_2)$. We note that $sk \in \mathcal{M}$. The public key is set as:

$$pk = \begin{cases} (0, e(g, h)^{a_1}, g^{a_2}) \in \{0, 1\} \times \mathbb{G}_T \times \mathbb{G}_1 & \text{if } \beta = 0 \\ (1, e(g, h)^{a_1}, h^{a_2}) \in \{0, 1\} \times \mathbb{G}_T \times \mathbb{G}_2 & \text{if } \beta = 1. \end{cases}$$

Encrypt(pk, M). The encryption algorithm parses the public key $pk = (\beta, Y_1, Y_2)$, where Y_2 may be in $\mathbb{G}_1$ or $\mathbb{G}_2$ depending on the structure of the public key, and message $M = (\alpha, m_1, m_2) \in \mathcal{M}$. Note that m_1 and m_2 cannot be zero, but these values can be easily included in the message space by a proper encoding.

Select random $r \leftarrow \mathbb{Z}_p$ and $R \leftarrow \mathbb{G}_T$. Set $I = F(R) \oplus \text{encode}(M)$.

Output the ciphertext C as:

$$C = \begin{cases} (g^r, R \cdot Y_1^r, Y_2^{rm_2} \cdot g^{m_1}, I) & \text{if } \beta = 0; \\ (h^r, R \cdot Y_1^r, Y_2^{rm_2}, I) & \text{if } \beta = 1. \end{cases}$$

We note that in the first case, $C \in \mathbb{G}_1 \times \mathbb{G}_T \times \mathbb{G}_1 \times \{0, 1\}^{\ell(\lambda)}$, while in the second $C \in \mathbb{G}_2 \times \mathbb{G}_T \times \mathbb{G}_2 \times \{0, 1\}^{\ell(\lambda)}$.

Decrypt(sk, C). The decryption algorithm parses the secret key $sk = (\beta, a_1, a_2)$ and the ciphertext $C = (C_1, C_2, C_3, C_4)$. Next, it computes:

$$R = \begin{cases} (C_2/e(C_1, h))^{a_1} & \text{if } \beta = 0; \\ (C_2/e(g, C_1))^{a_1} & \text{if } \beta = 1. \end{cases}$$

Then it computes $M' = F(R) \oplus C_4 \in \{0, 1\}^{\ell(\lambda)}$ and outputs the message $M = \text{decode}(M')$.

Discussion Like the circular-secure scheme of Boneh et al. [13], the above cryptosystem is a variation on El Gamal [19]. It is a practical system, which on first glance might be somewhat reminiscent of schemes the readers are used to seeing in the literature. The scheme includes a few “artificial” properties: (1) placing a public key in either $\mathbb{G}_1$ or $\mathbb{G}_2$ at random and (2) the fact that the ciphertext value C_3 is unused in the decryption algorithm. We will shortly see that these features are “harmless” in a semantic-security sense, but very useful for recovering the secret keys of the system in the presence of a two cycle. While it is not unusual for counterexamples to have artificial properties (e.g., [16, 22]), we can address these points as well.³ In Appendix C, we show that property (1) can be removed by doubling the length of the ciphertext. For property (2), we observe that many complex protocols such as group signatures (e.g., [12]) combine ciphertexts with other components that are unused in decryption but are quite important to the protocol as a whole. Thus, we believe our counterexample is not that far fetched. It is possible that such an attack could exist on one of today’s commonly-used encryption algorithms.

We first show that Π_{cpa} meets the standard notion of CPA security.

Theorem 4.3 *Encryption scheme Π_{cpa} is IND-CPA secure under the Decisional Diffie-Hellman Assumption in $\mathbb{G}_1$ and $\mathbb{G}_2$ (SXDH).*

The proof is given in Appendix B. It is relatively standard and involves repeated applications of the DDH assumption and PRG security.

³While our scheme is different from that of Acar et al. [2], that scheme also has similar artificial properties such as the presence of values that are not used in decryption.

4.3 The Attack

Despite being IND-CPA-secure, cryptosystem Π_{cpa} is not even weakly circular secure for 2-cycles. Specifically, given a circular encryption of two keys, we show that an adversary can distinguish another ciphertext with advantage $1/2$. Our adversary actually does much more than this: with probability $1/2$ over the coins used in key generation, *it can recover both secret keys*.

This is the first circular attack that allows the adversary to recover the secret keys. (In Appendix C, we discuss how to improve these probabilities to almost 1.) Our attack combines elements of both ciphertexts in an attempt to recover sk_A , which can then be used to decrypt the first ciphertext and obtain sk_B . It is counterintuitive that this is possible, given that it is easy to see that IND-CPA-security guarantees that it is safe for *one* of them to send their message.

Theorem 4.4 Π_{cpa} is not IND-WCIRC-CPA²-secure.

Proof. We give PPT adversary $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ such that $\text{Adv}_{\Pi_{\text{cpa}}, \mathcal{A}}^{2\text{-wcirc-cpa}}(\lambda)$ is equal to $1/2$. Since IND-WCIRC-CPA security requires that this advantage be negligible, this attack breaks security. The adversary proceeds as follows. The first stage of the adversary, $\mathcal{A}_1$, obtains the two public keys, which we will write as pk_A and pk_B , and an encrypted cycle, which we will write as (C_A, C_B) .

If both keys have $\beta = 0$ or $\beta = 1$ (call this event E_1), the adversary aborts and instructs the second stage ($\mathcal{A}_2$) to output a random bit. Since the two keys are independently generated by the challenger, this event will occur with probability exactly $1/2$. Below we will condition on E_1 not happening, and wlog assume that $pk_A = (0, e(g, h)^{a_1}, g^{a_2})$ and $pk_B = (1, e(g, h)^{b_1}, h^{b_2})$. The corresponding secret keys $sk_A = (0, a_1, a_2)$, $sk_B = (1, b_1, b_2)$ are not known to the adversary.

We write the given ciphertexts $C_A = (c_{A,1}, c_{A,2}, c_{A,3}, c_{A,4})$ and $C_B = (c_{B,1}, c_{B,2}, c_{B,3}, c_{B,4})$. $\mathcal{A}_1$ will output two arbitrary distinct messages, and request that the challenge use pk_A . For the state passed to $\mathcal{A}_2$, it now computes:

$$X := c_{B,2} \cdot \frac{e(c_{A,1}, c_{B,3})}{e(c_{A,3}, c_{B,1})}.$$

$\mathcal{A}_1$ sets $\hat{sk}_A = \text{decode}(c_{B,4} \oplus F(X))$ and passes this with the challenge messages as state to $\mathcal{A}_2$.

$\mathcal{A}_2$ receives a ciphertext y and the passed state. It parses $\hat{sk}_A$ as a secret key for Π_{cpa} and computes $\text{Dec}(\hat{sk}_A, y)$, and tests if this is equal to either of the challenge messages. If so, it outputs the corresponding bit. Otherwise it outputs a random bit.

Let's explore why this test works. Write $C_A = \text{Enc}(pk_A, sk_B)$ and $C_B = \text{Enc}(pk_B, sk_A)$. Then:

$$\begin{aligned} C_A &= (c_{A,1}, c_{A,2}, c_{A,3}, c_{A,4}) \\ &= (g^r, R \cdot e(g, h)^{ra_1}, g^{ra_2b_2+b_1}, F(R) \oplus \text{encode}(sk_B)) \\ C_B &= (c_{B,1}, c_{B,2}, c_{B,3}, c_{B,4}) \\ &= (h^s, S \cdot e(g, h)^{sb_1}, h^{sa_2b_2}, F(S) \oplus \text{encode}(sk_A)) \end{aligned}$$

for some $r, s \in \mathbb{Z}_p$ and $R, S \in \mathbb{G}_T$. Then we have that:

$$\begin{aligned} X &:= c_{B,2} \cdot \frac{e(c_{A,1}, c_{B,3})}{e(c_{A,3}, c_{B,1})} = S \cdot e(g, h)^{sb_1} \cdot \frac{e(g^r, h^{sa_2b_2})}{e(g^{ra_2b_2+b_1}, h^s)} \\ &= S \cdot e(g, h)^{sb_1} \cdot \frac{e(g, h)^{rsa_2b_2}}{e(g, h)^{rsa_2b_2} \cdot e(g, h)^{sb_1}} = S. \end{aligned}$$

Thus, $\mathcal{A}_1$ recovers $\hat{sk}_A = sk_A$ as $\text{decode}(c_{B,4} \oplus F(S))$, and $\mathcal{A}_2$ will correctly guess bit b in this case.

Write $\hat{b}$ for the output of $\mathcal{A}_2$. We have

$$\begin{aligned}
\text{Adv}_{\Pi_{\text{cpa}}, \mathcal{A}}^{2\text{-wcirc-cpa}}(\lambda) &= 2 \Pr[\hat{b} = b] - 1 \\
&= 2(\Pr[\hat{b} = b | E_1] \Pr[E_1] + \\
&\quad \Pr[\hat{b} = b | \neg E_1] \Pr[\neg E_1]) - 1 \\
&= 2(1 \cdot 1/2 + 1/2 \cdot 1/2) - 1 \\
&= 1/2
\end{aligned}$$

This completes the proof. $\square$

4.4 Extension: A Counterexample for CCA Security

We show that there exists an IND-CCA-secure cryptosystem, which suffers a complete break when Alice and Bob trade secret keys over an insecure channel; i.e., transmit the two-key cycle $E(pk_A, sk_B)$ and $E(pk_B, sk_A)$. Our construction follows the “double-encryption” approach to building IND-CCA systems from IND-CPA systems as pioneered by Naor and Yung [33] and refined by Dolev, Dwork and Naor [18] and Sahai [37]. Our building blocks will be:

1. The IND-CPA-secure cryptosystem $\Pi_{\text{cpa}} = (G, E, D)$ from Section 4. Let $E(pk, m; r)$ be the encryption of m under public key pk with randomness r .
2. An adaptively non-malleable non-interactive zero-knowledge (NIZK) proof system with unpredictable simulated proofs and uniquely applicable proofs for the language L of consistent pairs of encryptions, defined as:

$$L = \left\{ (e_0, e_1, c_0, c_1) : \begin{array}{l} \exists m, r_0, r_1 \in \{0, 1\}^* \text{ s.t.} \\ c_0 = E(e_0, m; r_0) \text{ and } c_1 = E(e_1, m; r_1) \end{array} \right\}.$$

A proof system for L can be realized under relatively mild assumptions, such as the difficulty of factoring Blum integers (e.g., [37]). One complication is that the secret keys for this cryptosystem now change and the construction must be adapted accordingly, so that the secret key can still be recovered by the adversary during a circular attack. We show that this is possible.

5 Conclusion and Open Problems

In this work, we presented a natural relaxation of the circular security definition, which may prove interesting for positive results in its own right. We demonstrated that its guarantees are *not* already captured by standard definitions of encryption. To do this, we presented symmetric and public-key encryption systems that are secure in the IND-CPA and IND-CCA sense, but fail catastrophically in the presence of an encrypted cycle. This provides the first answer to the foundational question on whether IND-CCA-security captures (weak or regular) circular security for all cycles larger than self-loops. In either case, it does not.

Our work leaves open the interesting problem of finding a public-key counterexample for cycles of size ≥ 3 . Secondly, while our symmetric counterexample depended only on the existence of AE-secure symmetric encryption, our public-key counterexample, like that of Acar et al. [2], required a specific bilinear map assumption. It would be highly interesting to find a counterexample assuming only that IND-CPA- or IND-CCA-secure systems exist.

Finally, we observe that our public-key counterexample contains a novel and curious property – *certain combinations of independently generated ciphertexts trigger the release of their underlying plaintext*. From Rabin’s $\frac{1}{2}$ -OT system to DH-DDH gap groups, the cryptographic community has a strong history of turning such oddities to an advantage. If we view a cryptosystem with this property as a new primitive, what new functionalities can be realized using it?

Acknowledgments

The authors thank Ronald Rivest for the suggestion to view the public key counterexample in Section 4 as a potential building block for other functionalities.

References

- [1] Martin Abadi and Phillip Rogaway. Reconciling two views of cryptography (the computational soundness of formal encryption). *J. Cryptology*, 15(2):103–127, 2002.
- [2] Tolga Acar, Mira Belenkiy, Mihir Bellare, and David Cash. Cryptographic agility and its relation to circular encryption. In *EUROCRYPT '10*, volume 6110 of LNCS, pages 403–422. Springer, 2010.
- [3] Pedro Adao, Gergei Bana, Jonathan Herzog, and Andre Scedrov. Soundness of formal encryption in the presence of key-cycles. In *ESORICS '05*, volume 3679 of LNCS, pages 374–396, 2005.
- [4] Adi Akavia, Shafi Goldwasser, and Vinod Vaikuntanathan. Simultaneous hardcore bits and cryptography against memory attacks. In *TCC '09*, volume 5444 of LNCS, pages 474–495, 2009.
- [5] Benny Applebaum, David Cash, Chris Peikert, and Amit Sahai. Fast cryptographic primitives and circular-secure encryption based on hard learning problems. In *CRYPTO '09*, volume 5677 of LNCS, pages 595–618, 2009.
- [6] Giuseppe Ateniese, Jan Camenisch, and Breno de Medeiros. Untraceable RFID tags via insubvertible encryption. In *CCS '05*, pages 92–101, 2005.
- [7] M. Backes, B. Pfitzmann, and A. Scedrov. Key-dependent message security under active attacks - BRSIM/UC-soundness of Dolev-Yao-style encryption with key cycles. *J.of Comp.Security*, 16(5):497–530, 2008.
- [8] Lucas Ballard, Matthew Green, Breno de Medeiros, and Fabian Monrose. Correlation-resistant storage. Technical Report TR-SP-BGMM-050705, Johns Hopkins University, CS Dept, 2005. <http://spar.isi.jhu.edu/~mgreen/correlation.pdf>.
- [9] Mira Belenkiy, Melissa Chase, Markulf Kolweiss, and Anna Lysyanskaya. Non-interactive anonymous credentials. In *TCC '08*, volume 4948 of LNCS, pages 356–374, 2008.
- [10] Mihir Bellare and Chanathip Namprempre. Authenticated encryption: Relations among notions and analysis of the generic composition paradigm. *J. Cryptology*, 21(4):469–491, 2008.
- [11] John Black, Phillip Rogaway, and Thomas Shrimpton. Encryption-scheme security in the presence of key-dependent messages. In *SAC*, volume 2595 of LNCS, pages 62–75, 2002.
- [12] Dan Boneh, Xavier Boyen, and Hovav Shacham. Short group signatures. In *CRYPTO '04*, volume 3152 of LNCS, pages 45–55, 2004.
- [13] Dan Boneh, Shai Halevi, Michael Hamburg, and Rafail Ostrovsky. Circular-Secure Encryption from Decision Diffie-Hellman. In *CRYPTO '08*, volume 5157 of LNCS, pages 108–125, 2008.
- [14] Jan Camenisch, Nishanth Chandran, and Victor Shoup. A public key encryption scheme secure against key dependent chosen plaintext and adaptive chosen ciphertext attacks. In *EUROCRYPT '09*, volume 5479 of LNCS, pages 351–368, 2009.
- [15] Jan Camenisch and Anna Lysyanskaya. An efficient system for non-transferable anonymous credentials with optional anonymity revocation. In *EUROCRYPT '01*, volume 2045 of LNCS, pages 93–118, 2001.

- [16] Ran Canetti, Oded Goldreich, and Shai Halevi. The random oracle methodology, revisited. *J. of the ACM*, 51(4):557–594, 2004.
- [17] Yevgeniy Dodis, Yael Tauman Kalai, and Shachar Lovett. On cryptography with auxiliary input. In *STOC '09*, pages 621–630, 2009.
- [18] Danny Dolev, Cynthia Dwork, and Moni Naor. Nonmalleable cryptography. *SIAM J. Computing*, 30(2):391–437, 2000.
- [19] Taher El Gamal. A public key cryptosystem and a signature scheme based on discrete logarithms. In *CRYPTO '84*, pages 10–18, 1984.
- [20] Steven D. Galbraith. Supersingular curves in cryptography. In *ASIACRYPT '01*, volume 2248 of LNCS, pages 495–513, 2001.
- [21] S. Goldwasser and S. Micali. Probabilistic encryption. *J. Comput. Syst. Sci.*, 28(2):270–299, 1984.
- [22] Shafi Goldwasser and Yael Tauman Kalai. On the (In)security of the Fiat-Shamir Paradigm. In *FOCS '03*, page 102, 2003.
- [23] Matthew Green and Susan Hohenberger. Universally composable adaptive oblivious transfer. In *ASIACRYPT*, volume 5350, pages 179–197, 2008.
- [24] Jens Groth and Amit Sahai. Efficient non-interactive proof systems for bilinear groups. In *EUROCRYPT '08*, volume 4965 of LNCS, pages 415–432, 2008.
- [25] Iftach Haitner and Thomas Holenstein. On the (im)possibility of key dependent encryption. In *TCC '09*, volume 5444 of LNCS, pages 202–219, 2009.
- [26] Shai Halevi and Hugo Krawczyk. Security under key-dependent inputs. In *ACM CCS '07*, pages 466–475, 2007.
- [27] Johan Hastad, Russell Impagliazzo, Leonid A. Levin, and Michael Luby. A pseudorandom generator from any one-way function. *SIAM J. Computing*, 28(4):1364–1396, 1999.
- [28] Dennis Hofheinz and Dominique Unruh. Towards key-dependent message security in the standard model. In *EUROCRYPT '08*, volume 4965 of LNCS, pages 108–126, 2008.
- [29] Jonathan Katz and Yehuda Lindell. *Introduction to Modern Cryptography*. Chapman & Hall/CRC, 2008.
- [30] Peeter Laud and Ricardo Corin. Sound computational interpretation of formal encryption with composed keys. In *ICISC*, volume 2971, pages 55–66, 2003.
- [31] Noel McCullagh and Paulo S. L. M. Barreto. A new two-party identity-based authenticated key agreement. In *CT-RSA '04*, volume 3376, pages 262–274, 2004.
- [32] Moni Naor and Gil Segev. Public-key cryptosystems resilient to key leakage. In *CRYPTO '09*, volume 5677 of LNCS, pages 18–35, 2009.
- [33] Moni Naor and Moti Yung. Public-key cryptosystems provably secure against chosen ciphertext attacks. In *STOC '90*, pages 427–437, 1990.
- [34] Charles Rackoff and Daniel R. Simon. Non-interactive zero-knowledge proof of knowledge and chosen ciphertext attack. In *CRYPTO '91*, volume 576 of LNCS, pages 433–444, 1991.
- [35] Phillip Rogaway and Thomas Shrimpton. A provable-security treatment of the key-wrap problem. In *EUROCRYPT*, pages 373–390, 2006.

- [36] Ron Rothblum. On the circular security of bit-encryption. Cryptology ePrint Archive, Report 2012/102, 2012. <http://eprint.iacr.org/>.
- [37] Amit Sahai. Non-malleable non-interactive zero knowledge and adaptive chosen-ciphertext security. In *FOCS '99*, pages 543–553, 1999.
- [38] Mike Scott. Authenticated id-based key exchange and remote log-in with simple token and pin number, 2002. Available at <http://eprint.iacr.org/2002/164>.

A Security Proof for Π_{ae}

A.1 Security against Key Recovery Attacks

It will simplify our results to use the following concept of key recovery security, which is implied by AE security.

Definition A.1 (KR) Let $\Pi = (\text{KeyGen}, \text{Enc}, \text{Dec})$ be a symmetric-key encryption scheme for the message space M . Let the random variable $\text{KR}(\Pi, \mathcal{A}, \lambda)$ be defined by the following probabilistic algorithm:

$$\begin{aligned} & \text{KR}(\Pi, \mathcal{A}, \lambda) \\ & \quad K \leftarrow \text{KeyGen}(1^\lambda) \\ & \quad \hat{K} \leftarrow \mathcal{A}^{\mathcal{E}_K^{\text{kr}}(\cdot), \mathcal{D}_K^{\text{kr}}(\cdot)}(1^\lambda) \\ & \quad \text{Output } (\hat{K} \stackrel{?}{=} K). \end{aligned}$$

Here the oracle $\mathcal{E}_K^{\text{kr}}(\cdot)$ takes as input a message $m \in M$ and returns $\text{Enc}(K, m)$, and the oracle $\mathcal{D}_K^{\text{kr}}(\cdot)$ takes as input a ciphertext and returns $\text{Dec}(K, c)$.

We denote the KR advantage of $\mathcal{A}$ by

$$\text{Adv}_{\Pi, \mathcal{A}}^{\text{kr}}(\lambda) = \Pr[\text{KR}(\Pi, \mathcal{A}, \lambda) = 1].$$

We say that Π is KR secure if $\text{Adv}_{\Pi, \mathcal{A}}^{\text{kr}}(\lambda)$ is negligible for all PPT $\mathcal{A}$.

We will use the following theorem below. The proof is standard.

Theorem A.2 Any AE-secure symmetric-key encryption scheme is also KR-secure.

A.2 Proof of Security for System Π_{ae}

Theorem A.3 Encryption scheme Π_{ae} is AE secure whenever Π'_{ae} is AE secure.

Proof. We prove the theorem by giving a reduction to the AE security of Π'_{ae} . We proceed by describing a pair of hybrid games, where the first H_0 is defined to be the AE experiment from Definition 2.3 with Π_{ae} , and the second is a modified experiment that will be seen to be essentially equivalent to the AE experiment with Π'_{ae} .

We denote the hybrids H_0, H_1 , and define them as follows:

H_0 : The AE experiment with Π_{ae} .

H_1 : Exactly as in H_0 , except that the oracles $\mathcal{E}_{K,b}^{\text{ae}}(\cdot, \cdot)$ and $\mathcal{D}_{K,b}^{\text{ae}}(\cdot)$ use modified versions of the algorithms Enc and Dec which ignore their “If” statements and proceed directly the “Else” clause.

Fix some PPT adversary $\mathcal{A}$, and let

$$\text{Adv}_{\mathcal{A}}^{H_i}(\lambda) = 2 \Pr[H_i(\mathcal{A}, \lambda) = 1] - 1$$

for $i = 0, 1$. Then we have

$$\text{Adv}_{\mathcal{A}}^{H_0}(\lambda) = \text{Adv}_{\Pi_{\text{ae}}, \mathcal{A}}^{\text{ae}}(\lambda), \tag{1}$$

which is negligible by assumption. Next we relate $\text{Adv}_{\mathcal{A}}^{H_0}(\lambda)$ and $\text{Adv}_{\mathcal{A}}^{H_1}(\lambda)$.

Lemma A.4 For all PPT adversaries $\mathcal{A}$,

$$\text{Adv}_{\mathcal{A}}^{\text{H}_0}(\lambda) - \text{Adv}_{\mathcal{A}}^{\text{H}_1}(\lambda) \leq \epsilon_1(\lambda) \quad (2)$$

for some negligible function ϵ_1 .

Proof. Suppose to the contrary that a PPT adversary $\mathcal{A}$ exists that violates (2). Using $\mathcal{A}$ we construct an PPT adversary $\mathcal{B}$ such that $\text{Adv}_{\Pi'_{\text{ae}}, \mathcal{B}}^{\text{kr}}(\lambda)$ is non-negligible which contradicts the AE security Π'_{ae} by Theorem A.2.

The adversary $\mathcal{B}$ has access to two oracles in the KR experiment with Π'_{ae} , $\mathcal{E}_K^{\text{kr}}(\cdot)$ and $\mathcal{D}_K^{\text{kr}}(\cdot)$. $\mathcal{B}$ will run $\mathcal{A}$, which expects the two oracles $\mathcal{E}_{K,b}^{\text{ae}}(\cdot, \cdot)$, $\mathcal{D}_{K,b}^{\text{ae}}(\cdot)$ in the AE experiment with Π_{ae} .

$\mathcal{B}$ starts by selecting $b \xleftarrow{r} \{0, 1\}$ and initializing a list $\mathbf{L}$ to be empty. $\mathcal{B}$ then runs $\mathcal{A}$, simulating queries to $\mathcal{E}_{K,b}^{\text{ae}}(\cdot, \cdot)$ and $\mathcal{D}_{K,b}^{\text{ae}}(\cdot)$ as follows:

$\frac{\mathcal{E}_{K,b}^{\text{ae}}(m_0, m_1)}{\text{UseCycle}(m_b)} \\ \text{Return } \mathcal{E}_K^{\text{kr}}(m_b)$	$\frac{\mathcal{D}_{K,b}^{\text{ae}}(c)}{\text{If } b = 0 \text{ then}} \\ \text{Return } \perp$
$\frac{\text{UseCycle}(x)}{\text{If } x \neq np(\lambda) \text{ then}} \\ \text{Return}$	Else
$\text{Parse } x \text{ as } (c_1, \dots, c_n)$	$\text{Parse } c \text{ as } \tilde{K} \parallel \tilde{m}$
$K_2 \leftarrow \mathcal{D}_K^{\text{kr}}(c_1)$	$\text{Add } \tilde{K} \text{ to } \mathbf{L}$
$\text{For } i = 2 \text{ to } n$	$\text{Return } \mathcal{D}_K^{\text{kr}}(c)$
$K_{i \bmod n+1} \leftarrow \text{Dec}'(K_i, c_i)$	
$\text{Add } K_1 \text{ to } \mathbf{L}$	

When $\mathcal{A}$ halts, $\mathcal{B}$ selects and outputs $\hat{K}$ at random from $\mathbf{L}$.

Before moving on, let us intuitively explain how $\mathcal{B}$ is simulating the game. We have implemented the oracle simulation so that $\mathcal{B}$ assumes that the “If” statements in both oracles do not ever pass, and indeed it properly simulates both hybrids as long as this is case. It keeps track of the keys induced by the queries of $\mathcal{A}$ which might have caused an “If” statement to pass, and afterwards it chooses a random one and hopes it was the first such query.

Let E be the event that $\mathcal{A}$ queries either $\mathcal{E}_{K,b}^{\text{ae}}(\cdot, \cdot)$ or $\mathcal{D}_{K,b}^{\text{ae}}(\cdot)$ at a point that causes their “If” statements to evaluate to true. It is apparent that H_0 and H_1 are identical unless E occurs, so we have

$$\text{Adv}_{\mathcal{A}}^{\text{H}_0}(\lambda) - \text{Adv}_{\mathcal{A}}^{\text{H}_1}(\lambda) \leq \Pr[E].$$

Conditioned on E occurring, we have that $\hat{K}$ is equal to K (where K was chosen in the KR experiment) with probability $1/Q$, where Q is (polynomial) number of queries issued by $\mathcal{A}$. This follows from the fact that $\mathcal{B}$ perfectly simulates H_0 until the first query that triggers the event E . Thus, $\mathcal{B}$ recovers the secret key with probability at least $\text{Adv}_{\Pi'_{\text{ae}}, \mathcal{B}}^{\text{kr}}(\lambda) = \Pr[E]/Q$. But this is negligible by the assumption that Π'_{ae} is AE-secure and hence KR-secure, which bounds $\Pr[E]$ by a negligible function. $\square$

Lemma A.5 For every PPT adversary $\mathcal{A}$

$$\text{Adv}_{\Pi'_{\text{ae}}, \mathcal{A}}^{\text{ae}}(\lambda) = \text{Adv}_{\mathcal{A}}^{\text{H}_1}(\lambda). \quad (3)$$

Proof. This lemma follows by the observation that in H_1 , $\mathcal{A}$ is interacting with oracles that are functionally identical to those in $\text{AE}(\Pi'_{\text{ae}}, \mathcal{A}, \lambda)$. The only difference is in the message space restriction in H_1 , which is a strict subset of those allowed in $\text{AE}(\Pi'_{\text{ae}}, \mathcal{A}, \lambda)$. $\square$

Finally, we observe that $\text{Adv}_{\Pi'_{\text{ae}}, \mathcal{B}}^{\text{ae}}(\lambda)$ is negligible by assumption. Combining this observation with (1), (2) and (3) proves the theorem. $\square$

B Security Proof for System Π_{cpa}

We first recall Theorem 4.3.

Theorem B.1 *Encryption scheme Π_{cpa} is IND-CPA secure under the Decisional Diffie-Hellman Assumption in $\mathbb{G}_1$ and $\mathbb{G}_2$ (SXDH).*

Proof. To show that scheme Π_{cpa} meets security Definition 2.1, suppose PPT adversary $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ has advantage ϵ in the $\text{IND-CPA}(\Pi_{\text{cpa}}, \mathcal{A}, \lambda)$ experiment. Let $\psi(\cdot)$ be some polynomial function that will be determined in the proof. Using a series of hybrid games we show that if all PPT adversaries have negligible advantage ϵ_1 in solving the DDH problem in $\mathbb{G}_1$ or $\mathbb{G}_2$ and advantage $\psi(\epsilon_1)$ at distinguishing the PRG F (secure under DDH) from a random function, then ϵ is bounded by the negligible value $4\epsilon_1 + 2\psi(\epsilon_1)$.

In all hybrids, the adversary plays the IND-CPA game with a challenger. The public key is distributed normally, but the structure of the challenge ciphertext differs between the hybrids. Let $\text{CT} = (C_1, C_2, C_3, C_4)$ denote the challenge ciphertext computed in IND-CPA and let $R_2 \xleftarrow{r} \mathbb{G}_T$, $R_3 \xleftarrow{r} \mathbb{G}_1$ (if $\beta = 0$) or $R_3 \xleftarrow{r} \mathbb{G}_2$ (if $\beta = 1$) and $R_4 \xleftarrow{r} \{0, 1\}^{|C_4|}$ be randomly chosen. The hybrids are as follows:

- H_0 : The challenge ciphertext is $\text{CT} = (C_1, C_2, C_3, C_4)$.
- H_1 : The challenge ciphertext is $\text{CT}_1 = (C_1, R_2, C_3, C_4)$.
- H_2 : The challenge ciphertext is $\text{CT}_2 = (C_1, R_2, R_3, C_4)$.
- H_3 : The challenge ciphertext is $\text{CT}_3 = (C_1, R_2, R_3, R_4)$.

We will write $\text{Adv}_{\mathcal{A}}^{H_i}(\lambda)$ to denote the advantage of $\mathcal{A}$ in H_i , i.e., $2\Pr[H_i(\mathcal{A}, \lambda) = 1] - 1$. By definition, the ciphertext in H_0 is as in $\text{IND-CPA}(\Pi_{\text{cpa}}, \mathcal{A}, \lambda)$, while the challenge ciphertext in hybrid H_3 information-theoretically hides the plaintext. We argue that under the DDH assumption in $\mathbb{G}_1$ and $\mathbb{G}_2$, for all PPT $\mathcal{A}$, we have

$$\text{Adv}_{\mathcal{A}}^{H_0} - \text{Adv}_{\mathcal{A}}^{H_3} \leq 2\epsilon_1 + \psi(\epsilon_1). \quad (4)$$

It remains to observe that, by definition,

$$\text{Adv}_{\mathcal{A}}^{H_0}(\lambda) = \text{IND-CPA}(\Pi_{\text{cpa}}, \mathcal{A}, \lambda), \quad (5)$$

and

$$\text{Adv}_{\mathcal{A}}^{H_3}(\lambda) = 0 \quad (6)$$

because the adversary's output is independent of the bit b it is trying to guess.

We now turn to proving (4). We start by bounding the difference in advantage between H_0 and H_1 .

Lemma B.2 *For all PPT $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$, if the DDH assumption holds in $\mathbb{G}_1$ and $\mathbb{G}_2$, then*

$$\text{Adv}_{\mathcal{A}}^{H_1}(\lambda) - \text{Adv}_{\mathcal{A}}^{H_0}(\lambda) \leq \epsilon_1.$$

Proof. Let $(e, p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, g = \langle \mathbb{G}_1 \rangle, h = \langle \mathbb{G}_2 \rangle)$ be the common parameters. Suppose for contradiction that an adversary $\mathcal{A}$ violates the inequality in the lemma. Then, we construct an adversary $\mathcal{A}'$ that decides the DDH problem in $\mathbb{G}_1$ or $\mathbb{G}_2$ with advantage ϵ' . $\mathcal{A}'$ works as follows.

1. Sample a bit $\beta \leftarrow \{0, 1\}$.
2. Obtain a DDH problem instance:

$$\Gamma = \begin{cases} (g, g^a, g^b, G) \in \mathbb{G}_1^4 & \text{if } \beta = 0; \\ (h, h^a, h^b, H) \in \mathbb{G}_2^4 & \text{if } \beta = 1. \end{cases}$$

3. Sample $v \leftarrow \mathbb{Z}_p^*$.
4. Set the public key as:

$$pk = \begin{cases} (0, e(g^a, h), g^v) \in \{0, 1\} \times \mathbb{G}_T \times \mathbb{G}_1 & \text{if } \beta = 0; \\ (1, e(g, h^a), h^v) \in \{0, 1\} \times \mathbb{G}_T \times \mathbb{G}_2 & \text{if } \beta = 1. \end{cases}$$

5. Run $\mathcal{A}_1(pk)$ to produce a tuple (M_0, M_1, z) . Parse M_0 as (α, m_1, m_2) .
6. Sample $R \leftarrow \mathbb{G}_T$ and set $I \leftarrow F(R) \oplus \text{encode}(M_0)$.
7. Set the challenge ciphertext as:

$$C = \begin{cases} (g^b, R \cdot e(G, h), (g^b)^{vm_2} \cdot g^{m_1}, I) & \text{if } \beta = 0; \\ (h^b, R \cdot e(g, H), (h^b)^{vm_2}, I) & \text{if } \beta = 1. \end{cases}$$

Note that in the first case, $C \in \mathbb{G}_1 \times \mathbb{G}_T \times \mathbb{G}_1 \times \{0, 1\}^{\ell(\lambda)}$, while in the second case $C \in \mathbb{G}_2 \times \mathbb{G}_T \times \mathbb{G}_2 \times \{0, 1\}^{\ell(\lambda)}$.

8. Run $\mathcal{A}_2(C, z)$ and output whatever it outputs.

We argue that when Γ is a proper DDH instance, $\mathcal{A}'$ perfectly simulates the experiment $\mathbf{H}_0$. The distribution of keys and encryption values are exactly as they should be. When Γ is not a DDH instance, $\mathcal{A}'$ perfectly simulates the experiment $\mathbf{H}_1$. The only impacted ciphertext part is C_2 , where the proper public key information has been replaced by a random value. Thus, $\mathcal{A}'$'s advantage in solving DDH in $\mathbb{G}_1$ or $\mathbb{G}_2$ will be ϵ' . Under the DDH assumption in $\mathbb{G}_1, \mathbb{G}_2$, $\epsilon' \leq \epsilon_1$. $\square$

Lemma B.3 *For all PPT $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$, if the DDH assumption holds in $\mathbb{G}_1$ and $\mathbb{G}_2$, then*

$$\text{Adv}_{\mathcal{A}}^{\mathbf{H}_2}(\lambda) - \text{Adv}_{\mathcal{A}}^{\mathbf{H}_1}(\lambda) \leq \epsilon_1.$$

Proof. Suppose adversary $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ violates the lemma. Then, we construct an adversary $\mathcal{A}'$ that decides the DDH problem in $\mathbb{G}_1$ or $\mathbb{G}_2$ with advantage ϵ' as follows. Let $(e, p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, g = \langle \mathbb{G}_1 \rangle, h = \langle \mathbb{G}_2 \rangle)$ be the common parameters. $\mathcal{A}'$ works as follows:

1. Sample a bit $\beta \leftarrow \{0, 1\}$.
2. Obtain a DDH problem instance:

$$\Gamma = \begin{cases} (g, g^a, g^b, G) \in \mathbb{G}_1^4 & \text{if } \beta = 0; \\ (h, h^a, h^b, H) \in \mathbb{G}_2^4 & \text{if } \beta = 1. \end{cases}$$

3. Sample $v \leftarrow \mathbb{Z}_p^*$.
4. Set the public key as:

$$pk = \begin{cases} (0, e(g, h)^v, g^a) \in \{0, 1\} \times \mathbb{G}_T \times \mathbb{G}_1 & \text{if } \beta = 0; \\ (1, e(g, h)^v, h^a) \in \{0, 1\} \times \mathbb{G}_T \times \mathbb{G}_2 & \text{if } \beta = 1. \end{cases}$$

5. Run $\mathcal{A}_1(pk)$ to produce a tuple (M_0, M_1, z) . Parse M_0 as (α, m_1, m_2) .
6. Sample $R, R_2 \leftarrow \mathbb{G}_T$ and set $I \leftarrow F(R) \oplus \text{encode}(M_0)$.
7. Set the challenge ciphertext as:

$$C = \begin{cases} (g^b, R_2, G^{m_2} \cdot g^{m_1}, I) & \text{if } \beta = 0; \\ (h^b, R_2, H^{m_2}, I) & \text{if } \beta = 1. \end{cases}$$

8. Run $\mathcal{A}_2(C, z)$ and output whatever it outputs.

When Γ is a proper DDH instance, $\mathcal{A}'$ perfectly simulates experiment $\mathbf{H}_1$. When Γ is not a DDH instance, $\mathcal{A}'$ perfectly simulates experiment $\mathbf{H}_2$. The only impacted ciphertext part is C_3 , where the proper public key information has been replaced by a random value. Thus, $\mathcal{A}'$'s advantage in solving DDH in $\mathbb{G}_1$ or $\mathbb{G}_2$ will be ϵ' . Under the DDH assumption in $\mathbb{G}_1, \mathbb{G}_2$, $\epsilon' \leq \epsilon_1$. $\square$

Lemma B.4 *For all PPT $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ if F is secure under the DDH assumption in $\mathbb{G}_1, \mathbb{G}_2$ then*

$$\text{Adv}_{\mathcal{A}}^{\mathbf{H}_3}(\lambda) - \text{Adv}_{\mathcal{A}}^{\mathbf{H}_2}(\lambda) \leq \psi(\epsilon_1).$$

Proof. Let $(e, p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, g = \langle \mathbb{G}_1 \rangle, h = \langle \mathbb{G}_2 \rangle)$ be the common parameters. Note that in our construction, F has domain $\mathbb{G}_T$ and range $\{0, 1\}^{\ell(\lambda)}$.⁴ Let us suppose that adversary $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ violates the lemma. Then, we construct an adversary $\mathcal{A}'$ that breaks the security of the PRG F with advantage ϵ' . $\mathcal{A}'$ accepts as input a value I' sampled from ensemble E_b where $E_0 = \{R \leftarrow \mathbb{G}_T : F(R)\}_\lambda$, $E_1 = \{U_{\ell(\lambda)}\}_\lambda$ and $b \in \{0, 1\}$ and operates as follows:

1. Compute $(pk, sk) \leftarrow \text{KeyGen}(1^k)$ and parse $pk = (\beta, Y_1, Y_2)$.
2. Run $\mathcal{A}_1(pk)$ to produce a tuple (M_0, M_1, z) .
3. Sample $r \leftarrow \mathbb{Z}_p$, $R_2 \leftarrow \mathbb{G}_T$ and $R_3 \leftarrow \mathbb{G}_1$ (if $\beta = 0$) or $R_3 \leftarrow \mathbb{G}_2$ (if $\beta = 1$). Set $I \leftarrow I' \oplus \text{encode}(M_0)$. Compute the challenge ciphertext as follows:

$$C = \begin{cases} (g^r, R_2, R_3, I) & \text{if } \beta = 0; \\ (h^r, R_2, R_3, I) & \text{if } \beta = 1. \end{cases}$$

4. Run $\mathcal{A}_2(C, z)$ and output whatever it outputs.

If I' is sampled from distribution E_0 then $\mathcal{A}'$ perfectly simulates H_2 . If I' is sampled from the uniform distribution E_1 , then $I' \oplus \text{encode}(M_0)$ is uniformly distributed in $\{0, 1\}^{\ell(\lambda)}$ and $\mathcal{A}'$ perfectly simulates H_3 . Additionally, R is independent of the adversary's view. Thus $\mathcal{A}'$'s advantage in distinguishing the two distributions will be ϵ' . Under the DDH assumption, we have $\epsilon' \leq \psi(\epsilon_1)$. $\square$

We complete the proof of the theorem by combining (4), (5), (6), and Lemmas B.2, B.3, and B.4. $\square$

C An Alternative Counterexample for CPA Security

As mentioned in Section 4, one “artificial” feature of the cryptosystem Π_{cpa} is that the **KeyGen** algorithm randomly embeds the public key into either $\mathbb{G}_1$ or $\mathbb{G}_2$ with probability $1/2$ and then the group setting of the ciphertext also differs depending on the public key. We know of no deployed cryptosystems that alternate the setting of keys in such a manner. Some readers might hope that this property renders our result inapplicable to the domain of “practical” cryptosystems, i.e., to assume that cryptosystems with a single, defined key and ciphertext structure are immune to the concerns we note here. We must disappoint these readers.

Below we propose an alternative IND-CPA-secure scheme Π'_{cpa} that does not exhibit this “group switching” feature, and yet still breaks catastrophically in the face of a 2-cycle. *Indeed, this result is even stronger than that of Section 4 since it permits an adversary to win the IND-CIRC-CPA game with a higher probability.* Π'_{cpa} has keys and ciphertexts that are twice the length of those in Π_{cpa} .

Construction Π'_{cpa} Cryptosystem $\Pi'_{\text{cpa}} = (\text{KeyGen}', \text{Enc}', \text{Dec}')$ uses $\Pi_{\text{cpa}} = (\text{KeyGen}, \text{Enc}, \text{Dec})$ as a building block. As before we assume that a single set of bilinear group parameters will be shared across all keys generated at a given security level and are implicitly provided to all algorithms. Let $\mathcal{M}$ be the message space of Π_{cpa} . Then the message space for Π'_{cpa} is $\mathcal{M}' = \mathcal{M} \times \mathcal{M}$. We define the system as follows.

KeyGen' (1^λ) . The key generation algorithm runs **KeyGen** repeatedly to obtain pk_1, sk_1 and pk_2, sk_2 where $pk_1 = (0, \cdot, \cdot)$ and $pk_2 = (1, \cdot, \cdot)$.⁵ The public key is set as $pk = (pk_1, pk_2)$, and the secret key as $sk = (sk_1, sk_2)$.

Encrypt' (pk, M) . The encryption algorithm parses the public key $pk = (pk_1, pk_2)$, and message $M = (m_1, m_2) \in \mathcal{M}'$. Output the ciphertext C as:

$$C = (\text{Enc}(pk_1, m_2), \text{Enc}(pk_2, m_1))$$

⁴Although this specification differs slightly from Definition 4.2, this specific construction can be constructed from traditional PRGs using standard techniques.

⁵This can be accomplished probabilistically by repeatedly calling **KeyGen** and discarding redundant keypairs; alternatively the **KeyGen** algorithm can be trivially modified to produce the needed keys in only two calls.

$\text{Decrypt}'(sk, C)$. The decryption algorithm parses the secret key $sk = (sk_1, sk_2)$ and the ciphertext $C = (C_1, C_2)$. Next, it computes:

$$M = (\text{Dec}(sk_2, C_2), \text{Dec}(sk_1, C_1))$$

Correctness follows trivially from the correctness of Π_{cpa} .

Theorem C.1 *Encryption scheme Π'_{cpa} is IND-CPA secure under the Decisional Diffie-Hellman Assumption in $\mathbb{G}_1$ and $\mathbb{G}_2$ (SXDH).*

Attack on IND-CIRC-CPA Security The above scheme breaks completely for 2-key cycles.

Theorem C.2 *Encryption scheme Π'_{cpa} is not IND-CIRC-CPA secure for cycles of length 2.*

Proof sketch. To show that scheme Π'_{cpa} is *not* IND-CIRC-CPA-secure for key cycles of length two, we recall the attack of Section 4.3. As in that attack, we assume that the adversary receives $C_A = \text{Enc}(pk_A, sk_B)$ and $C_B = \text{Enc}(pk_B, sk_A)$ or two encryptions of a fixed message, and must distinguish which. Unlike that attack, we do not abort based on the structure of the public keys. Instead we receive $pk_A = (pk_{A,1}, pk_{A,2})$, $pk_B = (pk_{B,1}, pk_{B,2})$, $C_A = (C_{A,1}, C_{A,2})$ and $C_B = (C_{B,1}, C_{B,2})$. Now, there are two options. Either:

1. $C_{A,1} = \text{Enc}(pk_{A,1}, sk_{B,2})$ and $C_{B,2} = \text{Enc}(pk_{B,2}, sk_{A,1})$ and $C_{A,2} = \text{Enc}(pk_{A,2}, sk_{B,1})$ and $C_{B,1} = \text{Enc}(pk_{B,1}, sk_{A,2})$; or
2. $C_{A,1} = \text{Enc}(pk_{A,1}, \alpha_2)$ and $C_{B,2} = \text{Enc}(pk_{B,2}, \alpha_1)$ and $C_{A,2} = \text{Enc}(pk_{A,2}, \alpha_1)$ and $C_{B,1} = \text{Enc}(pk_{B,1}, \alpha_2)$ for any fixed $(\alpha_1, \alpha_2) \in \mathcal{M}'$ as defined by Definition 2.4.

If we are in case 1, then we simply apply the exact attack from Section 4.3 twice to the pairs $(C_{A,1}, C_{B,2})$ and $(C_{A,2}, C_{B,1})$ to recover both secret keys in full $(sk_{A,1}, sk_{A,2})$ and $(sk_{B,1}, sk_{B,2})$ with probability 1. Once this is done and detected, D outputs 1.

If we are in case 2, then let $\alpha_1 = (\cdot, m_1, m_2)$ and $\alpha_2 = (\cdot, m'_1, m'_2)$. Parse $sk_{A,1} = (0, a_1, a_2)$ and $sk_{B,2} = (1, b_1, b_2)$ and we have:

$$\begin{aligned} C_{A,1} &= (c_{A,1}, c_{A,2}, c_{A,3}, c_{A,4}) \\ &= (g^r, R \cdot e(g, h)^{ra_1}, g^{ra_2m'_2+m'_1}, F(R) \oplus \text{encode}(\alpha_2)) \\ C_{B,2} &= (c_{B,1}, c_{B,2}, c_{B,3}, c_{B,4}) \\ &= (h^s, S \cdot e(g, h)^{sb_1}, h^{sm_2b_2}, F(S) \oplus \text{encode}(\alpha_1)) \end{aligned}$$

for some $r, s \in \mathbb{Z}_p$ and $R, S \in \mathbb{G}_T$. Then we have that:

$$\begin{aligned} X &:= c_{B,2} \cdot \frac{e(c_{A,1}, c_{B,3})}{e(c_{A,3}, c_{B,1})} = S \cdot e(g, h)^{sb_1} \cdot \frac{e(g^r, h^{sm_2b_2})}{e(g^{ra_2m'_2+m'_1}, h^s)} \\ &= S \cdot e(g, h)^{s(b_1-m'_1)} \cdot (e(g, h)^{s(m_2b_2-m'_2a_2)})^r \end{aligned}$$

Now, D will return 1 if and only if $sk_A = \text{decode}((F(S) \oplus \text{encode}(\alpha_1)) \oplus F(X))$. What is the probability that this event occurs? First, suppose that $s(m_2b_2 - m'_2a_2) \bmod p \neq 0$ (event E_1), which happens with probability $\geq 1 - 3/(p-1) = (p-4)/(p-1)$ for honest executions. Next, consider the values α_1, α_2, s, S as fixed and r is the only variable. What is the chance that the challenger's random choice of r will induce a value X such that $F(X) = F(S) \oplus \text{encode}(\alpha_1) \oplus \text{encode}(sk_A)$? First, we observe that since $s(m_2b_2 - m'_2a_2) \neq 0$ and r is chosen uniformly at random in $\mathbb{Z}_p$, then X is also distributed uniformly at random in $\mathbb{G}_T$. Thus, by the assumption that F is computationally indistinguishable from a uniform, random function, D will incorrectly guess a key cycle in this case with probability at most $2^{-\ell(\lambda)}$ plus a negligible amount $\nu(\lambda)$, where λ is the security parameter.

Thus, D 's total probability of success in this attack is:

$$\begin{aligned}
\Pr[D \text{ wins}] &= \Pr[\text{Case 1}] \cdot \Pr[D \text{ wins} \mid \text{Case 1}] \\
&\quad + \Pr[\text{Case 2}] \cdot \Pr[D \text{ wins} \mid \text{Case 2}] \\
&\geq \frac{1}{2} \cdot 1 + \frac{1}{2} \cdot (\Pr[E_1] \cdot \Pr[D \text{ wins} \mid E_1]) \\
&\geq \frac{1}{2} + \frac{1}{2} \cdot \left(\frac{p-4}{p-1} \cdot (1 - 2^{-\ell(\lambda)} - \nu(\lambda)) \right) \\
&\geq \frac{3}{4} - \frac{(2^{-\ell(\lambda)} + \nu(\lambda))}{2} \quad \text{for all } p \geq 7
\end{aligned}$$

Of course, for practical 80-bit or higher values of p , this probability is much closer to 1. □

Billion-Gate Secure Computation with Malicious Adversaries

Benjamin Kreuter
brk7bx@virginia.edu
University of Virginia

abhi shelat
abhi@virginia.edu
University of Virginia

Chih-hao Shen
cs6zb@virginia.edu
University of Virginia

Abstract

The goal of this paper is to assess the feasibility of two-party secure computation in the presence of a malicious adversary. Prior work has shown the feasibility of billion-gate circuits in the semi-honest model, but only the 35k-gate AES circuit in the malicious model, in part because security in the malicious model is much harder to achieve. We show that by incorporating the best known techniques and parallelizing almost all steps of the resulting protocol, evaluating billion-gate circuits is feasible in the malicious model. Our results are in the standard model (i.e., no common reference strings or PKIs) and, in contrast to prior work, we do not use the random oracle model which has well-established theoretical shortcomings.

1 Introduction

Protocols for secure computation allow two or more mutually distrustful parties to collaborate and compute some function on each other's inputs, with privacy and correctness guarantees. Andrew Yao showed that secure two-party protocols can be constructed for any computable function [34]. Yao's protocol involves representing the function as a boolean circuit and having one party (called the *generator*) encrypt the circuit in such a way that it can be selectively decrypted by the other party (called the *evaluator*) to compute the output, a process called *garbling*. In particular, oblivious transfers are used for the evaluator to obtain a subset of the decryption keys that are needed to compute the output of the function.

Yao's protocol is of great practical significance. In many real-world situations, the inputs to a function may be too valuable or sensitive to share. Huang et al. explored the use of secure computation for biometric identification [14] in national security applications, in which it is desirable for individual genetic data to be kept private but still checked against a classified list. In a similar

security application, Osadchy et al. described how face recognition could be performed in a privacy-preserving manner [30]. The more general case of multiparty computation has already seen real-world use in computing market clearing prices in Denmark [2].

Yao's original protocol ensures the privacy of each party's input and the correctness of the output under the *semi-honest* model, in which both parties follow the protocol honestly. This model has been the basis for several scalable secure computation systems [4, 10, 12, 13, 17, 22, 26]. It is conceivable, however, that one of the parties may deviate from the protocol in an attempt to violate privacy or correctness. Bidders may attempt to manipulate the auction output in their favor; spies may attempt to obtain sensitive information; and a computer being used for secure computation may be infected with malware. Securing against *malicious* participants, who may deviate arbitrarily from pre-agreed instructions, in an efficient manner is of more practical importance.

There have been several attempts on practical systems with security against active, malicious adversaries. Lindell and Pinkas presented an approach based on garbled circuits that uses the cut-and-choose technique [23], with an implementation of this system having been given by Pinkas et al. [31]. Nielsen et al. presented the LEGO+ system [29], which uses efficient oblivious transfers and authenticated bits to enforce honest behaviors from participants. shelat and Shen proposed a hybrid approach that integrates sigma protocols into the cut-and-choose technique [33]. The protocol compiler presented by Ishai, Prabhakaran, and Sahai [16] also uses an approach based on oblivious transfer, and was implemented by Lindell, Oxman, and Pinkas [21]. In all these cases, AES was used as a benchmark for performance tests.

Protocols for general multiparty computation with security against a malicious *majority* have also been presented. Canetti et al. gave a construction of a *universally composable* protocol in the *common reference string* model [5]. The protocol compiler of Ishai et al.,

mentioned above, can be used to construct a multiparty protocol with security against a dishonest majority in the UC model [16]. Bendlin et al. showed a construction based on homomorphic encryption [1], which was improved upon by Damgård et al. [7]; these protocols were also proved secure in the UC model, and thus require additional setup assumptions. The protocol of Damgård et al. (dubbed “SPDZ” and pronounced “speedz”) is based on a preprocessing model, which improves the amortized performance. Damgård et al. presented an implementation of their protocol, which could evaluate the function $(x \times y) + z$ in about 3 seconds with a 128 bit security level, but with an amortized time of a few milliseconds.

This paper presents a scalable two-party secure computation system which guarantees privacy and correctness in the presence of a malicious party. The system we present can handle circuits with hundreds of millions or even billions of gates, while requiring relatively modest computing resources. Our system follows the Fairplay framework, allowing general purpose secure computation starting from a high level description of a function. We present a system with numerous technical advantages over the Fairplay system, both in our compiler and in the secure computation protocol. Unlike previous work, we do not rely solely on AES circuits as our benchmark; our goal is to evaluate circuits that are orders of magnitude larger than AES in the malicious model, and we use AES only as a comparison with other work. We prove the security of our protocol assuming *circular 2-correlation robust* hash functions and the hardness of the elliptic curve discrete logarithm problem, and require neither additional setup assumptions nor the random oracle model.

2 Contributions

Our principal contribution is to build a high performance secure two-party computation system that integrates state-of-the-art techniques for dealing with *malicious* adversaries efficiently. Although some of these techniques have been reported individually, we are not aware of any attempt to incorporate them all into one system, while ensuring that a security proof can still be written for that system. Even though some of the techniques are claimed to be compatible, it is not until everything is put together and someone has gone through all the details can a system as a whole be said to be provably secure.

System Framework We start by using Yao’s garbled circuit [34] protocol for securely computing functions in the presence of semi-honest adversaries, and shelat and Shen’s cut-and-choose-based transformation [33] that converts Yao’s garbled circuit protocol into one that

is secure against malicious adversaries.

We then modify the above to use Ishai et al.’s oblivious transfer extension [15] that has efficient amortized computation time for oblivious transfers secure against malicious adversaries, and Lindell and Pinkas’ random combination technique [23] that defends against selective failure attacks. We implement Kiraz’s randomized circuit technique [18] that guarantees that the generator gets either no output or an authentic output, i.e., the generator cannot be tricked into accepting arbitrary output.

Optimization Techniques For garbled circuit generation and evaluation, we incorporate Kolesnikov and Schneider’s free-XOR technique that minimizes the computation and communication cost for XOR gates in a circuit [20]. We also adopt Pinkas et al.’s garbled-row-reduction technique that reduces the communication cost for k -fan-in non-XOR gates by $1/2^k$ [31], which means at least a 25% communication saving in our system since we only have gates of 1-fan-in or 2-fan-in. Finally, we implement Goyal et al.’s technique for reducing communication as follows: during the cut-and-choose step, the check circuits are given to the evaluator by revealing the random seeds used to produce them rather than the check circuits themselves [11]. Combined with the 60%-40% check-evaluation ratio proposed by shelat and Shen [33], this technique provides a near 60% saving in communication. As far as we know, although these techniques exist individually, ours is the first system to incorporate all of these mutually-compatible state-of-the-art techniques.

Circuit-Level Parallelism The most important new technique that we use is to exploit the embarrassingly parallel nature of shelat and Shen’s protocol for achieving security in the malicious model. Exploiting this, however, requires careful engineering in order to achieve good performance while maintaining security. We parallelize all computation-intensive operations such as oblivious transfers or circuit construction by splitting the generator-evaluator pair into hundreds of slave pairs. Each of the pairs works on an independently generated copy of the circuit in a parallel but synchronized manner as synchronization is required for shelat and Shen’s protocol [33] to be secure.

Computation Complexity For the computation time of a secure computation, there are two main contributing factors: the input processing time I (due to oblivious transfers) and the circuit processing time C (due to garbled circuit construction and evaluation). In the semi-honest model, the system’s computation time is simply $I + C$. Security in the malicious model, however, requires several extra checks. In the first instantiation of our sys-

tem, through heavy use of circuit-level parallelism, our system needs roughly $I + 2C$ to compute hundreds of copies of the circuit. Thus when the circuit size is sufficiently larger than the input size, our system (secure in the malicious model) needs roughly twice as much computation time as that needed by the original Yao protocol (secure in the semi-honest model). This is a tremendous improvement over prior work [31,33] which needed 100x more time than the semi-honest Yao. In the second instantiation of our scheme, we are able to achieve $I + C$ computation time, albeit at the cost of moderately more communication overhead.

Large Circuits In the Fairplay system, a garbled circuit is fully constructed before being sent over a network for the other party to evaluate. This approach is particularly problematic when hundreds of copies of a garbled circuit are needed against malicious adversaries. Huang et al. [13] pointed out that keeping the whole garbled circuit in memory is unnecessary, and that instead, the generation and evaluation of garbled gates could be conducted in a “pipelined” manner. Consequently, not only do both parties spend less time idling, only a small number of garbled gates need to reside in memory at one time, even when dealing with large circuits. However, this pipelining idea does not work trivially with other optimization techniques for the following two reasons:

- The cut-and-choose technique requires the generator to finish constructing circuits before the coin flipping (which is used to determine check circuits and evaluation circuits), but the evaluator cannot start checking or evaluating before the coin flipping. A naive approach would ask the evaluator to hold the circuits and wait for the results of the coin flipping before she proceeds to do her jobs. When the circuit is of large size, keeping hundreds of copies of such a circuit in memory is undesirable.
- Similarly, the random seed checking technique [11] requires the generator to send the hash for each garbled circuit, and later on send the random seeds for check circuits so that the communication for check circuits is vastly reduced. Note that the hash for an evaluation circuit is given away before the garbled circuit itself. However, a hash is calculated only after the whole circuit is generated. So the generation-evaluation pipelining cannot be applied directly.

Our system, however, integrates this pipelining idea with the optimization techniques mentioned above, and is capable of handling circuits of billions of gates.

AES-NI Besides the improvements by the algorithmic means, we also incorporate the Intel Advanced En-

ryption Standard Instructions (AES-NI) in our system. While the encryption is previously suggested to be

$$\text{Enc}_{X,Y}(Z) = H(X||Y) \oplus Z$$

in the literature [6,20], where H is a 2-circular correlation robust function instantiated either with SHA-1 [13] or SHA-256 [31], we propose an alternative that

$$\text{Enc}_{X,Y}^k(Z) = \text{AES-256}_{X||Y}(k) \oplus Z,$$

where k is the index of the garbled gate. With the help of the latest instruction set, an AES-256 operation could take as little as 30% of the time for SHA-256. Since this operation is heavily used in circuit operations, with the help of AES-NI instructions, we are able to reduce the circuit computation time C by at least 20%.

Performance To get a sense of our improvements, we list the experimental results of the benchmark function—AES—from the most recent literature and our system. The latest reported system in the semi-honest model was built by Huang et al. [13] and needs 1.3 seconds (where $I = 1.1$ and $C = 0.2$) to complete a block of secure AES computation. The fastest known system in the malicious model was proposed by Nielson et al. [29] and has an amortized performance 1.6 seconds per block (or more precisely, $I = 79$ and $C = 6$ for 54 blocks). Our system provides security in the malicious model and needs 1.4 ($= I + 2C$, where $I = 1.0$ and $C = 0.2$) seconds per block. Note that both the prior systems require the full power of a random oracle, while ours requires a weaker cryptographic primitive, 2-circular correlation robust functions, which was recently shown to be sufficient to prove the security of the free-XOR technique. It should also be noted that our system benefits greatly from parallel computation, which was not tested for LEGO+.

Scalable Circuit Compiler One of the major bottlenecks that prevents large-scale secure computation is the need for a scalable compiler that generates a circuit description from a function written in a high-level programming language. Prior tools could barely handle circuits with 50,000 gates, requiring significant computational resources to compile such circuits. While this is just enough for an AES circuit, it is not enough for the large circuits that we evaluate in this paper.

We present a scalable boolean circuit compiler that can be used to generate circuits with billions of gates, with moderate hardware requirements. This compiler performs some simple but highly effective optimizations, and tends to favor XOR gates. The toolchain is flexible, allowing for different levels of optimizations and can be parameterized to use more memory or more CPU time when building circuits.

As a first sign that our compiler advances the state of the art, we observe that it automatically generates a smaller boolean circuit for the AES cipher than the hand-optimized circuit reported by Pinkas et al. [31]. AES plays an important role in secure computation, and oblivious AES evaluation can be used as a building block in cryptographic protocols. Not only is it one of the most popular building blocks in cryptography and real life security, it is often used as a benchmark in secure computation. With the textbook algorithm, the well-known Fairplay compiler can generate an AES circuit that has 15,316 non-XOR gates. Pinkas et al. were able to develop an optimized AES circuit that has 11,286 non-XOR gates. By applying an efficient S-box circuit [3] and using our compiler, we were able to construct an AES circuit that has 9,100 non-XOR gates. As a result, our AES circuit only needs 59% and 81% of the communication needed by the other two, respectively.

Most importantly, with our system and the scalable compiler, we are able to run experiments on circuits with sizes in the range of billions of gates. To the best of our knowledge, secure computation with such large circuits has never been run in the malicious model before. These circuits include 256-bit RSA (266,150,119 gates) and 4095x4095-bit edit distance (5,901,194,475). As the circuit size grows, resource management becomes crucial. A circuit of billions of gates can easily result in several GB of data stored in memory or sent over the network. Special care is required to handle these difficulties.

Paper Organization The organization of this paper is as follows. A variety of security decisions and optimization techniques will be covered in Section 3 and Section 4, respectively. Then, our system, including a compiler, will be introduced in Section 6 and Section 5. Finally, the experimental results are presented in Section 6 followed by the conclusion and future work in Section 7.

3 Techniques Regarding Security

The Yao protocol, while efficient, assumes honest behavior from both parties. To achieve security in the malicious model, it is necessary to enforce honest behavior. The *cut-and-choose* technique is one of the most efficient methods in the literature and is used in our system. Its main idea is for the generator to prepare multiple copies of the garbled circuit with independent randomness, and the evaluator picks a random fraction of the received circuits, whose randomness is then revealed. If any of the chosen circuits (called *check circuits*) is not consistent with the revealed randomness, the evaluator aborts; otherwise, she evaluates the remaining circuits (called *eval-*

uation circuits) and takes the majority of the outputs, one from each evaluation circuit, as the final output.

The intuition is that to pass the check, a malicious generator can only sneak in a few faulty circuits, and the influence of these (supposedly minority) faulty circuits will be eliminated by the majority operation at the end. On the other hand, if a malicious generator wants to manipulate the final output, she needs to construct faulty majority among evaluation circuits, but then the chance that none of the faulty circuits is checked will be negligible. So with the help of the cut-and-choose method, a malicious generator either constructs many faulty circuits and gets caught with high probability, or constructs merely a few and has no influence on the final output.

However, the cut-and-choose technique is not a cure-all. Several subtle attacks have been reported and would be a problem if not properly handled. These attacks include the *generator's input inconsistency attack*, the *selective failure attack*, and the *generator's output authenticity attack*, which are discussed in the following sections. Note that in this section, n denotes the input size and s denotes the number of copies of the circuit.

Generator's Input Consistency Recall that in the cut-and-choose step, multiple copies of a circuit are constructed and then evaluated. A malicious generator is therefore capable of providing altered inputs to different evaluation circuits. It has been shown that for some functions, there are simple ways for the generator to extract information about the evaluator's input [23]. For example, suppose both parties agree to compute the inner-product of their input, that is, $f([a_2, a_1, a_0], [b_2, b_1, b_0]) \mapsto a_2b_2 + a_1b_1 + a_0b_0$ where a_i and b_i is the generator's and evaluator's i -th input bit, respectively. Instead of providing $[a_2, a_1, a_0]$ to all evaluation circuits, the generator could send $[1, 0, 0]$, $[0, 1, 0]$, and $[0, 0, 1]$ to different copies of the evaluation circuits. After the majority operation from the cut-and-choose technique, the generator learns $\text{major}(b_2, b_1, b_0)$, the majority bit in the evaluator's input, which is not what the evaluator agreed to reveal in the first place.

There exist several approaches to deter this attack. Mohassel and Franklin [28] proposed the equality-checker that needs $O(ns^2)$ commitments to be computed and exchanged. Lindell and Pinkas [23] developed an approach that also requires $O(ns^2)$ commitments. Later, Lindell and Pinkas [24] proposed a pseudorandom synthesizer that relies on efficient zero-knowledge proofs for specific hardness assumptions and requires $O(ns)$ group operations. Shelat and Shen [33] suggested the use of malleable claw-free collections, which also uses $O(ns)$ group operations, but they showed that witness-indistinguishability suffices, which is more efficient than zero-knowledge proofs by a constant factor.

In our system, we incorporate the malleable claw-free collection approach because of its efficiency. Although the commitment-based approaches can be implemented using lightweight primitives such as collision-resistant hash functions, they incur high communication overhead for the extra complexity factor s , that is, the number of copies of the circuit. On the other hand, the group-based approach could be more computationally intensive, but this discrepancy is compensated again due to the parameter s .¹ Hence, with similar computation cost, group-based approaches enjoy lower communication overhead.

Selective Failure A more subtle attack is *selective failure* [19, 28]. A malicious generator could use inconsistent keys to construct the garbled gate and OT so that the evaluator’s input can be inferred from whether or not the protocol completes. In particular, a cheating generator could assign (K_0, K_1) to an input wire in the garbled circuit while using (K_0, K_1^*) instead in the corresponding OT, where $K_1 \neq K_1^*$. As a result, if the evaluator’s input is 0, she learns K_0 from OT and completes the evaluation without complaints; otherwise, she learns K_1^* and gets stuck during the evaluation. If the protocol expects the evaluator to share the result with the generator at the end, the generator learns whether or not the evaluation failed, and therefore, the evaluator’s input is leaked.

Lindell and Pinkas [23] proposed the random input replacement approach that involves replacing each of the evaluator’s input bits with an XOR of s additional input bits, so that whether the evaluator aborts due to a selective failure attack is almost independent (up to a bias of 2^{1-s}) of her actual input value. Both Kiraz [18] and shelat and Shen [33] suggested a solution that exploits committing OTs so that the generator commits to her input for the OT, and the correctness of the OTs can later be checked by opening the commitments during the cut-and-choose. Lindell and Pinkas [24] also proposed a solution to this problem using cut-and-choose OT, which combines the OT and the cut-and-choose steps into one protocol to avoid this attack.

Our system is based on the random input replacement approach due to its scalability. It is a fact that the committing OT or the cut-and-choose OT does not alter the circuit while the random input replacement approach inflates the circuit by $O(sn)$ additional gates. However, it has been shown that $\max(4n, 8s)$ additional gates suffice [31]. Moreover, both the committing OT and the cut-

and-choose OT require $O(ns)$ group operations, while the random input replacement approach needs only $O(s)$ group operations. Furthermore, we observe that the random input replacement approach is in fact compatible with the OT extension technique. Therefore, we were able to build our system which has the group operation complexity independent of the evaluator’s input size, and as a result, our system is particularly attractive when handling a circuit with a large evaluator input.

Generator’s Output Authenticity It is not uncommon that *both* the generator and evaluator receive outputs from a secure computation, that is, the goal function is $f(x, y) = (f_1, f_2)$, where the generator with input x gets output f_1 , and the evaluator with input y gets f_2 .² In this case, the security requires that *both* the input and output are hidden from each other. In the semi-honest setting, the straightforward solution is to let the generator choose a random number c as an extra input, convert $f(x, y) = (f_1, f_2)$ into a new function $f^*((x, c), y) = (\lambda, (f_1 \oplus c, f_2))$, run the original Yao protocol for f^* , and instruct the evaluator to pass the encrypted output $f_1 \oplus c$ back to the generator, who can then retrieve her real output f_1 with the secret input c chosen in the first place. However, the situation gets complicated when either of the participants could potentially be malicious. In particular, a malicious evaluator might claim an arbitrary value to be the generator’s output coming from the circuit evaluation. Note that the two-output protocols we consider are not *fair* since the evaluator always learns her own output and may refuse to send the generator’s output. However, they can satisfy the notion that the evaluator cannot trick the generator into accepting arbitrary output.

Many approaches have been proposed to ensure the generator’s output authenticity. Lindell and Pinkas [23] proposed a solution similar to the aforementioned solution in the semi-honest setting, where the goal function is modified to compute $f_1 \oplus c$ and its MAC so that the generator can verify the authenticity of her output. This approach incurs a cost of adding $O(n^2)$ gates to the circuit. Kiraz [18] presented a two-party computation protocol in which a zero knowledge proof of size $O(s)$ is conducted at the end. shelat and Shen [33] suggested a signature-based solution which, similar to Kiraz’s, adds n gates to the circuit, and requires a proof of size $O(s + n)$ at the end. However, they observed that witness-indistinguishable proofs are sufficient.

Lindell and Pinkas’ approach, albeit straightforward, might introduce greater communication overhead than the description function itself. We therefore employ the approach that takes the advantages of the remaining two solutions. In particular, we implement Kiraz’s approach

¹To give concrete numbers, with an Intel Core i5 processor and 4GB DDR3 memory, a SHA-256 operation (from OpenSSL) requires 1,746 cycles, while a group operation (160-bit elliptic curve from the PBC library with preprocessing) needs 322,332 cycles. It is worth mentioning that s is at least 256 in order to achieve security level 2^{-80} . The gap between a symmetric operation and an asymmetric one becomes even smaller when modern libraries such as RELIC are used instead of PBC.

²Here f_1 and f_2 are short for $f_1(x, y)$ and $f_2(x, y)$ for simplicity.

(smaller proof size), but only a witness-indistinguishable proof is performed (weaker security property).

4 Techniques Regarding Performance

Yao's garbled circuit technique has been studied for decades. It has drawn significant attention for its simplicity, constant round complexity, and computational efficiency (since circuit evaluation only requires fast symmetric operations). The fact that it incurs high communication overhead has provoked interest that has led to the development of fruitful results.

In this section, we will first briefly present the Yao garbled circuit, and then discuss the optimization techniques that greatly reduce the communication cost while maintaining the security. These techniques include free-XOR, garbled row reduction, random seed checking, and large circuit pre-processing. In addition to these original ideas, practical concerns involving large circuits and parallelization will be addressed.

4.1 Baseline Yao's Garbled Circuit

Given a circuit that consists of 2-fan-in boolean gates, the generator constructs a garbled version as follows: for each wire w , the generator picks a random permutation bit $\pi_w \in \{0, 1\}$ and two random keys $w_0, w_1 \in \{0, 1\}^{k-1}$. Let $W_0 = w_0 || \pi_w$ and $W_1 = w_1 || (\pi_w \oplus 1)$, which are associated with bit value 0 and 1 of wire w , respectively. Next, for gate $g \in \{f : \{0, 1\} \times \{0, 1\} \mapsto \{0, 1\}\}$ that has input wire x with (X_0, X_1, π_x) , input wire y with (Y_0, Y_1, π_y) , and output wire z with (Z_0, Z_1, π_z) , the garbled truth table for g has four entries:

$$GTT_g \begin{cases} \text{Enc}(X_0 \oplus \pi_x || Y_0 \oplus \pi_y, Z_{g(0 \oplus \pi_x, 0 \oplus \pi_y)}) \\ \text{Enc}(X_0 \oplus \pi_x || Y_1 \oplus \pi_y, Z_{g(0 \oplus \pi_x, 1 \oplus \pi_y)}) \\ \text{Enc}(X_1 \oplus \pi_x || Y_0 \oplus \pi_y, Z_{g(1 \oplus \pi_x, 0 \oplus \pi_y)}) \\ \text{Enc}(X_1 \oplus \pi_x || Y_1 \oplus \pi_y, Z_{g(1 \oplus \pi_x, 1 \oplus \pi_y)}) \end{cases}$$

$\text{Enc}(K, m)$ denotes the encryption of message m under key K . Here the encryption key is a concatenation of two labels, and each label is a random key concatenated with its permutation bit. Intuitively, π_x and π_y permute the entries in GTT_g so that for $i_x, i_y \in \{0, 1\}$, the $(2i_x + i_y)$ -th entry represents the input pair $(i_x \oplus \pi_x, i_y \oplus \pi_y)$ for gate g , in which case the label associated with the output value $g(i_x \oplus \pi_x, i_y \oplus \pi_y)$ could be retrieved. More specifically, to evaluate the garbled gate GTT_g , suppose $X || b_x$ and $Y || b_y$ are the retrieved labels for input wire x and wire y , respectively, the evaluator will use $X || b_x || Y || b_y$ to decrypt the $(2b_x + b_y)$ -th entry in GTT_g and retrieve label $Z || b_z$, which is then used to evaluate the gates at the next level. The introduction of the permutation bit helps to identify the correct entry in GTT_g , and thus, only one, rather than all, of the four entries will be decrypted.

4.2 Free-XOR

Kolesnikov and Schneider [20] proposed the free-XOR technique that aims for removing the communication cost and decreasing the computation cost for XOR gates.

The idea is that the generator first randomly picks a global key R , where $R = r || 1$ and $r \in \{0, 1\}^{k-1}$. This global key has to be hidden from the evaluator. Then for each wire w , instead of picking both W_0 and W_1 at random, only one is randomly chosen from $\{0, 1\}^k$, and the other is determined by $W_b = W_{1 \oplus b} \oplus R$. Note that π_w remains the rightmost bit of W_0 . For an XOR gate having input wire x with $(X_0, X_0 \oplus R, \pi_x)$, input wire y with $(Y_0, Y_0 \oplus R, \pi_y)$, and output wire z , the generator lets $Z_0 = X_0 \oplus Y_0$ and $Z_1 = Z_0 \oplus R$. Observe that

$$\begin{aligned} X_0 \oplus Y_1 &= X_1 \oplus Y_0 = X_0 \oplus Y_0 \oplus R = Z_0 \oplus R = Z_1 \\ X_1 \oplus Y_1 &= X_0 \oplus R \oplus Y_0 \oplus R = X_0 \oplus Y_0 = Z_0. \end{aligned}$$

This means that while evaluating an XOR gate, XORing the labels for the two input wires will directly retrieve the label for the output wire. So no garbled truth table is needed, and the cost of evaluating an XOR gate is reduced from a decryption operation to a bitwise XOR.

This technique is only secure when the encryption scheme satisfies certain security properties. The solution provided by the authors is

$$\text{Enc}(X || Y, K) = H(X || Y) \oplus Z,$$

where $H : \{0, 1\}^{2k} \mapsto \{0, 1\}^k$ is a random oracle. Recently, Choi et al. [6] have further shown that it is sufficient to instantiate $H(\cdot)$ with a weaker cryptographic primitive, *2-circular correlation robust functions*. Our system instantiates this primitive with $H(X || Y) = \text{SHA-256}(X || Y)$. However, when AES-NI instructions are available, our system instantiates it with $H^k(X || Y) = \text{AES-256}(X || Y, k)$, where k is the gate index.

4.3 Garbled Row Reduction

The GRR (Garbled Row Reduction) technique suggested by Pinkas et. al [31] is used to reduce the communication overhead for non-XOR gates. In particular, it reduces the size of the garbled truth table for 2-fan-in gates by 25%.

Recall that in the baseline Yao's garbled circuit, both the 0-key and 1-key for each wire are randomly chosen. After the free-XOR technique is integrated, the 0-key and 1-key for an XOR gate's output wire depend on input key and R , but the 0-key for a non-XOR gate's output wire is still free. The GRR technique is to make a smart choice for this degree of freedom, and thus, reduce one entry in the garbled truth table to be communicated over network.

In particular, the generator picks (Z_0, Z_1, π_z) by letting $Z_{g(0 \oplus \pi_x, 0 \oplus \pi_y)} = H(X_0 \oplus \pi_x || Y_0 \oplus \pi_y)$, that is, either Z_0 or Z_1

is assigned to the encryption mask for the 0-th entry of the GTT_g , and the other one is computed by the equation $Z_b = Z_{1 \oplus b} \oplus R$. Therefore, when the evaluator gets $(X_0 \pi_x, Y_0 \pi_y)$, both $X_0 \pi_x$ and $Y_0 \pi_y$ have rightmost bit 0, indicating that the 0-th entry needs to be decrypted. However, with GRR technique, she is able to retrieve $Z_{g(0 \oplus \pi_x, 0 \oplus \pi_y)}$ by running $H(\cdot)$ without inquiring GTT_g .

Pinkas et al. claimed that this technique is compatible with the free-XOR technique [31]. For rigorousness purposes, we carefully went through the details and came up with a security proof for our protocol that confirms this compatibility. The proof will be included in the full version of this paper.

4.4 Random Seed Checking

Recall that the cut-and-choose approach requires the generator to construct multiple copies of the garbled circuit, and more than half of these garbled circuits will be fully revealed, including the randomness used to construct the circuit. Goyal, Mohassel, and Smith [11] therefore pointed out an insight that the evaluator could examine the correctness of those check circuits by receiving a hash of the garbled circuit first, acquiring the random seed, and reconstructing the circuit and hash by herself.

This technique results in the communication overhead for check circuits independent of the circuit size. This technique has two phases that straddle the coin-flipping protocol. Before the coin flipping, the generator constructs multiple copies of the circuit as instructed by the cut-and-choose procedure. Then the generator sends to the evaluator the hash of each garbled circuit, rather than the circuit itself. After the coin flipping, when the evaluation circuits and the check circuits are determined, the generator sends to the evaluator the full description of the evaluation circuits and the random seed for the check circuits. The evaluator then computes the evaluation circuits and tests the check circuits by reconstructing the circuit and comparing its hash with the one received earlier. As a result, even for large circuits, the communication cost for each check circuit is simply a hash value plus the random seed. Our system provides that 60% of the garbled circuits are check circuits. As a result, this optimization significantly reduces communication overhead.

4.5 Working with Large Circuits

A circuit for a reasonably complicated operation can easily consist of hundreds of millions of gates. For example, a 1023-bit edit distance circuit has 309,454,016 gates. When circuits grow to such a size, the task of achieving high performance secure computation becomes challenging.

An $(I + 2C)$ -time solution Our solution for handling large circuits is based on Huang et. al's work [13], which is the only prior work capable of handling large circuits (of up to 1.2 billion non-XOR gates) in the semi-honest setting. Intuitively, the generator could work with the evaluator in a pipeline manner so that small chunks of gates are being processed at a time. The generator could start to work on the next chunk while the evaluator is still processing the current one. However, this technique does not work directly with the random seed checking technique described above in §4.4 because the generator has to finish circuit construction and hash calculation before the coin flipping, but the evaluator could start the evaluation only after the coin flipping. As a result, the generator needs a way to construct the circuit first, wait for the coin flipping, and send the evaluation circuits to the evaluator without keeping them in memory the whole time. We therefore propose that the generator constructs the evaluation circuits all over again after the coin flipping, with the same random seed used before and the same keys for input wires gotten from OT.

We stress that when fully parallelized, the second construction of an evaluation circuit does not incur overhead to the overall execution time. Although we suggest to construct an evaluation circuit twice, the fact is that according to the random seed checking, a check circuit is already being constructed twice—once before the coin flipping by the generator for hash computation and once after by the evaluator for correctness verification. As a result, when each generator-evaluator pair is working on a single copy of the garbled circuit, the constructing time for a evaluation circuit totally overlaps with that for a check circuit. We therefore achieve the overall computation time $I + 2C$ mentioned earlier, where the first C is for the generator to calculate the circuit hash, and the other C is either for the evaluator to reconstruct a check circuit or for both parties to work on an evaluation circuit in a pipeline manner as suggested by Huang et. al [13].

Achieving an $(I + C)$ -time solution We observe that there is a way to achieve $I + C$ computation time, which exactly matches the running time of Yao in the semi-honest setting. This idea, however, is not compatible with the random-seed technique, and therefore represents a trade-off between communication and computation. Recall that the generator has to finish circuit construction and hash evaluation before beginning coin flipping, whereas the evaluator can start evaluating only after receiving the coin flipping results. The idea is to run the coin flipping in the way that only the evaluator gets the result and does not reveal it to the generator until the circuit construction is completed. Since the generator is oblivious to the coin flipping result, she sends every garbled circuit to the evaluator, who could then either

evaluate or check the received circuit. In order for the evaluator to get the generator’s input keys for evaluation circuits and the random seed for the check circuits, they run an OT, where the evaluator uses the coin flipping result as input and the generator provides either the random seed (for the check circuit) or his input keys (for the evaluation circuit). After the generator completes circuit construction and reveals the circuit hash, the evaluator compares the hash with her own calculation, if the hashes match, she proceeds with the rest of the original protocol. Note that this approach comes at the cost of sacrificing the random seed checking technique and its 60% savings in communication.

Working Set Optimization Another problem encountered while dealing with large circuits is the *working set minimization problem*. Note that the *circuit value problem* is log-space complete for P. It is suspected that $L \neq P$, that is, there exist some circuits that can be evaluated in polynomial time but require more than logarithmic space. This open problem captures the difficulty of handling large circuits during both the construction and evaluation, where at any moment there is a set of wires, called the *working set*, that are available and will be referenced in the future. For some circuits, the working set is inherently super-logarithmic. A naive approach is to keep the most recent D wires in the working set, where D is the upper bound of the input-output distance of all gates. However, there may be wires which are used as inputs to gates throughout the entire circuit, and so this technique could easily result in adding almost the whole circuit to the working set, which is especially problematic when there are hundreds of copies of a circuit of billions of gates. While reordering the circuit or adding identity gates to minimize D would mitigate this problem, doing so while maintaining the topological order of the circuit is known to be an NP-complete problem, the *graph bandwidth problem* [9].

Our solution to this difficulty is to pre-process the circuit so that each gate comes with a usage count. Our system has a compiler that converts a program in high-level language into a boolean circuit. Since the compiler is already using global optimization in order to reduce the circuit size, it is easy for the global optimizer to analyze the circuit and calculate the usage count for each gate. With this information, it is easy for the generator and evaluator to decrement the counter for each gate whenever it is being referenced and to toss away the gate whenever its counter becomes zero. In other words, we keep track of merely useful information and heuristically minimize the size of the working set, which is small compared with the original circuit size as shown in Table 1.

	AES	Dot ₄ ⁶⁴	RSA-32	EDT-255
circuit size	49,912	460,018	1,750,787	15,540,196
wrk set size	323	711	235	2,829

Table 1: The size of the working set for various circuits

5 Boolean Circuit Compiler

Although the Fairplay circuit compiler can generate circuits, it requires a very large amount of computational resources to generate even relatively small circuits. Even on a machine with 48 gigabytes of RAM, Fairplay terminates with an out-of-memory error after spending 20 minutes attempting to compile an AES circuit. This makes Fairplay impractical for even relatively small circuits, and infeasible for some of the circuits tested in this project. One goal of this project was to have a general purpose system for secure computation, and so writing application specific programs to generate circuits, a technique used by others [13], was not an option.

To address this problem, we have implemented a new compiler that generates a more efficient output format than Fairplay, and which requires far lower computational resources to compile circuits. We were able to generate the AES circuit in only a few seconds on a typical desktop computer with only 8GB of RAM, and were able to generate and test much larger non-trivial circuits. We used the well-known *flex* and *bison* tools to generate our compiler, and implemented an optimizer as a separate tool. We also use the results from [31] to reduce 3 arity gates to 2 arity gates.

As a design decision, we created an imperative, untyped language with static scoping. We allow code, variables, and input/output statements to exist in the global scope; this allows very simple programs to be written without too much extra syntax. Functions may be declared, but may not be recursive. Variables do not need to be declared before being used in an unconditional assignment; variables assigned within a function’s body that are not declared in the global scope are considered to be local. Arrays are a language feature, but array indices must be constants or must be determined at compile time. If run-time determined indices are required for a function, a loop that selects the correct index may be used; this is necessary for oblivious evaluation. Variables may be arbitrarily concatenated, and bits or groups of bits may be selected from any variable and bits or ranges of bits may be assigned to; as with arrays, the index of a bit must be determined at compile time, or else a loop must be used. Note that loop variables may be used as such an index, since loops are always completely unrolled, and therefore the loop index can always be resolved at compile time. Additional language features are planned as future

work.

We use some techniques from the Fairplay compiler in our own compiler. In particular we use the single assignment algorithm from Fairplay, which is required to deal with assignments that occur inside of *if* statements. Otherwise, our compiler has several distinguishing characteristics that make it more resource efficient than Fairplay. The front end of our compiler attempts to generate circuits as quickly as possible, using as little memory as possible and performing only rudimentary optimizations before emitting its output. This can be done with very modest computational resources, and the intermediate output can easily be translated into a circuit for evaluation. The main optimizations are performed by the back end of the compiler, which identifies gates that can be removed without affecting the output of the circuit as a whole.

Unlike the Fairplay compiler, we avoided the use of hash tables in our compiler, using more memory-efficient storage. Our system can use one of three storage strategies: memory-mapped files, flat files without any mapping, and Berkeley DB. In our tests, we found that memory mapped files always resulted in the highest performance, but that Berkeley DB is only sometimes better than direct access without any mapping.

In the following sections, we describe these contributions in more detail, and provide experimental results.

5.1 Circuit Optimizations

The front-end of our compiler tends to generate inefficient circuits, with large numbers of unnecessary gates. As an example, for some operations the compiler generates large numbers of identity gates i.e. gates whose outputs follow one of their inputs. It is therefore essential to optimize the circuits emitted by the front end, particularly to meet our system's overall goal of practicality.

Our compiler uses several stages of optimization, most of which are global. As a first step, a local optimization removes redundant gates, i.e. gates that have the same truth table and input wires. This first step operates on a fixed-size chunk of the circuit, but we have found that there are diminishing improvements as the size of this window is increased. We also remove constant gates, identity gates, and inverters, which are generated by the compiler and which may be inadvertently generated during the optimization process. Finally, we remove gates that do not influence the output, which can be thought of as dead code elimination. The effectiveness of each optimization on different circuits is shown in Figure 1. The circuit that was least optimizable was the edit distance circuit, being reduced to only 82% of its size from the front end, whereas the RSA signing and the dot product circuits were the most optimizable, being reduced to

roughly half of the gates emitted by the front end.

Gate Removal The front-end of the compiler emits gates in topological order, and similar to Fairplay, our compiler assigns explicit identifiers to each emitted gate. To remove gates efficiently, we store a table that maps the identifiers of gates that were removed to the previously emitted gates, and for each gate that is scanned the inputs are rewritten according to this table. The table itself is then emitted, so that the identifiers of non-removed gates can be corrected. This mapping process can be done in linear time and space using an appropriate key-value store.

Removing Redundant Gates Some of the gates generated by the front end of our compiler have the same truth table and input wires as previously generated gates; such gates are redundant and can be removed. This removal process has the highest memory requirement of any other optimization step, since a description of every non-redundant gate must be stored. However, we found during our experiments that this optimization can be performed on discrete chunks of the circuit with results that are very close to performing the optimization on the full circuit, and that there are diminishing improvements in effectiveness as the size of the chunks is increased. Therefore, we perform this optimization using chunks, and can use hash tables to improve the speed of this step.

Removing Identity Gates and Inverters The front end may generate identity gates or inverters, which are not necessary. This may happen inadvertently, such as when a variable is incremented by a constant, or as part of the generation of a particular logic expression. While removing identity gates is straightforward, the removal of inverters requires more work, as gates which have inverted input wires must have their truth tables rewritten. There is a cascading effect in this process; the removal of some identity gates or inverters may transform later gates into identity gates or inverters. This step also removes gates with constant outputs, such as an XOR gate with two identical inputs. Constant propagation and folding occur as a side effect of this optimization.

Removing Unused Gates Finally, some gates in the circuit may not affect the output value at all. For this step, we scan the circuit backwards, and store a table of live gates; we then re-emit the live gates in the circuit and skip the dead gates. Immediately following this step, the circuit is prepared for the garbled circuit generator, which includes generating a usage count for each gate.

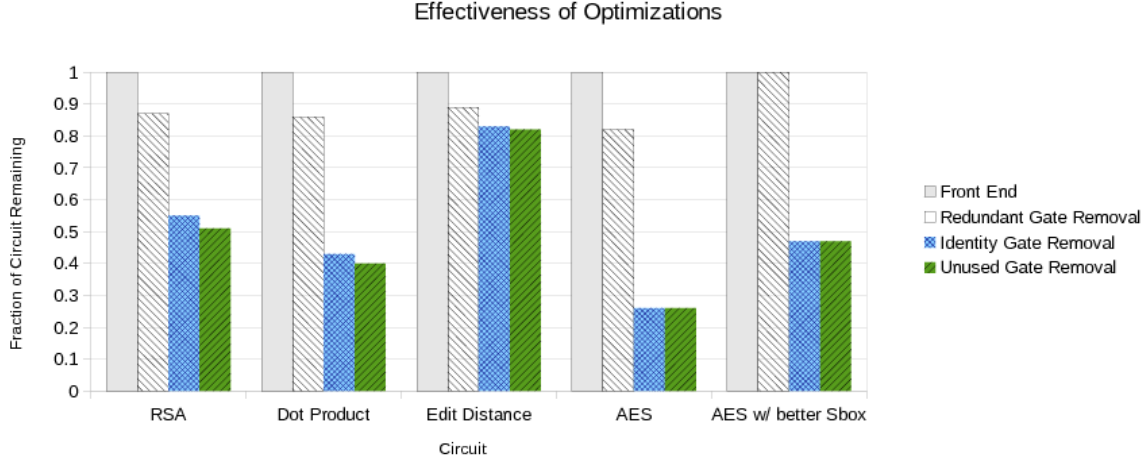


Figure 1: Average fraction of circuits remaining after each optimization is applied in sequence. We see that the *relative change* in circuit sizes after each optimization is dependent on the circuit itself, with some circuits being optimized more than others.

Circuit	DB (s)	mmap (s)	flat (s)
7200RPM Spinning Disk (ext4-fs)			
AES	$4.3 \pm 0.5\%$	$1.05 \pm 1\%$	$3.48 \pm 0.3\%$
RSA-32	$103 \pm 0.3\%$	$24.6 \pm 0.2\%$	$78.4 \pm 0.3\%$
Dot ₄ ⁶⁴	$32.56 \pm 0.1\%$	$7.1 \pm 0.3\%$	$28.37 \pm 0.1\%$
EDT-255	$975 \pm 0.1\%$	$240 \pm 1\%$	$700 \pm 0.9\%$
Solid-State Drive			
AES	$3.62 \pm 0.3\%$	$0.86 \pm 1\%$	$3.17 \pm 0.6\%$
RSA-32	$96.5 \pm 0.2\%$	$21.6 \pm 0.4\%$	$68.3 \pm 0.3\%$
Dot ₄ ⁶⁴	$30.5 \pm 0.5\%$	$6.27 \pm 1\%$	$25.9 \pm 0.2\%$
EDT-255	$907 \pm 0.1\%$	$200 \pm 0.4\%$	$590 \pm 1\%$
Amazon EC2			
AES	$5.56 \pm 4\%$	$1.12 \pm 0\%$	$7.11 \pm 0.3\%$
RSA-32	$208 \pm 0.4\%$	$45.7 \pm 3\%$	$240 \pm 0.1\%$
Dot ₄ ⁶⁴	$46.3 \pm 0.1\%$	$9.2 \pm 0.2\%$	$60.7 \pm 0.2\%$
EDT-255	$2500 \pm 1\%$	$405 \pm 0.2\%$	$2050 \pm 0.2\%$
Circuit Sizes			
AES	RSA-32	Dot ₄ ⁶⁴	EDT-255
49,912	1,750,787	460,018	15,540,196

Table 2: Compile times for different storage systems for small circuits (sizes include input gates), using different storage media. Results are averaged over 30 experiments, with 95% confidence intervals. On EC2, a high-memory quadruple extra large instance was used.

Key-Value Stores Unfortunately, even though our compiler is more resource efficient than Fairplay, it still requires space that is linear in the size of the circuit. For very large circuits, circuits with billions of gates or more, this may exceed the amount of RAM that is available. Our compiler can make use of a computer’s hard drive to store intermediate representations of circuits and information about how to remove gates from the circuit. We used memory-mapped I/O to reduce the impact this has on performance; however, our use of *mmap* and *truncate* is not portable, and so our system also supports using an unmapped file or Berkeley DB. Our tests revealed that, as expected, memory-mapped I/O achieves the highest performance, but that Berkeley DB is sometimes better than unmapped files on high-latency filesystems. A summary of the performance of each method on a variety of storage systems is shown in Table 2.

Using the hard drive in this manner, we were able to compile our largest circuits. The performance impact of writing to disk should not be understated; a several-billion-gate edit distance 4095x4095 circuit required more than 3 days to compile on an Amazon EC2 high-memory image, with 68 GB of RAM, one third of which was spent waiting on I/O. Note, however, that this is a one-time cost; a compiled circuit can be used in unlimited evaluations of a secure computation protocol.

5.2 Compiler Testing Methodology

We tested the performance of our compiler using five circuits. The first was AES, to compare our compiler with the Fairplay system. We also used AES with the compact S-Box description given by Boyar and Parnia [3], which results in a smaller AES circuit. We used an RSA

RSA Size	Circuit Size	Compile Time (s)	Gates/s	Edit-Dist Size	Circuit Size	Compile Time (s)	Gates/s
16	208,499	$2.6 \pm 7\%$	80,000	31x31	144,277	$1.70 \pm 0.7\%$	84,900
32	1,750,787	$21.6 \pm 0.4\%$	81,100	63x63	717,233	$8.56 \pm 0.7\%$	83,800
64	14,341,667	$189 \pm 0.3\%$	75,900	127x127	3,389,812	$41.7 \pm 0.5\%$	81,300
128	116,083,983	$1810 \pm 0.3\%$	64,100	255x255	15,540,196	$200 \pm 0.4\%$	77,700

Table 3: Time required to compile and optimize RSA and edit distance circuits on a workstation with an Intel Xeon 5506 CPU, 8GB of RAM and a 160GB SSD, using the textbook modular exponentiation algorithm. Note that the throughput for edit distance is higher even for comparably sized circuits; this is because the front end generates a more efficient circuit without any optimization. Compile times are averaged over 30 experiments, with 95% confidence intervals reported.

signing circuit with various toy key sizes, up to 128 bits, to test our compiler’s handling of large circuits; RSA circuits have cubic size complexity, allowing us to generate very large circuits with small inputs. We also used an edit distance circuit, which was the largest test case used by Huang et al. [13]; unlike the other test circuits, there is no multiplication routine in the inner loop of this function. Finally we used a dot product with error, a basic sampling function for the LWE problem, which is similar to RSA in creating large circuits, but also demonstrates our system’s ability to handle large input sizes.

After compiling these circuits, we tested the correctness by first performing a direct, offline evaluation of the circuit, and comparing the output to a non-circuit implementation. We then compared the output of an online evaluation to the offline evaluation. Additionally, for the AES circuit, we compared the output of the circuit generated by our compiler to the output of a circuit generated using Fairplay. We tested all three key-value stores on a variety of filesystems, including a fast SSD, a spinning disk, and an Amazon EC2 instance store, checking for correctness as described above in each case.

5.3 Summary of Compiler Performance

Our compiler is able to emit and optimize large circuits in relatively short periods of time, less than an hour for circuits with tens of millions of gates on an inexpensive workstation. In Figure 1 we summarize the effectiveness of the various optimization stages on different circuits; in circuits that involve multiplication in finite fields or modulo an integer, the identity gate removal step is the most important, removing more than half of the gates emitted by the front-end. The edit distance circuit is the best-case for our front end, as less than 1/5 of the gates that are emitted can be removed by the optimizer. The throughput of our compiler is dependent on the circuit being compiled, with circuits which are more efficiently generated by the front-end being compiled faster; in Table 3 we compare the generation of RSA circuits to edit distance circuits.

6 Experimental Results

In this section, we give a detailed description of our system, upon which we have implemented various real world secure computation applications. The experimental environment is the Ranger cluster in the Texas Advanced Computing Center. Ranger is a blade-based system, where each node is a SunBlade x6240 blade running a Linux kernel and has four AMD Opteron quad-core 64-bit processors, as an SMP unit. Each node in the Ranger system has 2.3 GHz core frequency and 32 GB of memory, and the point-to-point bandwidth is 1 GB/sec. Although Ranger is a high-end machine, we use only a small fraction of its power for our system, only 512 out of 62,976 cores. Note that we use the PBC (Pairing-Based Cryptography) library [25] to implement the underlying cryptographic protocols such as oblivious transfers, witness-indistinguishable proofs, and so forth. However, moving to more modern libraries such as RELIC [32] is likely to give even better results, especially to those circuits with large input and output size.

System Setup In our system, both the generator and the evaluator run an equal number of processes, including a root process and many slave processes. A root process is responsible for coordinating its own slave processes and the other root process, while the slave processes work together on repeated and independent tasks. There are three pieces of code in our system: the generator, the evaluator, and the IP exchanger. Both the generator’s and evaluator’s program are implemented with Message Passing Interface (MPI) library. The reason for the IP exchanger is that it is common to run jobs on a cluster with dynamic working node assignment. However, when the nodes are dynamically assigned, the generator running on one cluster and the evaluator running on another might have a hard time locating each other. Therefore, a fixed location IP exchanger helps the match-up process as described in Figure 2. Our system provides two modes—the user mode and the simulation mode. The former works as mentioned above, and the latter simply

spawns an even number of processes, half for the generator and the other half for the evaluator. The network match-up process is omitted in the latter mode to simplify the testing of this system.

To achieve a security level of 2^{-80} , meaning that a malicious player cannot successfully cheat with probability better than 2^{-80} , requires at least 250 copies of the garbled circuit [33]. For simplicity, we used 256 copies in our experiments, that is, security parameters $k = 80$ and $s = 256$. Each experiment was run 30 times (unless stated otherwise), and in the following sections we report the average runtime of our experiments.

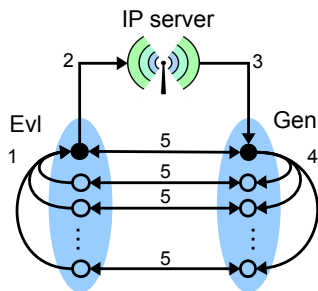


Figure 2: Both the generator and evaluator consist of a root process (solid dot) and a number of slave processes (hollow dots). The match-up works as follows: the slave evaluator processes send their IP’s to the root evaluator process (Step 1), who then forwards them to the IP exchanger (Step 2). Next, the root generator process comes to acquire these IP’s (Step 3) and dispatch them to its slaves (Step 4), who then proceed to pair up with one of the slave evaluator processes (Step 5) and start the main protocol. The arrows show the message flow.

Timing methodology When there is more than one process on each side, care must be taken in measuring the timings of the system. The timings reported in this section are the time required by the root process at each stage of the system. This was chosen because the root process will always be the *longest* running process, as it must wait for each slave process to run to completion. Moreover, in addition to doing all the work that the slaves do, the root processes also perform the input consistency check and the coin tossing protocol.

Impacts of the Performance Optimization Techniques

We have presented several performance optimization techniques in Section 4 with theoretical analyses, and here we demonstrate their empirical effectiveness in Table 4. As we have anticipated, the Random Seed Checking reduces the communication cost for the garbled circuits by 60%, and the Garbled Row Reduction further reduces by another 25%. In the RS and GRR columns,

the small deviation from the theoretical fraction 40% and 30%, respectively, is due to certain implementation needs. Our compiler is designed to reduce the number of non-XOR gates. In these four circuits, the ratio of non-XOR gates is less than 43%. So after further applying the Free-XOR technique, the final communication is less than 13% of that in the baseline approach.

	non-XOR (%)	Baseline (MB)	RS (%)	GRR (%)	FX (%)
AES	30.81	509	39.97	30.03	9.09
Dot ₄ ⁶⁴	29.55	4,707	39.86	29.91	8.88
RSA-32	34.44	17,928	39.84	29.88	10.29
EDT-255	41.36	159,129	39.84	29.87	12.36

Table 4: The impact of various optimization techniques: The Baseline shows the communication cost for 256 copies of the original Yao garbled circuit when $k = 80$; RS shows the remaining fraction after Random Seed technique is applied; GRR shows when Garbled Row Reduction is further applied; and FX shows when the previous two techniques and the Free-XOR are applied. (The communication costs here only include those in the generation and evaluation stages.)

Performance Gain by AES-NI On a machine with 2.53 GHz Intel Core i5 processor and 4GB 1067 MHz DDR3 memory, it takes 784 clock cycles to run a single SHA-256 (with OpenSSL 1.0.0g), while it needs only 225 cycles for AES-256 (with AES-NI). To measure the benefits of AES-NI, we use two instantiations to construct various circuits, listed in Table 5, and observe a consistent 20% saving in circuit construction.³

	size (gate)	AES-NI (sec)	SHA-256 (sec)	Ratio (%)
AES	49,912	0.12± 1%	0.15± 1%	78.04
Dot ₄ ⁶⁴	460,018	1.11±0.4%	1.41±0.5%	78.58
RSA-32	1,750,787	4.53±0.5%	5.9±0.8%	76.78
EDT-255	15,540,196	42.0±0.5%	57.6± 1%	72.92

Table 5: Circuit generation time (for a single copy) with different instantiations (AES-NI vs SHA-256) of the 2-circular correlation robust function.

AES We used AES as a benchmark to compare our compiler to the Fairplay compiler, and as a test circuit

³The reason that saving 500+ cycles does not lead to more improvements is that this encryption operation is merely one of the contributing factors to generating a garbled gate. Other factors, for example, include GNU hash_map table insertion (~1,200 cycles) and erase (~600 cycles).

for our system. We tested the full AES circuit, as specified in FIPS-197 [8]. In the semi-honest model, it is possible to reduce the number of gates in an AES circuit by computing the key schedule offline; e.g. this is one of the optimizations employed by Huang et al. [13]. In the malicious model, however, such an optimization is not possible; the party holding the key could attempt to reduce the security level of the cipher by computing a malicious key schedule. So in our experiments we compute the entire function, including the key schedule, online.

In this experiment, two parties collaboratively compute the function $f : (x, y) \mapsto (\perp, \text{AES}_x(y))$, i.e., the circuit generator holds the encryption key x , while the evaluator has the message y to be encrypted. At the end, the generator will not receive any output, whereas the evaluator will receive the ciphertext $\text{AES}_x(y)$.

Type	Fairplay	Ours-A	Pinkas et al.	Ours-B
non-XOR	15,316	15,300	11,286	9,100
XOR	35,084	34,228	22,594	21,628

Table 6: The components of the AES circuits from different sources. Ours-A comes from the textbook AES algorithm, and Ours-B uses an optimized S-box circuit from [3]. (Sizes do not include input or output wires)

First of all, we demonstrate the performance of our compiler in Table 6. We have shown in Section 5 that our compiler is capable of large circuit generation. We also found in our experiments that our compiler produces smaller AES circuit than Fairplay. Given the same high-level description of AES encryption (textbook AES), our compiler produces a circuit with a smaller gate count and even fewer non-XOR gates. When applying the compact S-Box description proposed by Boyar and Parelta [3] to the high-level description as input to our compiler, a smaller AES circuit than the hand-optimized one from Pinkas et al. is generated with less effort.

In Table 7, both the computational and communication costs for each main stage are listed under the traditional setting, where there is only one process on each side. These main stages include oblivious transfer, garbled circuit construction, the generator’s input consistency check, and the circuit evaluation. Each row includes both the computation and communication time used. Note that network conditions could vary from setting to setting. Our experiments run in a local area network, and the data can only give a rough idea on how fast the system could be in an ideal environment. However, the precise amount of data being exchanged is reported.

We notice in Table 7 that the evaluator spends an unreasonable amount of time on communication with respect to the amount of data to be transmitted in both the oblivious transfer and circuit construction stages.

		Gen (sec)		Eval (sec)	Comm (KB)
OT	comp	45.8±0.09%		34.0±0.2%	5,516
	comm	0.1± 1%		11.9±0.6%	
Gen.	comp	35.6± 0.5%		—	3
	comm	—		35.6±0.5%	
Inp. Chk	comp	—		1.75±0.2%	266
	comm	—		—	
Evl.	comp	14.9± 0.6%		32.4±0.4%	28,781
	comm	18.2± 1%		3.2±0.8%	
Total	comp	96.3± 0.3%		68.0±0.2%	34,566
	comm	18.3± 1%		50.8±0.4%	

Table 7: The 95% two-sided confidence intervals of the computation and communication time for each stage in the experiment $(x, y) \mapsto (\perp, \text{AES}_x(y))$.

This is because the evaluator spends that time waiting for the generator to finish computation-intensive tasks. The same reasoning explains why in the circuit evaluation stage the generator spends more time in communication than the evaluator. This waiting results from the fact that both parties need to run the protocol in a synchronized manner. A generator-evaluator pair cannot start next communication round while any other pair has not finished the current one. This synchronization is crucial since our protocol’s security is guaranteed only when each communication round is performed sequentially. While the parallelization of the program introduces high performance execution, it does not and should not change this essential property. A stronger notion of security such as universal security will be required if asynchronous communication is allowed. By using TCP sockets in “blocking” mode, we enforce this communication round synchronization.

Note that the low communication during the circuit construction stage is due to the random seed checking technique. Also, the fact that the generator spends more time in the evaluation stage than she traditionally does comes from the second construction for evaluation circuits. Recall that only the evaluation circuits need to be sent to the evaluator. Since only 40% of the garbled circuits (102 out of 256) are evaluation-circuits, the ratio of the generator’s computation time in the generation and evaluation stage is $35.63:14.92 \simeq 5:2$.

We were unfortunately unable to find a cluster of hundreds of nodes that all support AES-NI. Our experimental results, therefore, do not show the full potential of all the optimization techniques we have proposed. However, recall that for certain circuits the running time in the semi-honest setting is roughly half of that in the

node #	4		16		64		256	
	Gen	Evl	Gen	Evl	Gen	Evl	Gen	Evl
OT	12.56±0.1%	8.41±0.1%	4.06±0.1%	2.13±0.2%	1.96±0.1%	0.58±0.2%	0.64±0.1%	0.19±0.2%
Gen.	8.18±0.4%	–	1.92±0.7%	–	0.49±0.4%	–	0.14± 1%	–
Inp. Chk	–	0.42± 4%	–	0.10± 10%	–	–	–	–
Evl.	3.3± 4%	7.08± 1%	0.80± 10%	1.58± 4%	0.23± 17%	0.37± 7%	0.12±0.5%	0.05±0.6%
Inter-com	4± 5%	13.2±0.3%	0.93± 10%	4.08±0.8%	0.31± 20%	1.98± 1%	0.11± 40%	0.72±0.2%
Intra-com	0.17± 30%	0.23± 20%	0.18± 8%	0.25± 6%	0.45± 20%	0.48± 15%	0.34± 30%	0.34± 30%
Total time	28.3±0.3%	29.4±0.3%	7.90±0.5%	8.17±0.4%	3.45± 2%	3.44± 2%	1.4± 10%	1.3± 9%

Table 8: The average and error interval of the times (seconds) running AES circuit. The number of nodes represents the degree of parallelism on each side. “–” means that the time is smaller than 0.05 seconds. Inter-com refers to the communication between the two parties, and intra-com refers to communication between nodes for a single party.

malicious setting. We estimate a 20% improvement in the performance of garbled circuit generation when the AES-NI instruction set becomes ubiquitous, based on the preliminary results presented above in Table 5.

Table 8 shows that the Yao protocol really benefits from the circuit-level parallelization. Starting from Table 7, where each side only has one process, all the way to when each side has 256 processes, as the degree of parallelism is multiplied by four, the total time reduces into a quarter. Note that the communication costs between the generator and evaluator remain the same, as shown in Table 7. It may seem odd that the communication costs are *reduced* as the number of processes increase. The real interpretation of this data is that as the number of processes increases, the “waiting time” decreases.

Notice that as the number of processes increases, the ratio of the time the generator spends in the construction and evaluation stage decreases from 5:2 to 1:1. The reason is that the number of garbled circuit each process handles is getting smaller and smaller. Eventually, we reach the limit of the benefits that the circuit-level parallelism could possibly bring. In this case, each process is dealing with merely a single copy of the garbled circuit, and the time spent in both the generation and evaluation stages is the time to construct a garbled circuit.

To the best of our knowledge, completing an execution of secure AES in the malicious model within 1.4 seconds is the best result that has ever been reported. The next best result from Nielsen et al. [29] is 1.6 seconds, and it is an amortized result (85 seconds for 54 blocks of AES encryption in parallel) in the random oracle model. This is only a crude comparison, however; our experimental setup uses a cluster computer while Nielsen et al. used only two desktops. A better comparison would be possible given a parallel implementation of Nielsen et al.’s system, and we are interested in seeing how much of an improvement such an implementation could achieve.

Large Circuits In this experiment, we run the 4095-bit edit distance circuit, that is, $(x, y) \mapsto (\perp, \text{EDT}(x, y))$, where $x, y \in \{0, 1\}^{4095}$. In particular, we use the $I + C$ approach, where the computation time could be roughly a half of that of the $I + 2C$ approach with the price of not getting to use the random-seed technique. Recall that in the $I + C$ approach, the generator and the evaluator conduct the cut-and-choose in a way that the generator does not know the check circuits until she finishes transferring all the garbled circuits. Next, both the parties run the circuit generation and evaluation in a pipeline manner, where one party is generating and giving away garbled gates on one end, and the other party is evaluating and checking the received gates at the other end at the same time. The results are shown in Table 9.

	Gen (sec)	Eval (sec)	Comm (Byte)
OT	19.73±0.5% 1.1± 6%	5.26±0.4% 15.6±0.6%	1.7×10^8
Cut-& Choose	1.1±0.8% –	– 1.5± 2%	6.5×10^7
Gen./Evl.	24,400± 1% 4,900± 1%	14,600± 3% 14,700± 2%	1.8×10^{13}
Inp. Chk	0.6± 20% 0.4± 40%	– 0.60± 20%	8.5×10^6
Total	24,400± 1% 4,900± 1%	14,600± 3% 14,700± 2%	1.8×10^{13}

Table 9: The result of $(x, y) \mapsto (\perp, \text{EDT-4095}(x, y))$. Each party is comprised of 256 cores in a cluster. This table comes from 6 invocations of the system. Similarly, the upper row in each stage is the computation time, while the lower is the communication time.

This circuit generated by our compiler has 5.9 billion gates, and 2.4 billion of those are non-XOR. It is worth

mentioning that, without the random-seed technique, the communication cost shown in Table 9 can also be estimated by $256 \times 2.4 \times 10^9 \times 3 \times 10 = 1.8 \times 10^{13}$, since 256 copies of the garbled circuits need to be transferred, each copy has 2.4 billion non-free gates, each non-free gate has three entries, and each entry has $k = 80$ bits.

In addition to showing that our system is capable of handling the largest circuits ever reported, we also have shown a speed in the malicious setting that is comparable to those in the semi-honest setting. In particular, we were able to complete a single execution of 4095-bit edit distance circuit in less than 8.2 hours with a rate of 82,000 (non-XOR) gates per second. Note that Huang et al.'s system is the only one, to the best of our knowledge, that is capable of handling such large circuits [13]; they reported a rate of over 96,000 (non-XOR) gates per second for an edit-distance circuit in the semi-honest setting.

7 Conclusion

We have presented a general purpose secure two party computation system which offers security against malicious adversaries and which can efficiently evaluate circuits with hundreds of millions and even billions of gates on affordable hardware. Our compiler can generate large circuits using fewer computational resources than similar compilers, and offers improved flexibility to users of the system. Our evaluator can take advantage of parallel computing resources, which are becoming increasingly common and affordable. As future work, we plan further improvements to our compiler and language, as well as experiments on systems other than Ranger. The source code for this system can be downloaded from the authors' website (<http://crypto.cs.virginia.edu/>), along with example functions, including those describe in this paper.

8 Acknowledgements

We would like to thank Benny Pinkas, Thomas Schneider, Nigel Smart and Stephen Williams for providing us with a copy of their optimized AES circuit. We would also like to thank Gabriel Robins for his advice on techniques for minimizing circuits in VLSI systems. Supported by Defense Advanced Research Projects Agency (DARPA) and the Air Force Research Laboratory (AFRL) under contract FA8750-11-2-0211. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the Defense Advanced Research Projects Agency or the US government.

References

- [1] BENDLIN, R., DAMGÅRD, I., ORLANDI, C., AND ZAKARIAS, S. Semi-homomorphic Encryption and Multiparty Computation. In *EUROCRYPT'11* (2011), pp. 169–188.
- [2] BOGETOFT, P., CHRISTENSEN, D. L., DAMGÅRD, I., GEISLER, M., JAKOBSEN, T. P., KRØIGAARD, M., NIELSEN, J. D., NIELSEN, J. B., NIELSEN, K., PAGTER, J., SCHWARTZBACH, M. I., AND TOFT, T. Secure Multiparty Computation Goes Live. In *Financial Cryptography* (2009), pp. 325–343.
- [3] BOYAR, J., AND PERALTA, R. A New Combinational Logic Minimization Technique with Applications to Cryptology. In *LNCS*, vol. 6049. Springer Berlin / Heidelberg, 2010, pp. 178–189.
- [4] BRICKELL, J., AND SHMATIKOV, V. Privacy-preserving Graph Algorithms in the Semi-honest Model. In *ASIACRYPT'05* (2005).
- [5] CANETTI, R., LINDELL, Y., OSTROVSKY, R., AND SAHAI, A. Universally Composable Two-Party and Multi-Party Secure Computation, 2002. <http://eprint.iacr.org/2002/140>.
- [6] CHOI, S. G., KATZ, J., KUMARESAN, R., AND ZHOU, H.-S. On the Security of the “Free-XOR” Technique, 2011. <http://eprint.iacr.org/2011/510>.
- [7] DAMGARD, I., PASTRO, V., SMART, N., AND ZAKARIAS, S. Multiparty Computation from Somewhat Homomorphic Encryption, 2011. <http://eprint.iacr.org/2011/535>.
- [8] FIPS. *Advanced Encryption Standard (AES)*, 2001.
- [9] GAREY, M. R., GRAHAM, R. L., JOHNSON, D. S., AND KNUTH, D. E. Complexity Results for Bandwidth Minimization. *SIAM Journal on Applied Mathematics* 34, 3 (1978), pp. 477–495.
- [10] GENTRY, C., HALEVI, S., AND SMART, N. P. Homomorphic Evaluation of the AES Circuit, 2012. <http://eprint.iacr.org/2012/099>.
- [11] GOYAL, V., MOHASSEL, P., AND SMITH, A. Efficient Two-party and Multi-party Computation against Covert Adversaries. In *EUROCRYPT'08* (2008), Springer-Verlag, pp. 289–306.
- [12] HENECKA, W., KOGL, S., SADEGHI, A.-R., SCHNEIDER, T., AND WEHRENBURG, I. TASTY: Tool for Automating Secure Two-party computations. In *CCS'10* (2010).

- [13] HUANG, Y., EVANS, D., KATZ, J., AND MALKA, L. Faster Secure Two-Party Computation Using Garbled Circuits. In *USENIX Security* (2011).
- [14] HUANG, Y., MALKA, L., EVANS, D., AND KATZ, J. Efficient Privacy-Preserving Biometric Identification. In *NDSS'11* (2011).
- [15] ISHAI, Y., KILIAN, J., NISSIM, K., AND PETRANK, E. Extending Oblivious Transfers Efficiently. In *CRYPTO'03*, vol. 2729 of *LNCS*. Springer Berlin / Heidelberg, 2003, pp. 145–161.
- [16] ISHAI, Y., PRABHAKARAN, M., AND SAHAI, A. Founding Cryptography on Oblivious Transfer Efficiently. In *CRYPTO'08*, vol. 5157 of *LNCS*. Springer Berlin / Heidelberg, 2008, pp. 572–591.
- [17] JHA, S., KRUGER, L., AND SHMATIKOV, V. Towards Practical Privacy for Genomic Computation. In *IEEE Symposium on Security and Privacy* (2008).
- [18] KIRAZ, M. *Secure and Fair Two-Party Computation*. PhD thesis, Technische Universiteit Eindhoven, 2008.
- [19] KIRAZ, M., AND SCHOENMAKERS, B. A Protocol Issue for The Malicious Case of Yao's Garbled Circuit Construction. In *27th Symposium on Information Theory in the Benelux* (2006).
- [20] KOLESNIKOV, V., AND SCHNEIDER, T. Improved Garbled Circuit: Free XOR Gates and Applications. In *ALP'08* (2008), vol. 5126 of *LNCS*, pp. 486–498.
- [21] LINDELL, Y., OXMAN, E., AND PINKAS, B. The IPS Compiler: Optimizations, Variants and Concrete Efficiency. In *CRYPTO'11* (2011), pp. 259–276.
- [22] LINDELL, Y., AND PINKAS, B. Privacy Preserving Data Mining. *Journal of Cryptology* 15, 3 (2002).
- [23] LINDELL, Y., AND PINKAS, B. An Efficient Protocol for Secure Two-Party Computation in the Presence of Malicious Adversaries. In *EUROCRYPT'07* (2007).
- [24] LINDELL, Y., AND PINKAS, B. Secure Two-party Computation via Cut-and-choose Oblivious Transfer. In *TCC'11* (2011), Springer-Verlag, pp. 329–346.
- [25] LYNN, B. Pairing-Based Cryptography Library, 2006. <http://crypto.stanford.edu/abc/>.
- [26] MALKA, L. VMCrypt: modular software architecture for scalable secure computation. In *CCS'11* (2011), pp. 715–724.
- [27] MALKHI, D., NISAN, N., PINKAS, B., AND SELLA, Y. Fairplay: A Secure Two-Party Computation System. In *USENIX Security* (2004), vol. 13, pp. 287–302.
- [28] MOHASSEL, P., AND FRANKLIN, M. Efficiency Tradeoffs for Malicious Two-Party Computation. In *PKC'06* (2006).
- [29] NIELSEN, J. B., NORDHOLT, P. S., ORLANDI, C., AND BURRA, S. S. A New Approach to Practical Active-Secure Two-Party Computation, 2011. <http://eprint.iacr.org/2011/091>.
- [30] OSADCHY, M., PINKAS, B., JARROUS, A., AND MOSKOVICH, B. SCiFi: A System for Secure Face Identification. In *IEEE Symposium on Security and Privacy* (2010).
- [31] PINKAS, B., SCHNEIDER, T., SMART, N., AND WILLIAMS, S. Secure Two-Party Computation Is Practical. In *ASIACRYPT'09* (2009), vol. 5912 of *LNCS*, pp. 250–267.
- [32] RELIC, 2012. <http://code.google.com/p/relic-toolkit/>.
- [33] SHELAT, A., AND SHEN, C.-H. Two-output Secure Computation with Malicious Adversaries. In *EUROCRYPT'11* (2011), pp. 386–405.
- [34] YAO, A. Protocols for Secure Computations. In *FOCS'82* (1982), pp. 160–164.

The Knowledge Tightness of Parallel Zero-Knowledge

Kai-Min Chung^{*1}, Rafael Pass^{**1}, and Wei-Lung Dustin Tseng¹

Department of Computer Science, Cornell University, Ithaca, NY, USA.
{chung,rafael,wtdseng}@cs.cornell.edu

Abstract. We investigate the concrete security of black-box zero-knowledge protocols when composed in parallel. As our main result, we give essentially tight upper and lower bounds (up to logarithmic factors in the security parameter) on the following measure of security (closely related to knowledge tightness): the number of queries made by black-box simulators when zero-knowledge protocols are composed in parallel. As a function of the number of parallel sessions, k , and the round complexity of the protocol, m , the bound is roughly $k^{1/m}$.

We also construct a modular procedure to amplify simulator-query lower bounds (as above), to generic lower bounds in the black-box concurrent zero-knowledge setting. As a demonstration of our techniques, we give a self-contained proof of the $o(\log n / \log \log n)$ lower bound for the round complexity of black-box concurrent zero-knowledge protocols, first shown by Canetti, Kilian, Petrank and Rosen (STOC 2002). Additionally, we give a new lower bound regarding constant-round black-box concurrent zero-knowledge protocols: the running time of the black-box simulator must be at least $n^{\Omega(\log n)}$.

Keywords: Zero-Knowledge, Knowledge Tightness, Concrete Security, Concurrent Zero-Knowledge Lower Bounds.

1 Introduction

Zero-knowledge interactive proofs, introduced by Goldwasser, Micali and Rackoff [GMR89] are paradoxical constructions allowing one player (called the prover) to convince another player (called the verifier) of the validity of a mathematical statement $x \in L$, while providing no additional knowledge to the verifier. In addition to being an independent construct of interest, zero-knowledge have become a extremely useful tool in construction of numerous cryptographic protocols.

^{*} Chung is supported by a Simons Foundation Fellowship.

^{**} Pass is supported in part by a Alfred P. Sloan Fellowship, Microsoft New Faculty Fellowship, NSF CAREER Award CCF-0746990, AFOSR YIP Award FA9550-10-1-0093, and DARPA and AFRL under contract FA8750-11-2-0211. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the Defense Advanced Research Projects Agency or the US government.

A fundamental question regarding zero-knowledge protocols is whether their composition remains zero-knowledge. In theoretical constructions as well as in practice, a zero-knowledge protocol is sometimes composed in parallel (to amplify soundness or to improve efficiency, for example). It is well known that the definition of zero-knowledge (ZK) is not closed under parallel composition [GK96b]. On the other hand, we know numerous constructions of constant-round zero-knowledge protocols that are secure when composed in parallel [FS90,GK96a,Gol02]. As a result, the subject of ZK with respect to parallel composition is widely considered closed.

We turn our attention to another fundamental question regarding zero-knowledge: its knowledge tightness. In its original definition, the zero-knowledge property is formalized by requiring that the view of any probabilistic polynomial time (PPT) verifier V in an interaction with a prover can be “indistinguishably reconstructed” by a PPT simulator S that interacts with no one. Since whatever V “sees” in the interaction can be reconstructed by the simulator, the interaction does not yield any knowledge to V that V cannot already compute by itself. Because the simulator is allowed to be an arbitrary PPT machine, this traditional notion of ZK only guarantees that the *class* of PPT verifiers learn nothing.

To more concretely measure the knowledge gained by a particular verifier, Goldreich, Micali and Wigderson [GMW91] (see also [Gol01]) put forward the notion of *knowledge tightness*: informally, the “tightness” of a simulation is the ratio of the (expected) running-time of the simulator, divided by the (worst-case) running-time of the verifier. Thus, in a knowledge-tight ZK proof, the verifier is expected to gain no more knowledge than what it could have computed in time closely related to its *worst-case* running-time. In addition to theoretical interests, the knowledge tightness of a zero-knowledge protocol is a helpful aid for setting the security parameter in practice. It is easy to check that the original zero-knowledge protocols [GMR89,GMW91,Blu86] all enjoy constant knowledge tightness. The aforementioned protocols secure under parallel composition [FS90,GK96a,Gol02] also enjoy constant knowledge tightness when executed in isolation; however, when composed in parallel, the tightness of these protocols seem increase/loosen linearly (sometimes even quadratically) with respect to the number of parallel sessions (based on the currently known analysis of their simulators)!

Since we do want to execute zero-knowledge protocols in parallel (for instance in the application of secure multi-party computation), a natural question is to ask: how does the knowledge tightness of a protocol vary when we increase the number of parallel repetitions?

1.1 Our results

In this work we give essentially tight upper and lower bounds to the above question. Our results focus on *black-box* zero-knowledge and “simulator queries”, which we explain below.

Informally, a protocol is *black-box* zero-knowledge if there exists a *universal* simulator S , called the *black-box simulator*, such that S generates the view of

any adversarial verifier V^* if S is given black-box access to V^* . Essentially all known constructions of zero-knowledge (with the notable exception of [Bar01]) and all practical zero-knowledge protocols are black-box zero-knowledge. Given a black-box simulator S , we focus on bounding the number of black-box queries made by S to a given adversarial verifier V^* ; we refer to this as the *simulator-query* complexity. It is easy to see that the number of queries made by a black-box simulator is closely related to knowledge tightness; in fact, for the case of constant round protocols, they are asymptotically equivalent.

We state our main theorems below:

Theorem 1. *Let n be the security parameter. For any $m = m(n)$, there exists a $2m + 7$ -round black-box zero-knowledge argument Π for all of NP based on one-way functions, with perfect completeness and negligible soundness error, such that for any polynomially bounded $k = k(n)$, the parallel composition of k -copies of the protocol, Π^k , remains black-box zero-knowledge with simulator-query complexity $O(mk^{1/m} \log^2 n)$.*

It is easy to extend the above theorem to proofs assuming the existence of collision-resistant hash-functions. We complement Theorem 1 with a lower bound:

Theorem 2. *Let n be the security parameter, L be a language, and $m = m(n) \in O\left(\frac{\log n}{\log \log n}\right)$. Suppose Π is a $m(n)$ -round black-box zero-knowledge argument for L with perfect completeness and negligible soundness error, and suppose there exist a polynomially bounded $k(n) \geq n$ such that the parallel composition of k -copies of the protocol, Π^k , remains black-box zero-knowledge with simulator-query complexity $O(k^{1/m}/(\log^2 n))$. Then, $L \in \text{BPP}$.*

For protocols with sub-logarithmic number of rounds, Theorem 1 and 2 are tight up to logarithmic factors in the security parameter; essentially, the simulator-query complexity is asymptotically close to $k^{1/m}$ (in most cases, think of k as a low polynomial in n). We mention that one can achieve simulator-query complexity $O(m)$ (independent of k) when $m = \omega(\log n)$.

Briefly, our results show that the concrete security of constant-round black-box zero-knowledge protocols actually decays polynomially in the number of parallel sessions. Fortunately, this decay can be significantly slowed if we consider protocols with more rounds (even if we simply use a large constant m).

1.2 Related Works

While we are unaware of any past work that explicitly studies the knowledge tightness of parallelized zero-knowledge protocols, there are numerous related publications that focus on the composition of zero-knowledge protocols, or on the concrete security of zero-knowledge simulator. Dwork, Naor and Sahai [DNS04] introduces the notion of *concurrent zero-knowledge* protocols; these protocols must stay zero-knowledge even when composed arbitrarily (a strengthening over parallel composition). Micali and Pass [MP06] introduces the notion of *precision*; in a precise zero-knowledge protocol, the running time of the simulator should

be closely related to the running time of the adversarial verifier, on a view by view basis¹ (a strengthening over knowledge tightness).

Even with these stronger requirements, Pandey et. al. [PPS⁺08] is able to construct protocols that are simultaneously precise and (black-box) concurrent zero-knowledge. Note that our results are incomparable with the result of [PPS⁺08] for many reasons, one of which being that black-box concurrent zero-knowledge protocols require logarithmically many rounds [CKPR01], while our setting is mainly interesting for sub-logarithmic-round protocols. Interestingly, [PPS⁺08] actually gives a construction of a family of precise concurrent zero-knowledge protocols, with trade-offs between round-complexity and precision, much like our observed trade-off between round-complexity and knowledge tightness for the case of parallelized zero-knowledge.

1.3 Connection to Concurrent Zero-Knowledge

We also present a connection from simulator-query lower bounds for zero-knowledge, to round-complexity lower bounds for concurrent zero-knowledge (cZK). Due to lack of space we postpone the result on Concurrent Zero-knowledge to the full version. We briefly discuss the ideas as follows.

We start by describing the common framework for all known black-box zero-knowledge lower bounds (e.g., [KPR98, Ros00, CKPR01, BL02, Kat08, HRS09, PTW09]). Let Π be a protocol for a language L . To show that Π cannot be zero-knowledge unless the language L is trivial (i.e., $L \in \text{BPP}$), we start by constructing a decision procedure for L . Let S be the black-box zero-knowledge simulator of Π , and let V^* be some “hard to simulate” adversarial verifier, and consider the following decision procedure $\mathcal{D}$: on input x , $\mathcal{D}(x)$ accepts if and only if $S^{V^*}(x)$ generates an accepting view of $V^*(x)$. Usually, the completeness of $\mathcal{D}$ follows easily from the zero-knowledge property; to show that $\mathcal{D}$ is sound often requires more work. Our query-complexity lower bounds (Theorem 2) also follow the same framework. That is, we construct some adversarial verifier V_{para}^* that schedules multiple sessions in parallel, and show that for any zero-knowledge simulator S with appropriately bounded query-complexity, if $x \notin L$, then $S^{V_{\text{para}}^*}(x)$ cannot generate an accepting view of $V_{\text{para}}^*(x)$.

Inspired by the work of Canetti, Kilian, Petrank and Rosen [CKPR01], we next present a modular construction of a concurrent adversarial verifier V_{conc}^* whose purpose is to *amplify* query-complexity lower bounds of more basic verifiers. For example, consider V_{para}^* , an adversarial verifier that is restricted to parallel composition. Our modular construction would take V_{para}^* as input, and output an adversarial verifier $V_{\text{conc}}^* = V_{\text{conc}}^*(V_{\text{para}}^*)$ that, among other things, nests multiple incarnations of V_{para}^* in a way that takes full advantage of the concurrent scheduling. Under appropriate parameters, our analysis would conclude that for any zero-knowledge simulator S with *polynomially* bounded query-complexity,

¹ For example, to achieve precision 2, if the simulator S generates a view of V^* and the running time of V^* on that view is T , then the simulator S must have run in time $2T$.

if $x \notin L$, then $S^{V_{\text{conc}}^*}(x)$ cannot generate an accepting view of $V_{\text{conc}}^*(x)$ (recall again that this is the key step for most zero-knowledge lower bounds).

To demonstrate our framework, we re-prove the result of [CKPR01] — a $o(\log n / \log \log n)$ round-complexity lower bound for black-box concurrent zero-knowledge (the currently best known round-complexity lower bound); we believe the resulting analysis is quite clean. We also give a second lower bound concerning constant-round cZK protocols:

Theorem (Informal). *Let L be a non-trivial language, and let Π be a constant-round black-box concurrent zero-knowledge protocol with a potentially possibly super-polynomial time simulator. Then the simulator must run in time $n^{\Omega(\log n)}$.*

Incidentally, Pass and Venkitasubramaniam [PV08] do construct constant-round black-box concurrent zero-knowledge protocols for all of $\mathbf{NP}$ in the model where both the simulator and the adversarial verifier runs in quasi-polynomial time $n^{\text{poly}(\log n)}$.

We also find our modular framework satisfying on a philosophical level: it serves as an framework in which lower bounds for restricted compositions of zero-knowledge (in this example parallel composition) can be transformed into lower bounds for zero-knowledge in the fully concurrent setting. A similar and celebrated example occurs in the work of Goldreich [Gol02], where it is shown that constructions of zero-knowledge protocols secure under parallel composition directly leads to constructions of concurrent zero-knowledge protocols secure in the timing model.

2 Preliminaries

We use $\mathbb{N}$ to denote the natural numbers $\{0, 1, \dots\}$, $[n]$ to denote the set $\{1, \dots, n\}$, and $|x|$ to denote the length of a string $x \in \{0, 1\}^*$. By $\text{ngl}(n)$, we mean a function negligible in n (i.e., $1/n^{\omega(1)}$). We assume familiarity with indistinguishability.

Interactive Protocols. An interactive protocol Π is a pair of interactive Turing machines, (P, V) , where V is probabilistic polynomial time (PPT). P is called the prover, while V is called the verifier. $\langle P, V \rangle(x)$ denotes the random variable (over the randomness of P and V) representing V 's output at the end of the interaction on common input x . If additionally V receives auxiliary input z , we write $\langle P(x), V(x, z) \rangle$ to denote V 's output. We assume WLOG that Π starts with a verifier message and ends with a prover message, and say Π has k **rounds** if the prover and verifier each sends k messages alternately. A full or partial **transcript** of Π is a sequence of alternating verifier and prover messages, $(v_1, p_1, \dots)$, where v denotes verifier messages and p denotes prover messages.

We may compose an interactive proof in parallel. Let $\Pi^k = (P^k, V^k)$ be the **parallel composition** of k copies of Π ; that is, each prover and verifier message in Π^k is just concatenation of k independent copies of the corresponding message in Π . Upon completion, V^k accepts if and only if all k sessions are accepted by V . We note that an adversarial verifier may chose to abort in one session but not another.

Zero Knowledge Protocols In the setting of zero knowledge, we consider an adversarial verifier that attempts to “gain knowledge” by interacting with an honest prover. An m -session **concurrent adversarial verifier** V^* is a probabilistic polynomial time machine that, on common input x and auxiliary input z , interacts with $m(|x|)$ independent copies of P concurrently (called **sessions**); the traditional stand-alone adversarial verifier is simply a 1-session adversarial verifier. There are no restrictions on how V^* schedules the messages among the different sessions, and V^* may choose to abort some sessions but not others. Let $\text{View}_{V^*}^P(x, z)$ be the random variable that denotes the **view** of V^* in an interaction with P (this includes the random coins of V^* and the messages received by V^*).

A **black-box simulator** S is a probabilistic polynomial time machine that is given black-box access to V^* (written as S^{V^*}). Formally, S fixes the random coins r of V^* a priori, and S is allowed to specify a valid partial transcript $\tau = (v_1, p_1, \dots, p_i)$ of V_r^* , and query V_r^* for the next verifier message v_{i+1} . Here, τ is **valid** if it is consistent with V_r^* , i.e., each verifier message v_j in τ is what V_r^* would have responded given the previous prover messages $p_1, \dots, p_{j-1}$ and the fixed random tape r . Note that S is allowed to “rewind” V^* by querying V^* with different partial transcripts that shares a common prefix.

Intuitively, an interactive proof is zero-knowledge (ZK) if the view of any stand-alone (1-session) adversarial verifier V^* can be generated by a simulator. The formal definition follows.

Definition 3 (Black-Box Zero-Knowledge [GMR89,GO94]). *Let $\Pi = \langle P, V \rangle$ be an interactive proof (or argument) for a language L . Π is **black-box zero-knowledge** if there exists a black-box simulator S such that for every common input x , auxiliary input z and every (stand-alone) adversary V^* , $S^{V^*(x,z)}(x)$ runs in time polynomial in $|x|$, and the ensembles $\{\text{View}_{V^*}^P(x, z)\}_{x \in L, z \in \{0,1\}^*}$ and $\{S^{V^*(x,z)}(x)\}_{x \in L, z \in \{0,1\}^*}$ are computationally indistinguishable as a function of $|x|$.*

3 Construction

We define a zero-knowledge argument PARALLELZK in Section 3.1, and show that it satisfies Theorem 1 in Section 3.2.

3.1 The Protocol

Our ZK argument PARALLELZK (also used in [PV08,PTV10]) is a slight variant of the precise ZK protocol of [MP06], which in turn is a generalization of the Feige-Shamir protocol [FS89]. The protocol for language $L \in \text{NP}$ proceeds in three stages, given a security parameter n , a common input statement $x \in \{0,1\}^n$, and a round-parameter m :

Stage Init: The verifier picks two random strings $r_1, r_2 \in \{0,1\}^n$ and sends their images $c_1 = f(r_1)$, $c_2 = f(r_2)$ through a one-way function f to the

prover. The verifier then acts as the prover in m parallel instances of a 4-round witness indistinguishable and special sound proof of knowledge (WI and SS-POK) of the NP statement “ c_1 or c_2 is in the image set of f ” (a witness here would be a pre-image of c_1 or c_2). All but the last two messages of each SS-POK is exchanged in this stage; we denote their partial transcripts by $(\alpha_1, \alpha_2, \dots, \alpha_k)$.

Stage 1: m message exchanges occur in Stage 1. In the j^{th} iteration, the prover sends β_j , a random second last message of the j^{th} SS-POK, and the verifier replies with the last message γ_j of the proof. These m iterations are called *slots*. Slot i is *convincing* if the verifier produces an accepting proof (i.e., the transcript $(\alpha_i, \beta_i, \gamma_i)$ is accepting). If there is ever an *unconvincing* slot, the prover aborts the whole session.

Stage 2: The prover provides a 4-round witness indistinguishable proof of knowledge (WI-POK) of knowledge of the statement “ $x \in L$, or one of c_1 or c_2 is in the image set of f ”.

Completeness and soundness follows directly from the proof of Feige and Shamir [FS89]; in fact, the protocol is an instantiation of theirs. Intuitively, to cheat in the protocol a prover must “know” an inverse to c_1 or c_2 (because Stage 2 is an argument of knowledge), which requires the prover to invert the one-way function f (its is shown in [FS90] that Stage Init and Stage 1 of the protocol cannot aid the prover in inverting f). A formal description of protocol ParallelZK is shown in Figure 1.

Common Input: an instance x of a language L with witness relation R_L .

Auxiliary Input for Prover: a witness w , such that $(x, w) \in R_L(x)$.

Stage Init:

V uniformly chooses $r_1, r_2 \in \{0, 1\}^n$.

$V \rightarrow P$: $c_1 = f(r_1), c_2 = f(r_2)$.

$V \leftrightarrow P$: Exchange in parallel (interactively) all but the last two messages $\alpha_1, \dots, \alpha_k$ of k WI and SS-POKs on common input (c_1, c_2) with respect to the witness relation:

$$R_{L'}(c_1, c_2) = \{r : f(r) = c_1 \text{ or } c_2\}$$

Note that V acts as the prover in these SS-POK's.

Stage 1: For $j = 1$ to k , exchange the j^{th} “slot”

$P \rightarrow V$: The second last message β_j of the j^{th} SS-POK.

$V \rightarrow P$: The last message γ_j of the j^{th} SS-POK.

P aborts if $(\alpha_j, \beta_j, \gamma_j)$ is not a valid SS-POK.

Stage 2:

$P \leftrightarrow V$: a 4-round computational-WI proof of knowledge from P to V on common input (c_1, c_2, x) with respect to the witness relation:

$$R_{L' \vee L}(c_1, c_2, x) = \{(r, w) : r \in R_{L'}(c_1, c_2) \text{ or } w \in R_L(x)\}$$

Fig. 1. PARALLELZK: a ZK argument for NP with round parameter m .

Remark 4. We note that here we use multiple slots to improve the knowledge tightness of parallel zero knowledge, whereas previously, multiple slots was typically used to achieve concurrent zero knowledge and $\omega(\log n)$ slots were considered. In contrast, we show that in the context of parallel zero knowledge, using even constant number of slots improves the knowledge tightness significantly. Indeed, both our simulation technique and its analysis presented in the next section are new, where we rewind each slot to resolve all sessions in parallel (as opposed to previous works that focused on one session at a time).

3.2 The Simulator

To show that protocol $\Pi = \text{PARALLELZK}$ satisfies Theorem 1, given any polynomially bounded $k = k(n)$, we need to construct a black-box zero-knowledge simulator $S = S_k$ for protocol Π^k (PARALLELZK repeated k times in parallel). On a very high-level, our simulator follows that of Feige and Shamir [FS90]: after fixing the SS-POK prefixes in Stage Init, the simulator rewinds one of the “slots” in Stage 1 (the last two messages of the SS-POKs). If the verifier responds with two convincing slots, the simulator uses the special-soundness property to extract a “fake witness” r such that $f(r) = c_1$ or c_2 , and uses this fake witness to simulate Stage 2 of the protocol.

Given an adversarial verifier V^* (for protocol Π^k) and a common input $x \in \{0, 1\}^n$, the simulator $S^{V^*}(x)$ does the following:

1. The simulator S interacts with V^* , following the honest prover strategy, until the end of Stage 1. We call this the *reference simulation*.
2. The simulator S attempts to *resolve* all k parallel sessions in the reference simulation by extracting a fake witness r from the SS-POKs for each non-aborting session; aborted sessions are automatically considered resolved (and no fake witnesses are needed). To do so, S repeats the following step (called a rewinding pass) as many times as necessary, until all sessions are resolved.
3. **A rewinding pass.** For each slot i , the simulator rewinds the reference simulation back to the beginning of slot i , sends V^* a fresh random message β'_i , and receives a new reply γ'_i (of course this is done in parallel for all k sessions). Note that for each unresolved session j , S already knows an accepting transcript $(\alpha_i, \beta_i, \gamma_i)$ of SS-POK from the reference simulation. If session j does not abort during slot i in this rewinding pass, then S learns another accepting transcript $(\alpha_i, \beta'_i, \gamma'_i)$ of SS-POK. In this case, S can resolve the session j by extracting a fake witness using the special-sound property.
4. S completes the reference simulation using extracted fake witnesses to simulate the Stage 2 proof (only needed in each parallel session that did not abort). S outputs the view of V^* on the reference simulation and this completion.

For simplicity, we assume that for sessions that did not abort in the reference simulation, the extraction of fake witnesses always succeeds whenever S receives

an accepting slot in a rewinding pass (i.e., we assume that S never sends the same value for β twice). This assumption can be made without loss of generality by the following modifications of the simulation strategy.

- Let the simulator S performs at most 2^n rewinding passes. If there exist any unsolved sessions j after 2^n rewinding passes, S resolves the session by brute force, i.e., by directly inverting the one-way function f to obtain a fake witness of length n . This modification increases the running time (but not the number of queries) of S by at most a $\text{poly}(n)$ factor (multiplicatively), and makes sure that S makes at most $\text{poly}(2^n)$ queries to V^* .
- Let the final verifier challenge in the SS-POK have length $|\beta| = n^2$. In this case, the probability of S ever querying V^* with the same value of β twice is $\text{poly}(2^n) \cdot 2^{-n^2} = 2^{-\Omega(n^2)}$, definitely negligible in n .

We now show two lemmas regarding S that together shows that PARALLELZK is zero-knowledge when composed in parallel.

Lemma 5. *S runs in expected polynomial time, and makes $O(mk^{1/m} \log^2 n)$ queries in expectation.*

Lemma 6. *On common input $x \in L$, the output of S is indistinguishable from the real view of V^* .*

We defer the proof of Lemma 5 to the next section, where we bound the *expected number of rewinding passes* before S extracts all necessary fake witnesses. We give a sketch of Lemma 6 now.

Proof (Proof Sketch). The output of S up to the end of Stage 1 (i.e., the reference simulation) is identical to the view of V^* , because S follows the honest prover strategy. The output of S in Stage 2 of the protocol is computationally indistinguishable from the view of V^* because the Stage 2 proof is witness indistinguishable. Formally, this can be shown with a hybrid argument where we incrementally exchange each of the k parallel Stage 2 proofs from using “fake witnesses” r such that $f(r) = c_1$ or c_2 (the simulator strategy), to a real witnesses w for $x \in L$ (the honest prover strategy).

3.3 Proof of Lemma 5

In this section, we prove Lemma 5 by bounding the expected number of rewinding passes in an execution of S . Let R be a random variable that denotes the number of rewinding passes. We will show that:

$$\mathbb{E}[R] = \mathbb{E}[\# \text{ rewinding passes }] \leq O(k^{1/m} \cdot \log^2 n).$$

This then implies Lemma 5 because outside of rewinding passes, $S^{V^*}(x)$ makes only $O(m)$ queries to V^* and runs in polynomial time.

Before presenting our analysis for the general case of m slots, we revisit the classical analysis for the case of single slot for intuition.

The case of single slot. The analysis is very simple. For every $j \in [k]$, let R_j denote the number of rewinding passes to resolve session j , and let p_j be the probability that session j does not abort during the single slot. Recall that session j is resolved if it aborts in the reference simulation, and otherwise, the simulator needs to rewind the slot several times until session j does not aborts again. Hence, the expected number of rewinding passes to resolve session j is

$$\mathbb{E}[R_j] = (1 - p_j) \cdot 0 + p_j \cdot \frac{1}{p_j} = 1.$$

By linearity of expectation, the expected number of rewinding passes is

$$\mathbb{E}[R] = \sum_j \mathbb{E}[R_j] = k \leq O(k \cdot \log^2 n).$$

We note that the above simple analysis is tight. Consider the case where during the slot, each session aborts independently with probability $(1 - 1/k)$. It is not hard to see that in this case, with constant probability, at least one session does not abort during the slot, and the simulator needs to rewind k times in expectation to resolve the survival session. Therefore, the expected number of rewinding passes is $\Omega(k)$.

In fact, it is instructive to note that the following natural generalization of the above example is essentially the worse-case example for the general case of m slots: during each slot $i \in [m]$, each survival session j aborts independently with probability $(1 - k^{-1/m})$. In this case, each session does not abort during the m slots with probability $(k^{-1/m})^m = 1/k$, and hence with constant probability, at least one session survives after m slots. Resolving the survival session requires $k^{1/m}/m$ rewinding passes in expectation, and hence the expected number of rewinding passes is $\Omega(k^{1/m}/m)$.

We note that although in the above example, each session aborts during each slot independently, in general, the aborting probability of each session at each slot can depends arbitrarily on the history and correlated arbitrarily.

The general case of m slots. To analyze the expected number of rewinding passes, we define the following $[0, 1]$ -valued random variables based on the reference simulation generated in Step 1. Let h_i denote the partial transcript of the reference simulation before slot i . For every slot $i \in [m]$ and session $j \in [k]$, we define random variable $p_{i,j}$ as follows.

- If session j is already aborted at the end of slot i , then we define $p_{i,j} \triangleq 1$.
- Otherwise, we define $p_{i,j}$ to be the conditional probability

$$p_{i,j} \triangleq \Pr[\text{session } j \text{ does not abort during slot } i \mid h_i].$$

For intuition, $p_{i,j}$ is essentially the probability that S can resolve session j by rewinding slot i . Now consider the *best slot* for each session — the slot with the highest $p_{i,j}$ value (this is the slot that S wants to rewind). We record this value as

$$p_j^* = \max_i p_{i,j}$$

Note that for a session j that aborts in the reference simulation, we have $p_j^* = 1$, indicating that sessions j is already resolved and matching the above intuition. Finally, the number of rewinding passes depends heavily on the *worst session* — the session with the worst p_j^* value (the “worst best slot”). We record this value as the *critical probability*:

$$p^* = \min_j p_j^*.$$

To see how the critical probability p^* plays an important role in the expected number of rewinding passes, note that on one hand, S needs roughly $1/p^*$ rewinding passes to resolve the worse-case session; on the other hand, the chance of having a reference simulation with small critical probability (say, $p^* \leq p$) is rare (at most p^m). Therefore, to upper bound $\mathbb{E}[R]$, we define the following events, which partition the probability space according to the critical probability. For every $t \in \mathbb{N}$, let

$$\alpha_t \stackrel{\text{def}}{=} \left(\frac{1}{2^t \cdot k^{1/m}} \right)$$

- Let A_0 be the event that $p^* \geq \alpha_0 = k^{-1/m}$, and for every $t \in \mathbb{N}$, let A_t be the event that

$$\alpha_t \leq p_j^* < \alpha_{t-1}.$$

Similarly for every session $j \in [k]$,

- Let $A_{0,j}$ be the event that $p_j^* \geq \alpha_0 = k^{-1/m}$, and for every $t \in \mathbb{N}$, let $A_{t,j}$ be the event that

$$\alpha_t \leq p_j^* < \alpha_{t-1}.$$

We can now express the expectation of the number of rewinding passes as follows.

$$\begin{aligned} \mathbb{E}[R] &= \sum_{t \geq 0} \Pr[A_t] \cdot \mathbb{E}[R \mid A_t] \\ &\leq \Pr[A_0] \cdot \mathbb{E}[R \mid A_0] + \sum_{t \geq 1} \left(\sum_{j=1}^k \Pr[A_{t,j}] \right) \cdot \mathbb{E}[R \mid A_t], \end{aligned}$$

where the last inequality follows by $A_t \subseteq \cup_j A_{t,j}$ (which follows from definition). We proceed to bound each term. For A_0 , we use trivial bound $\Pr[A_0] \leq 1$. For general $t \geq 1$ and every $j \in [k]$, we first observe that when $A_{t,j}$ happens, session j does not abort all of its m slots in the reference simulation (since otherwise, $p_j^* = 1$). This happened despite the fact that each slot i in session j in the reference simulation could have only survived (not aborted) with probability $p_{i,j} \leq \alpha_{t-1}$. Thus,

$$\Pr[A_{t,j}] \leq \alpha_{t-1}^m = \left(\frac{1}{2^{t-1} \cdot k^{1/m}} \right)^m = \frac{1}{2^{m(t-1)} \cdot k},$$

and,

$$\sum_{j=1}^k \Pr[A_{t,j}] \leq k \cdot \frac{1}{2^{m(t-1)} \cdot k} = \frac{1}{2^{m(t-1)}}.$$

It remains to bound $\mathbb{E}[R \mid A_t]$, which is given in the follow lemma.

Lemma 7. *For every $t \geq 0$, we have*

$$\mathbb{E}[R \mid A_t] \leq O\left(2^t \cdot k^{1/m} \cdot \log^2 n\right).$$

We apply Lemma 7 to upper bound $\mathbb{E}[R]$ first.

$$\begin{aligned} \mathbb{E}[R] &\leq \mathbb{E}[R \mid A_0] + \sum_{t \geq 1} \frac{1}{2^{m(t-1)}} \cdot \mathbb{E}[R \mid A_t] \\ &\leq O\left(k^{1/m} \cdot \log^2 n\right) + \sum_{t \geq 1} \frac{2^t}{2^{m(t-1)}} \cdot O\left(k^{1/m} \cdot \log^2 n\right) \\ &\leq O\left(k^{1/m} \cdot \log^2 n\right). \end{aligned}$$

This completes the proof of Lemma 5.

Proof (Proof of Lemma 7). The event A_t means that in the reference simulation, for every non-aborting session j , there exists a useful slot $i \in [m]$ such that

$$\Pr[\text{session } j \text{ is not aborted after slot } i \mid h_i] = p_{i,j} \geq \alpha_t.$$

Therefore, in each rewinding pass, the simulator S may learn an (additional) accepting transcript of SS-POK in session j with probability at least α_t , allowing it to extract a fake witness.

Fix a non-aborting session j , and define

$$q = \left(\frac{10 \cdot \log^2 n}{\alpha_t}\right) = O\left(2^t \cdot k^{1/m} \cdot \log^2 n\right),$$

Because the rewinding passes are independent, we have

$$\Pr[\text{session } j \text{ is resolved after } q \text{ rewinding passes}] = 1 - (1 - \alpha_t)^q \geq 1 - \text{ngl}(n).$$

Since there are at most k survival sessions, by the union bound,

$$\Pr[\text{all sessions are resolved after } q \text{ rewinding passes}] \geq 1 - \text{ngl}(n).$$

In other words, every q rewinding passes can solve all the sessions with probability at least $1 - \text{ngl}(n)$. It follows that

$$\begin{aligned} \mathbb{E}[R \mid A_t] &\leq (1 - \text{ngl}(n)) \cdot q + \text{ngl}(n) (1 - \text{ngl}(n)) \cdot 2q + \text{ngl}(n)^2 (1 - \text{ngl}(n)) \cdot 3q + \dots \\ &\leq O(q) = O\left(2^t \cdot k^{1/m} \cdot \log^2 n\right). \end{aligned}$$

4 Lower Bound

The proof of Theorem 2 follows a well-known framework (e.g., [GK96b, CKPR01]). Let S be a black-box zero-knowledge simulator for $\Pi^k = (P^k, V^k)$ that makes less than $q = O(k^{1/m}/\log^2 n)$ queries, and let V^{k*} be a particular adversarial verifier to be specified later. We define $\mathcal{D}$, a BPP decision procedure for L by combining S and V^{k*} : on input instance x , $\mathcal{D}(x)$ accepts if and only if $S^{V^{k*}}(x)$ outputs an accepting view of V^{k*} (i.e., all k sessions of V^{k*} accept). Using the zero-knowledge property, it is easy to show (see for example [GK96b]) that if the modified protocol $\Pi^{k*} = (P^k, V^{k*})$ is complete for L (based on our choice of V^{k*}), then $\mathcal{D}$ is complete for L as well. The main effort of the proof is to show that $\mathcal{D}$ is sound; this relies both on the choice of V^{k*} and the fact that S makes less than q queries to V^{k*} . We discuss our choice of V^{k*} in Section 4.1, and analyze the soundness of $\mathcal{D}$ in Section 4.2.

4.1 The Random Termination Verifier V^{k*}

In this section, we define a verifier V^{k*} for the parallelized protocol with two goals in mind: the protocol $\Pi^{k*} = (P^k, V^{k*})$ should be complete (so that $\mathcal{D}$ is complete), and V^{k*} should be sound against any rewinding simulator S that makes less than q queries to V^{k*} (so that $\mathcal{D}$ is sound).

Just as [CKPR01], we define V^{k*} to follow the honest verifier strategy V^k with one extra property: random termination.² Whenever the prover P^k or the rewinding simulator S makes a query to V^{k*} , V^{k*} determines, with independent and fresh randomness,³ whether or not to terminate immediately and *accept* with probability $\rho \in [0, 1]$, a parameter to be specified later; this is done *independently* for each of the k parallel sessions (i.e., one session may be terminated while other sessions continue). Due to this independence between parallel sessions, we often treat V^{k*} as k machines, $(V_1^*, \dots, V_k^*)$, each responsible for making the decision to terminate and generating the verifier messages for one session. Note that the fresh randomness is only used to decide whether to terminate or not; V^{k*} generates protocol messages using its default random tape that is kept the same between rewinds (as expected by following the honest verifier strategy).

Clearly, $\Pi^{k*} = (P^k, V^{k*})$ is still complete. It remains to show that V^{k*} is “sound” against the rewinding S ; that is, on input $x \notin L$, $S^{V^{k*}}$ is unlikely to

² The term “random termination” was first used by Haitner [Hai09], but the random termination verifier we considered already appeared in the earlier work of [CKPR01].

³ We use a well-known technique (see for example [GK96b, CKPR01]) to generate fresh independent randomness on the fly for each query from the simulator S , despite the fact that S may rewind V^{k*} between queries and force V^{k*} to use the same random tape. Let $\mathcal{H}$ be a family of q -wise independent hash-functions, and let V^{k*} sample one hash-function $h \leftarrow \mathcal{H}$ in the very beginning. Then whenever V^{k*} receives a query (from P^k or S), V^{k*} applies h to the current protocol transcript (the sequence of messages exchanged in the protocol so far) and use the output as a fresh random tape. Since S makes at most q queries to V^{k*} , the output distribution of the hash-function is truly uniformly random.

generate an accepting transcript of V^{k*} . From now on we drop the common input $x \notin L$. Intuitively, by randomly terminating, V^{k*} can better protect its randomness against S 's rewinds (when V^{k*} terminates, S learns nothing about V^{k*} 's fixed random tape), thus ensuring soundness. To make this intuition more concrete, suppose for example that S made q queries $\tau_1, \dots, \tau_q$ to V^{k*} , and without loss of generality outputs the view of V^{k*} on a subset of size m of those queries⁴, $T = \{\tau_{i_1}, \dots, \tau_{i_m}\}$. Further suppose that there exists a parallel session $j \in [k]$ such that V^{k*} does not terminate on the queries in T , but terminates on all remaining queries. Then intuitively, S 's rewinding does not help S convince V^{k*} in session j , and the soundness of the original protocol Π should imply that V^{k*} rejects with overwhelming probability in session j (and therefore rejects overall).

The core of our proof is to show that, with high probability, for every subset of size m of queries $T = \{\tau_{i_1}, \dots, \tau_{i_m}\}$ made by S , there exists a session $j \in [k]$ with *overwhelming probability* such that rewinds are “not helpful” for session j with respect to T in the above manner. We make this possible by setting the termination probability to $\rho = (1 - 1/q)$.

We now state the formal lemmas. Let n be the security parameter and L be a language. Suppose there exists a $m(n) \in O\left(\frac{\log n}{\log \log n}\right)$ -round argument $\Pi = (P, V)$ for L with perfect completeness and negligible soundness error. For any polynomially bounded $k(n) \geq n$, let S be a black-box zero-knowledge simulator of the parallelized protocol $\Pi^k = (P^k, V^k)$ that makes at most

$$q = k^{1/m} / (\log^2 n)$$

queries, and let V^{k*} be a random termination verifier of the parallelized protocol with termination probability

$$\rho = \left(1 - \frac{1}{q}\right) = \left(1 - \frac{1}{k^{1/m}} \cdot (\log^2 n)\right).$$

(These parameters passes the following sanity checks: q is polynomially bounded and $q \geq m$ — the simulator queries V^{k*} at least once for each round of the protocol. It is also useful later to know that $\binom{q}{m} \leq q^m \leq k$.) Then:

Lemma 8. *On input $x \in L$, $\mathcal{D}(x)$ accepts with probability 1, i.e., $S^{V^{k*}}(x)$ outputs an accepting view of V^{k*} with probability 1.*

Lemma 9. *On input $x \notin L$, the probability that $S^{V^{k*}}(x)$ generates an accepting view of V^{k*} is negligible, i.e., $\mathcal{D}$ has negligible soundness error.*

We sketch the proof of Lemma 8 now, and give the proof of Lemma 9 in the next section.

⁴ Without loss of generality, we may assume that before S outputs a view of V^{k*} , S first queries V^{k*} with the messages in the view (if S hasn't already). This may increase the number of queries by m , and thus weaken the resulting lower bound from q to $q - m$. Nevertheless, this does not change our lower bound since $q = \omega(m)$ in Theorem 2.

Proof (Proof Sketch). Using the zero-knowledge property, the output of S is indistinguishable from the view of V^{k*} in an execution with P^k . Therefore it is enough to show that $\langle P^k, V^{k*} \rangle(x)$ accepts with probability 1. In each parallel session $j \in [k]$, V_j^* accepts by definition if it decides to terminate in some protocol round. Otherwise, V_j^* is identical to V and would still accept with probability 1 because the original protocol $\Pi = (P, V)$ has perfect completeness.

4.2 Soundness of $\mathcal{D}$

Proof (Proof of Lemma 9). We prove Lemma 9 with a reduction. Suppose for the sake of contradiction that S convinces V^{k*} on some input $x \notin L$ with probability more than $1/p(n)$ for some polynomial p . Using S , we construct a cheating prover P^* for the original protocol $\Pi = (P, V)$ that convinces V with non-negligible probability.

Before we start, assume without loss of generality that S makes exactly q queries, and that before S outputs a view of V^{k*} , S would first query V^{k*} on all previous messages in the view. For technical convenience, we let V^{k*} make a fresh decision to terminate for each query and each session, *even if V^{k*} has already terminated previously in the same session*. I.e., regardless of history or message content, for each query and each parallel session, V^{k*} always terminates independently with probability ρ .

Our P^* is a natural extension of the classic reduction of [GK96b] — P^* guesses a session $j_0 \in [k]$ and m indices $T_0 = \{i_1, \dots, i_m\} \subseteq [q]$ uniformly at random, and interacts with an outside honest V by internally simulating an interaction of (S, V^{k*}) with V embedded in session j_0 , queries $\tau_{i_1}, \dots, \tau_{i_m}$ of V^{k*} . In comparison, the idea of guessing a random query subset is exactly as in [GK96b]. The difference is that the reduction in [GK96b] is for single session protocols, and in contrast, we reduce from parallel protocols to single session protocols. Hence, our reduction P^* guesses a random session as well.

In more details, P^* runs S and V^{k*} internally. It simulates $k - 1$ sessions of V^{k*} honestly (except $V_{j_0}^*$). When simulating $V_{j_0}^*$, for the i^{th} S query τ_i , P^* first simulate (with fresh randomness) $V_{j_0}^*$'s decision on termination. If $V_{j_0}^*$ decides to terminate but $i \in T_0$ or if $V_{j_0}^*$ does not terminate but $i \notin T_0$, P^* aborts (in both these cases, the termination decision of $V_{j_0}^*$ is incompatible with P^* 's choice of queries to forward). If the forwarded queries (index set T_0) are not “consistent” (e.g., if they query for the same round of the protocol more than once, or the query contains inconsistent transcript), P^* aborts as well. Note that if P^* does not abort, then V^{k*} is perfectly simulated (even in session j_0).

Now consider the following best case scenario. Suppose that at the end of the simulation, S successfully outputs an accepting view of V^{k*} . Moreover, suppose that the accepting view consists exactly of the queries in index set T_0 (this automatically guarantees that the forwarded queries are consistent), and suppose that P^* does not abort (i.e., termination decisions are compatible with the forwarded queries). Then, P^* will have successfully convinced the outside honest V . The rest of the proof is devoted to show that this best case scenario occurs with noticeable probability (roughly $1/(p \cdot k^2)$).

Let $T \subset [q]$ denote an index set $\{i_1, \dots, i_m\}$ of size m . For an index set $T \subset [q]$ and a session $j \in [k]$, we define $A(T, j)$ to be the event that, on session j , V^{k*} terminates on query τ_i iff $i \notin T$. Referring back to our intuition earlier, $A(T, j)$ denotes the event that for session j , S 's rewinds are not helpful with respect to the queries indexed by T . If event $A(T, j)$ holds, and S uses the queries indexed by T to form an accepting view of V^{k*} , and P^* guesses both $T_0 = T$ and $j_0 = j$ in the beginning, then P^* will have successfully convinced the outside honest V .

We claim that by the setting of parameters, we have

$$\Pr[\forall T \subset [q], \exists j \in [k] \text{ s.t. } A(T, j)] \geq 1 - \text{ngl}(n) \quad (1)$$

where $\text{ngl}(n)$ denotes a negligible quantity in n . In words, with overwhelming probability, for every possible index set T of size m that S may use to output a view of V^{k*} , there exists a session j such that P^* may guess $j_0 = j$ and be successful.

Before proving (1), we first use the claim to show that P^* convinces V with noticeable probability. Recall that S outputs an accepting view of V^{k*} with probability $1/p$. By a union bound, we have

$$\Pr[(S \text{ outputs accepting view of } V^{k*}) \wedge (\forall T \subset [q], \exists j \in [k] \text{ s.t. } A(T, j))] \geq (1/p) - \text{ngl}(n).$$

Note that when the above event holds, there exist a unique index $\hat{T}$ of m queries used by S to form an accepting view of V^{k*} , and there exists a session $\hat{j} \in [k]$ such that $A(\hat{T}, \hat{j})$ holds. As mentioned earlier, if P^* guesses $j_0 = \hat{j}$ and $T_0 = \hat{T}$ correctly, P^* will have successfully convinced V . Since P^* guesses j and T uniformly at random and independent of the interaction between S and V^{k*} , we have

$$\begin{aligned} & \Pr[P^* \text{ convinces } V] \\ & \geq \Pr[(S \text{ convinces } V^{k*}) \wedge (\forall T \subset [q], \exists j \in [k] \text{ s.t. } A(T, j)) \\ & \quad \wedge (P^* \text{ guesses } \hat{T} \text{ and } \hat{j} \text{ correctly})] \\ & \geq \frac{(1/p - \text{ngl}(n))}{k \cdot \binom{q}{m}} \geq \frac{1}{p \cdot k^2}, \end{aligned}$$

where in the last line we used $\binom{q}{m} \leq q^m \leq k$. This contradicts the fact that Π has negligible soundness error and completes our analysis.

It remains to show (1). By definition, each session j terminates on each query τ_i with probability exactly ρ , independent from any other session or query. Hence, for any session j and index set T of size m , the probability that event $A(T, j)$ holds is

$$\Pr[A(T, j)] = \rho^{q-m} \cdot (1 - \rho)^m \geq \left(1 - \frac{1}{q}\right)^q \cdot \left(\frac{1}{q}\right)^m \geq \Omega\left(\frac{1}{k} \cdot (\log^{2m} n)\right).$$

It follows that

$$\Pr[\exists j \in [k] \text{ s.t. } A(T, j)] \geq 1 - \left(1 - \Omega\left(\frac{1}{k} \cdot (\log^{2m} n)\right)\right)^k \geq 1 - e^{-\Omega(\log^{2m} n)}.$$

Finally, by a union bound, we have

$$\Pr[\forall T \subset [q], \exists j \in [k] \text{ s.t. } A(T, j)] \geq 1 - e^{-\Omega(\log^{2m} n)} \cdot \binom{q}{m} \geq 1 - \text{ngl}(n),$$

as claimed.

As with most lower bounds for black-box zero-knowledge, a careful reading reveals that Theorem 2 also applies to more liberal definitions of zero-knowledge, such as ε -zero-knowledge and zero-knowledge with expected polynomial time simulators. Additionally, note that the proof of Lemma 9 never assume that S is a zero-knowledge simulator, and works just as well for any PPT oracle machine S .

Remark 10. *By examining the technical inner workings of the proof of Canetti, Kilian, Petrank and Rosen [CKPR01] (which also uses a random termination verifier), we discovered that part of their analysis implicitly presents a lower bound for the number of queries made by black-box simulators for parallel zero-knowledge protocols. Compared with Theorem 2 and our analysis, the result of [CKPR01] establishes a weaker bound (and is arguably more complicated); this is not surprising, since establishing a parallel lower bound was not their goal.*

Specifically, [CKPR01] implicitly establishes a $\log^{\omega(1)}(k)$ lower bound on the number of simulator queries, whereas we were able to establish a lower bound of $k^{1/m}/(\log^2 n)$. Nevertheless, we believe that by adapting our parameters (which may seem strange for their setting), their analysis could be strengthened to match our lower bounds (we have not verified all the details, however).

Acknowledgments

We thank to Iftach Haitner and Johan Håstad for useful discussion in the early stage of this research.

References

- [Bar01] Boaz Barak. How to go beyond the black-box simulation barrier. In *FOCS '01*, pages 106–115, 2001.
- [BG02] Boaz Barak and Oded Goldreich. Universal arguments and their applications. In *Computational Complexity*, pages 162–171, 2002.
- [BL02] Boaz Barak and Yehuda Lindell. Strict polynomial-time in simulation and extraction. In *STOC '02*, pages 484–493, 2002.
- [Blu86] M. Blum. How to prove a theorem so no one else can claim it. *Proc. of the International Congress of Mathematicians*, pages 1444–1451, 1986.
- [CDS94] Ronald Cramer, Ivan Damgård, and Berry Schoenmakers. Proofs of partial knowledge and simplified design of witness hiding protocols. In *CRYPTO '94*, pages 174–187, 1994.
- [CKPR01] Ran Canetti, Joe Kilian, Erez Petrank, and Alon Rosen. Black-box concurrent zero-knowledge requires $\tilde{\omega}(\log n)$ rounds. In *STOC '01*, pages 570–579, 2001.

- [DNS04] Cynthia Dwork, Moni Naor, and Amit Sahai. Concurrent zero-knowledge. *J. ACM*, 51(6):851–898, 2004.
- [FS89] Uriel Feige and Adi Shamir. Zero knowledge proofs of knowledge in two rounds. In *CRYPTO*, pages 526–544, 1989.
- [FS90] Uriel Feige and Adi Shamir. Witness indistinguishable and witness hiding protocols. In *STOC*, pages 416–426, 1990.
- [GK96a] Oded Goldreich and Ariel Kahan. How to construct constant-round zero-knowledge proof systems for NP. *Journal of Cryptology*, 9(3):167–190, 1996.
- [GK96b] Oded Goldreich and Hugo Krawczyk. On the composition of zero-knowledge proof systems. *SIAM Journal on Computing*, 25(1):169–192, 1996.
- [GMR89] Shafi Goldwasser, Silvio Micali, and Charles Rackoff. The knowledge complexity of interactive proof systems. *SIAM Journal on Computing*, 18(1):186–208, 1989.
- [GMW91] Oded Goldreich, Silvio Micali, and Avi Wigderson. Proofs that yield nothing but their validity for all languages in NP have zero-knowledge proof systems. *J. ACM*, 38(3):691–729, 1991.
- [GO94] Oded Goldreich and Yair Oren. Definitions and properties of zero-knowledge proof systems. *Journal of Cryptology*, 7:1–32, 1994.
- [Gol01] Oded Goldreich. *Foundations of Cryptography — Basic Tools*. Cambridge University Press, 2001.
- [Gol02] Oded Goldreich. Concurrent zero-knowledge with timing, revisited. In *STOC '02*, pages 332–340, 2002.
- [Hai09] Iftach Haitner. A parallel repetition theorem for any interactive argument. In *FOCS '09*, pages 241–250, 2009.
- [HILL99] Johan Håstad, Russell Impagliazzo, Leonid Levin, and Michael Luby. A pseudorandom generator from any one-way function. *SIAM Journal on Computing*, 28:12–24, 1999.
- [HRS09] Iftach Haitner, Alon Rosen, and Ronen Shaltiel. On the (im)possibility of Arthur-Merlin witness hiding protocols. In *TCC '09*, pages 220–237, 2009.
- [Kat08] Jonathan Katz. Which languages have 4-round zero-knowledge proofs? In *Theory of Cryptography*, pages 73–88, 2008.
- [KPR98] Joe Kilian, Erez Petrank, and Charles Rackoff. Lower bounds for zero knowledge on the internet. In *FOCS '98*, pages 484–492, 1998.
- [MP06] Silvio Micali and Rafael Pass. Local zero knowledge. In *STOC '06*, pages 306–315, 2006.
- [Nao91] Moni Naor. Bit commitment using pseudorandomness. *Journal of Cryptology*, 4(2):151–158, 1991.
- [PPS⁺08] Omkant Pandey, Rafael Pass, Amit Sahai, Wei-Lung Dustin Tseng, and Muthuramakrishnan Venkatasubramanian. Precise concurrent zero knowledge. In *EUROCRYPT '08*, pages 397–414, 2008.
- [PTV10] Rafael Pass, Wei-Lung Dustin Tseng, and Muthuramakrishnan Venkatasubramanian. Eye for an eye: Efficient concurrent zero-knowledge in the timing model. In *TCC*, pages 518–534, 2010.
- [PTW09] Rafael Pass, Wei-Lung Dustin Tseng, and Douglas Wikström. On the composition of public-coin zero-knowledge protocols. In *CRYPTO '09*, pages 160–176, 2009.
- [PV08] Rafael Pass and Muthuramakrishnan Venkatasubramanian. On constant-round concurrent zero-knowledge. In *TCC '08*, pages 553–570, 2008.
- [Ros00] Alon Rosen. A note on the round-complexity of concurrent zero-knowledge. In *CRYPTO '00*, pages 451–468, 2000.

Computing on Authenticated Data

Jae Hyun Ahn Johns Hopkins University arjuna@cs.jhu.edu	Dan Boneh* Stanford University dabo@cs.stanford.edu	Jan Camenisch† IBM Research – Zurich jca@zurich.ibm.com
Susan Hohenberger‡ Johns Hopkins University susan@cs.jhu.edu	abhi shelat§ University of Virginia abhi@cs.virginia.edu	Brent Waters¶ University of Texas at Austin bwaters@cs.utexas.edu

December 21, 2011

Abstract

In tandem with recent progress on computing on encrypted data via fully homomorphic encryption, we present a framework for computing on *authenticated* data via the notion of slightly homomorphic signatures, or P -homomorphic signatures. With such signatures, it is possible for a third party to *derive* a signature on the object m' from a signature of m as long as $P(m, m') = 1$ for some predicate P which captures the “authenticatable relationship” between m' and m . Moreover, a derived signature on m' reveals *no extra information* about the parent m .

Our definition is carefully formulated to provide one unified framework for a variety of distinct concepts in this area, including arithmetic, homomorphic, quotable, redactable, transitive signatures and more. It includes being unable to distinguish a derived signature from a fresh one *even when given the original signature*. The inability to link derived signatures to their original sources prevents some practical privacy and linking attacks, which is a challenge not satisfied by most prior works.

Under this strong definition, we then provide generic constructions for all univariate and closed predicates, and specific efficient constructions for a broad class of natural predicates such as quoting, subsets, weighted sums, averages, and Fourier transforms. To our knowledge, these are the first efficient constructions for these predicates (excluding subsets) that provably satisfy this strong security notion.

*Supported by NSF, DARPA, and AFOSR. Applying to all authors, the views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the Defense Advanced Research Projects Agency or the US government.

†This work has been funded by the European Community’s Seventh Framework Programme (FP7/2007-2013) under grant agreement no. 216483 (PrimeLife).

‡Supported by the Defense Advanced Research Projects Agency (DARPA) and the Air Force Research Laboratory (AFRL) under contract FA8750-11-2-0211, the Office of Naval Research under contract N00014-11-1-0470, a Microsoft Faculty Fellowship and a Google Faculty Research Award.

§Supported by NSF CNS-0845811 and TC-1018543, Defense Advanced Research Projects Agency (DARPA) and the Air Force Research Laboratory (AFRL) under contract FA8750-11-2-0211, and a Microsoft New Faculty Fellowship.

¶Supported by NSF CNS-0915361 and CNS-0952692, AFOSR Grant No: FA9550-08-1-0352, DARPA PROCEED, DARPA N11AP20006, Google Faculty Research award, the Alfred P. Sloan Fellowship, Microsoft Faculty Fellowship, and Packard Foundation Fellowship.

1 Introduction

In tandem with recent progress on computing *any function* on encrypted data, e.g., [28, 54, 51], this work explores computing on unencrypted signed data. In the past few years, several independent lines of research touched on this area:

- Quoting/redacting: [53, 34, 1, 41, 31, 18, 17, 19] Given Alice’s signature on some message m anyone should be able to derive Alice’s signature on a subset of m . Quoting typically applies to signed text messages where one wants to derive Alice’s signature on a substring of m . Quoting can also apply to signed images where one wants to derive a signature on a subregion of the image (say, a face or an object) and to data structures where one wants to derive a signature of a subset of the data structure such as a sub-tree of a tree.
- Arithmetic: [35, 60, 23, 14, 27, 13, 12, 58] Given Alice’s signature on vectors $\mathbf{v}_1, \dots, \mathbf{v}_k \in \mathbb{F}_p^n$ anyone should be able to derive Alice’s signature on a vector $\mathbf{v}$ in the linear span of $\mathbf{v}_1, \dots, \mathbf{v}_k$. Arithmetic on signed data is motivated by applications to secure network coding [26]. We show that these schemes can be used to compute authenticated linear operations such as computing an authenticated weighted sum of signed data and an authenticated Fourier transform. As a practical consequence of this, we show that an untrusted database storing signed data (e.g., employee salaries) can publish an authenticated average of the data without leaking any other information about the stored data. Recent constructions go beyond linear operations and support low degree polynomial computations [12].
- Transitivity: [46, 40, 5, 32, 6, 49, 59, 45] Given Alice’s signature on edges in a graph G anyone should be able to derive Alice’s signature on a pair of vertices (u, v) if and only if there is a path in G from u to v . The derived signature on the pair (u, v) must be indistinguishable from a fresh signature on (u, v) had Alice generated one herself [40]. This requirement ensures that the derived signature on (u, v) reveals no information about the path from u to v used to derive the signature.

In this paper, we put forth a general framework for computing on authenticated data that encompasses these lines of research and much more. While prior definitions mostly contained artifacts specific to the type of malleability they supported and, thus, were hard to compare to one another, we generalize and strengthen these disparate notions into a single definition. This definition can be instantiated with any predicate, and we allow repeated computation on the signatures (e.g., it is possible to quote from a quoted signature.) During our study, we realized that the “privacy” notions offered by many existing definitions are, in our view, insufficient for some practical applications. We therefore require a stronger (and seemingly a significantly more challenging to achieve) property called *context hiding*. Under this definition, we provide two generic solutions for computing signatures on any univariate, closed predicate; however, these generic constructions are not efficient. We also present efficient constructions for three problems: quoting substrings in Section 5, a subset predicate in Section 6, and a weighted average over data in Section 7 (which captures weighted sums and Fourier transforms). Our quoting substring construction is novel and significantly more efficient than the generic solutions. For the problems of subsets and weighted averages, we show somewhat surprising connections to respective existing solutions in attribute-based encryption and network coding signatures.

1.1 Overview

A general framework. Let $\mathcal{M}$ be some message space and let $2^{\mathcal{M}}$ be its powerset. Consider a predicate $P : 2^{\mathcal{M}} \times \mathcal{M} \rightarrow \{0, 1\}$ mapping a set of messages and a message to a bit. Loosely speaking we say that a signature scheme supports computations with respect to P if the following holds:

Let $M \subset \mathcal{M}$ be a set of messages and let m' be a *derived* message, namely m' satisfies $P(M, m') = 1$. Then there exists an efficient procedure that can derive Alice's signature on m' from Alice's independent signatures on all of the messages in M .

For the quoting application, the predicate P is defined as $P(M, m') = 1$ iff m' is a quote from the set of messages M . Here we focus on quoting from a single message m so that P is false whenever M contains more than one component¹, and thus use the notation $P(m, m')$ as shorthand for $P(\{m\}, m')$. The predicate P for arithmetic computations is defined in Appendix A and essentially says that $P((\mathbf{v}_1, \dots, \mathbf{v}_k), \mathbf{v})$ is true whenever $\mathbf{v}$ is in the span of $\mathbf{v}_1, \dots, \mathbf{v}_k$.

We emphasize that signature derivation can be iterative. For example, given a message-signature pair (m, σ) from Alice, Bob can publish a derived message-signature pair (m', σ') for an m' where $P(m, m')$ holds. Charlie, using (m', σ') , may further derive a signature σ'' on m'' . In the quoting application, Charlie is quoting from a quote which is perfectly fine.

Security. We give a clean security definition that captures two properties: unforgeability and context hiding. We briefly discuss each in turn and give precise definitions in the next section.

- Unforgeability captures the idea that an attacker may be given various derived signatures (perhaps iteratively derived) on messages of his choice. The attacker should be unable to produce a signature on a message that is not derivable from the set of signed messages at his possession. E.g., suppose Alice generates (m, σ) and gives it to Bob who then publishes a derived signature (m', σ') . Then an attacker given (m', σ') should be unable to produce a signature on m or on any other message m'' such that $P(m', m'') = 0$.
- Context hiding captures an important privacy property: a signature should reveal nothing more than the message being signed. In particular, if a signature on m' was derived from a signature on m , an attacker should not learn anything about m other than what can be inferred from m' . This should be true even if the original signature on m is revealed. For example, a signed quote should not reveal anything about the message from which it was quoted, including its length, the position of the quote, whether its parent document is the same as another quote, whether it was derived from a given signed message or generated freshly, etc.

Defining context hiding is an interesting and subtle task. In the next section, we give a definition that captures a very strong privacy requirement. We discuss earlier attempts at defining privacy following our definition in Section 2.3; while many prior works use a similar sounding *intuition* as we give above, most contain a fundamental difference to ours in their *formalization*.

We note that notions such as group or ring signatures [24, 4, 20, 10, 48] have considered the problem of hiding the identity of a signer among a set of users. Context hiding ensures privacy for the data rather than the signer. Our goal is to hide the legacy of how a signature was created.

¹We leave it for future work to construct systems for securely quoting from two messages (or possibly more) as defined next.

Efficiency. We require that the size of a signature, whether fresh or derived, depend only on the size of the object being signed. This rules out solutions where the signature grows with each derivation.

Generic Approaches. We begin with two generic constructions that can be inefficient. They apply to *closed, univariate* predicates, namely predicates $P(M, m')$ where M contains a single message (P is false when $|M| > 1$) and where if $P(a, b) = P(b, c) = 1$ then $P(a, c) = 1$. The first construction uses any standard signature scheme S where the signing algorithm is deterministic. (One can enforce determinism using PRFs [29].) To sign a message $m \in \mathcal{M}$, one uses S to sign each message m' such that $P(m, m') = 1$. The signature consists of all these signature components. To verify a signature for m , one checks the signature component corresponding to the message m . To derive a signature m' from m , one copies the signature components for all m'' such that $P(m', m'') = 1$. Soundness of the construction follows from the security of the underlying standard scheme S and context hiding from the fact that signing in S is deterministic.

Unfortunately, these signatures may become large consisting up to $|\mathcal{M}|$ signature components — effecting both the signing time and signature size. Our second generic construction alleviates the space burden by using an RSA accumulator. The construction works in a similar brute force fashion where a signature on m is an accumulator value on all m' such that $P(m, m') = 1$. While this produces short signatures, the time component of both verification and derivation are even worse than the first generic approach. Thus, these generic approaches are too expensive for most interesting predicates. We detail these generic approaches and proofs in Section 4, where we also discuss a generic construction using NIZK.

Our Quoting Construction. We turn to more efficient constructions. First, we set out to construct a signature for quoting *substrings*², which although conceptually simple is non-trivial to realize securely. As an efficiency baseline, we note that the brute force generic construction of the quoting predicate would result in n^2 components for a signature on n characters. So any interesting construction must perform more efficiently than this. We prove our construction selectively secure.³ In addition, we give some potential future directions for achieving adaptive security and removing the use of random oracles.

Our construction uses bilinear groups to link different signature components together securely, but in such a way that the context can be hidden by a re-randomizing step in the derivation algorithm. A signature in our system on a message of length n consists of $n \lg n$ group elements; intuitively organized as $\lg n$ group elements assigned to each character. To derive a new signature on a substring of ℓ characters, one roughly removes the group elements not associated with the new substring and then re-randomizes the remaining part of the signature. This results in a new signature of $\ell \lg \ell$ group elements. The technical challenge consists in simultaneously allowing re-randomization and preserving the “linking” between successive characters. In addition, there is a second option in our derive algorithm that allows for the derivation of a short signature of $\lg \ell$ group elements; however the derive procedure cannot be applied again to this short signature. *Thus, we support quoting from quotes, and also provide a compression option which produces a very short quote, but the price for this is that it cannot be quoted from further.*

²A substring of $x_1 \dots x_n$ is some $x_i \dots x_j$ where $i, j \in [1, n]$ and $i \leq j$. We emphasize that we are not considering *subsequences*. Thus, it is *not* possible, in this setting, to extract a signature on “I like fish” from one on “I do not like fish”.

³Following an analog of [21], selective security for signatures requires the attacker to give the forgery message before seeing the verification key.

Computing Signatures on Subsets and Weighted Averages. Our final two contributions are schemes for deriving signatures on subsets and weighted averages on signatures. Rather than create entirely new systems, we show connections to existing Attribute-Based Encryption schemes and Network Coding Signatures. Briefly, our subset construction extends the concept of Naor [11] who observed that every IBE scheme can be transformed into a standard signature scheme by applying the IBE KeyGen algorithm as a signing algorithm. Here we show an analog for known Ciphertext-Policy (CP) ABE schemes. The KeyGen algorithm which generates a key for a set S of attributes can be used as a signing algorithm for the set S . For known CP-ABE systems [7, 36, 57] it is straightforward to derive a key for a subset S' of S and to re-randomize the signature/key. To verify a signature on S we can apply Naor's signature-from-IBE idea and encrypt a random message X to a policy that is an AND of all the attributes in S and see if the signature can be used as an ABE key to decrypt to X . Signatures for subsets have been previously considered in [32, §6.4], but without context hiding requirements. We provide further details in Section 6. Our construction for weighted sums is presented in Section 7, where we discuss how this applies to Fourier transforms.

Other Predicates. One can also imagine predicates P that support more complex operations on signed messages. One natural set of examples are spreadsheet operations such as median, standard deviation, and rounding on signed data (satisfying unforgeability and context hiding). Other examples include graph algorithms such as computing a signature on a perfect matching in a signed bipartite graph.

2 Definitions

Definition 2.1 (Derived messages) Let $\mathcal{M}$ be a message space and let $P : 2^{\mathcal{M}} \times \mathcal{M} \rightarrow \{0, 1\}$ be a predicate from sets over $\mathcal{M}$ and a message in $\mathcal{M}$ to a bit. We say that a message m' is **derivable** from the set $M \subseteq \mathcal{M}$ if $P(M, m') = 1$. We denote by $P^*(M)$ the set of messages derivable from M by repeated derivation. That is, let $P^0(M)$ be the set of messages derivable from M and for $i > 0$ let $P^i(M)$ be the set of messages derivable from $P^{i-1}(M)$. Then $P^*(M) := \cup_{i=0}^{\infty} P^i(M)$.

We define the closure of P , denoted P^* , as the predicate defined by $P^*(M, m) = 1$ iff $m \in P^*(M)$.

A P -homomorphic signature scheme Π for message space $\mathcal{M}$ and predicate P is a triple of PPT algorithms:

KeyGen(1^λ): the key generation algorithm outputs a key pair (pk, sk) . We treat the secret key sk as a signature on the empty tuple $\varepsilon \in \mathcal{M}^*$. We also assume that pk is embedded in sk .

SignDerive($pk, (\{\sigma_m\}_{m \in M}, M), m', w$): the algorithm takes as input the public key, a set of messages $M \subseteq \mathcal{M}$ and corresponding signatures $\{\sigma_m\}_{m \in M}$, a derived message $m' \in \mathcal{M}$, and possibly some auxiliary information w . It produces a new signature σ' or a special symbol $\perp$ to represent failure. For complicated predicates P , the auxiliary information w serves as a witness that $P(M, m') = 1$. To simplify the notation we often drop w as an explicit argument.

As shorthand we write $\mathbf{Sign}(sk, m) := \mathbf{SignDerive}(pk, (sk, \varepsilon), m, \cdot)$ to denote that any message can be derived when the original signature is the signing key. For a set of messages $M = \{m_1, \dots, m_k\} \subset \mathcal{M}^*$ it is convenient to let $\mathbf{Sign}(sk, M)$ denote independently signing each of the k messages, namely:

$$\mathbf{Sign}(sk, M) := (\mathbf{Sign}(sk, m_1), \dots, \mathbf{Sign}(sk, m_k)) .$$

Verify(pk, m, σ): given a public key, message, and purported signature σ , the algorithm returns 1 if the signature is valid and 0 otherwise.

We assume that testing $m \in \mathcal{M}$ can be done efficiently, and that **Verify** returns 0 if $m \notin \mathcal{M}$.

Correctness. We require that for all key pairs (sk, pk) generated by **KeyGen**(1^n) and for all $M \in \mathcal{M}^*$ and $m' \in \mathcal{M}$ we have:

- if $P(M, m') = 1$ then **SignDerive**($pk, (\mathbf{Sign}(sk, M), M), m'$) $\neq \perp$, and
- for all signature tuples $\{\sigma_m\}_{m \in M}$ such that $\sigma' \leftarrow \mathbf{SignDerive}(pk, (\{\sigma_m\}_{m \in M}, M), m') \neq \perp$, we have **Verify**(pk, m', σ') = 1.

In particular, correctness implies that a signature generated by **SignDerive** can be used as an input to **SignDerive** so that signatures can be further derived from derived signatures, if allowed by P .

Derivation efficiency. In many cases it is desirable that the size of a derived signature depend only on the size of the derived message. This rules out signatures that expand as one iteratively calls **SignDerive**. All the constructions in this paper are derivation efficient in this sense.

Definition 2.2 (Derivation-Efficient) *A signature scheme is derivation-efficient if there exists a polynomial p such that for all $(pk, sk) \leftarrow \mathbf{KeyGen}(1^\lambda)$, set $M \subseteq \mathcal{M}^*$, signatures $\{\sigma_m\}_{m \in M} \leftarrow \mathbf{Sign}(sk, M)$ and derived messages m' where $P(M, m') = 1$, we have*

$$|\mathbf{SignDerive}(pk, \{\sigma_m\}_{m \in M}, M, m')| = p(\lambda, |m'|).$$

2.1 Security: Unforgeability

To define unforgeability, we extend the basic notion of existential unforgeability with respect to adaptive chosen-message attacks [30]. The definition captures the idea that if the attacker is given a set of signed messages (either primary or derived) then the only messages he can sign are derivations of the signed messages he was given. This is defined using a game between a challenger and an adversary $\mathcal{A}$ with respect to scheme Π over message space $\mathcal{M}$.

— Game **Unforg**($\Pi, \mathcal{A}, \lambda, P$):

Setup: The challenger runs **KeyGen**(1^λ) to obtain (pk, sk) and sends pk to $\mathcal{A}$. The challenger maintains two sets T and Q that are initially empty.

Queries: Proceeding adaptively, the adversary issues the following queries to the challenger:

- $\mathbf{Sign}(m \in \mathcal{M})$: the challenger generates a unique handle h , runs $\mathbf{Sign}(sk, m) \rightarrow \sigma$ and places (h, m, σ) into a table T . It returns the handle h to the adversary.
- $\mathbf{SignDerive}(\vec{h} = (h_1, \dots, h_k), m')$: the oracle retrieves the tuples (h_i, σ_i, m_i) in T for $i = 1, \dots, k$, returning $\perp$ if any of them do not exist. Let $M := (m_1, \dots, m_k)$ and $\{\sigma_m\}_{m \in M} := \{\sigma_1, \dots, \sigma_k\}$. If $P(M, m')$ holds, then the oracle generates a new unique handle h' , runs $\mathbf{SignDerive}(pk, (\{\sigma_m\}_{m \in M}, M), m') \rightarrow \sigma'$ and places (h', m', σ') into T , and returns h' to the adversary.
- $\mathbf{Reveal}(h)$: Returns the signature σ corresponding to handle h , and adds (σ', m') to the set Q .

Output: Eventually, the adversary outputs a pair (σ', m') . The output of the game is 1 (i.e., the adversary wins the game) if:

- **Verify** $(pk, m', \sigma') = 1$ and,
- let $M \subseteq \mathcal{M}$ be the set of messages in Q then $P^*(M, m') = 0$ where P^* is the closure of P from Definition 2.1.

Else, the output of the game is 0. Define **Forg** $_{\mathcal{A}}$ as the probability that $\Pr[\mathbf{Unforg}(\Pi, \mathcal{A}, \lambda, P) = 1]$.

Interestingly, for some predicates it may be difficult to test if the adversary won the game. For all the predicates we consider in this paper, this will be quite easy.

Definition 2.3 (Unforgeability) *A P -homomorphic signature scheme Π is **unforgeable** with respect to adaptive chosen-message attacks if for all PPT adversaries $\mathcal{A}$, the function **Forg** $_{\mathcal{A}}$ is negligible in λ .*

*A P -homomorphic signature scheme Π is **selective unforgeable** with respect to adaptive chosen-message attacks if for all PPT adversaries $\mathcal{A}$ who begin the above game by announcing the message m' on which they will forge, **Forg** $_{\mathcal{A}}$ is negligible in λ .*

Properties of the definition. By taking P to be the equality oracle, namely $P(x, y) = 1$ iff $x = y$, we obtain the standard unforgeability requirement for signatures.

Notice that *Sign* and *SignDerive* queries return handles, but do not return the actual signatures. A system proven secure under this definition adequately rules out the following attack: suppose (m, σ) is a message signature pair and (m', σ') is a message-signature pair derived from it, namely $\sigma' = \mathbf{SignDerive}(pk, \sigma, m, m')$. For example, suppose m' is a quote from m . Then given (m', σ') it should be difficult to produce a signature on m and indeed our definition treats a signature on m as a valid forgery.

The unforgeability game imposes some constraints on P : (1) P must be reflexive, i.e. $P(m, m) = 1$ for all $m \in \mathcal{M}$, (2) P must be monotone, i.e. $P(M, m') \Rightarrow P(M', m')$ where $M \subseteq M'$. It is easy to see that predicates that do not satisfy these requirements cannot be realized under Definition 2.3.

2.2 Security: Context Hiding (a.k.a., Privacy)

Let M be some set and let m' be a derived message from M (i.e., $P(M, m') = 1$). Context hiding captures the idea that a signature on m' derived from signatures on M should reveal no information about M beyond what is revealed by m' . For example, in the case of quoting, a signature on a quote from m should reveal nothing more about m : not the length of m , not the position of the quote in m , etc. The same should hold even if the attacker is given signatures on multiple quotes from m .

We put forth the following powerful *statistical* definition of context hiding and discuss its implications following the definition. We were most easily able to leverage a statistical definition for our proofs, although we also give an alternative *computational* definition in Appendix A.

Definition 2.4 (Strong Context Hiding) *Let $M \subseteq \mathcal{M}^*$ and $m' \in \mathcal{M}$ be messages such that $P(M, m') = 1$. Let $(pk, sk) \leftarrow \mathbf{KeyGen}(1^\lambda)$ be a key pair. A signature scheme $(\mathbf{KeyGen}, \mathbf{SignDerive}, \mathbf{Verify})$ is **strongly context hiding** (for predicate P) if for all such triples $((pk, sk), M, m')$, the following two distributions are statistically close:*

$$\begin{aligned} & \left\{ (sk, \{\sigma_m\}_{m \in M} \leftarrow \mathbf{Sign}(sk, M), \mathbf{Sign}(sk, m')) \right\}_{sk, M, m'} \\ & \left\{ (sk, \{\sigma_m\}_{m \in M} \leftarrow \mathbf{Sign}(sk, M), \mathbf{SignDerive}(pk, (\{\sigma_m\}_{m \in M}, M), m')) \right\}_{sk, M, m'} \end{aligned}$$

*The distributions are taken over the coins of **Sign** and **SignDerive**. Without loss of generality, we assume that pk can be computed from sk .*

The definition states that a derived signature on m' , from an honestly-generated original signature, is statistically indistinguishable from a fresh signature on m' . This implies that a derived signature on m' is indistinguishable from a signature generated independently of M . Therefore, the derived signature cannot (provably) reveal any information about M beyond what is revealed by m' . By a simple hybrid argument the same holds even if the adversary is given multiple derived signatures from M .

Moreover, Definition 2.4 requires that a derived signature look like a fresh signature even if the original signature on M is known. Hence, if for example someone quotes from a signed recommendation letter and somehow the original signed recommendation letter becomes public, it would be impossible to link the signed quote to the original signed letter. The same holds even if the signing key sk is leaked.

Thus, Definition 2.4 captures a broad range of privacy requirements for derived signatures. Earlier work in this area [34, 17, 19, 16] only considered weaker privacy requirements using more complex definitions. The simplicity and breadth of Definition 2.4 is one of our key contributions.

Definition 2.4 uses statistical indistinguishability meaning that even an unbounded adversary cannot distinguish derived signatures from newly created ones. In Appendix A, we give a definition using computational indistinguishability which is considerably more complex since the adversary needs to be given signing oracles. In the unbounded case of Definition 2.4 the adversary can simply recover a secret key sk from the public key and answer its own signature queries which greatly simplifies the definition of context hiding. All the signature schemes in this paper satisfy the statistical Definition 2.4.

As mentioned above, the context-hiding guarantee applies to all derivations that begin with an honestly-generated signature. One might imagine a scenario where a malicious signer creates a signature that passes the verification algorithm, but contains a “watermark” that allows the signer to detect if other signatures are derived from it. To prevent such attacks from malicious signers, we could alter the definition so that indistinguishability holds for any derivative that results from a signature that passed the verification algorithm.

A simpler approach to proving unforgeability. For systems that are strongly context hiding, unforgeability follows from a simpler game than that of Section 2.1. In particular, it suffices to just give the adversary the ability to obtain top level signatures signed by sk . In Appendix A, we define this simpler unforgeability game and prove equivalence to Definition 2.3 using strong context hiding.

2.3 Related Work

Early work on quotable signatures [53, 34, 43, 42, 31, 18, 22, 16] supports quoting from a single document, but does not achieve the privacy or unforgeability properties we are aiming for. For example, if *simple quoting* of messages is all that is desired, then the following folklore solution would suffice: simply sign the Merkle hash of a document. A quote represents some sub-tree of the Merkle hash; so a quoter could include enough intermediate hash nodes along with the original signature in any quote. A verifier could simply hash the quote, and then build the Merkle hash tree using the computed hash and the intermediate hashes, and compare with the original signature. Notice, however, that every quote in this scheme reveals information about the original source

document. In particular, each quote reveals information about *where in the document* it appears. Thus, this simple quoting scheme is not *context hiding* in our sense.

The work whose definition is closest to what we envision is the recent work on redacted signatures of Chang et al. [22] and Brzuska et al. [16] (see also Naccache [44, p. 63] and Boneh-Freeman [13, 12]⁴). However, there is a subtle, but fundamental difference between their definition and the privacy notion we are aiming for. In our formulation, a quoted signature should be indistinguishable from a fresh signature, even when the distinguisher is given the original signature. (We capture this by an even stronger game where a derived signature is distributed statistically close to a fresh signature.) In contrast, the definitions of [22, 16, 13, 12] do not provide the distinguisher with the original signature. Thus, it may be possible to link a quoted document to its original source (and indeed it is in the constructions of [22, 16, 13, 12]), which can have negative privacy implications. Overcoming such document linkage while maintaining unforgeability is a real technical challenge. This requires moving beyond techniques that use *nonces* to link parts of messages.

Indeed, in most prior constructions, such as [22, 16], nonces are used to prevent “mix-and-match” attacks (e.g., forming a “quote” using pieces of two different messages.) Unfortunately, these nonces reveal the history of derivation, since they cannot change during each derivation operation. Arguably, much of the technical difficulty in our current work comes precisely from the effort to meet our definition and hide the lineage. We introduce new techniques in this work which link pieces together using randomness that can be re-randomized in controlled ways.

Another line of work studies computing on authenticated data by holders of secret information. Examples include *sanitizable* signatures [43, 1, 41, 19, 17] that allow a proxy to compute signatures on related messages, but requires the proxy to have a secret key, and *incremental* signatures [3], where the signer can efficiently make small edits to his signed data. In contrast, our proposal is more along the lines of homomorphic encryption and Rivest’s vision [46], where *anyone* can compute on the authenticated data.

3 Preliminaries: Algebraic Settings

Bilinear Groups and the CDH Assumption. Let $\mathbb{G}$ and $\mathbb{G}_T$ be groups of prime order p . A *bilinear map* is an efficient mapping $\mathbf{e} : \mathbb{G} \times \mathbb{G} \rightarrow \mathbb{G}_T$ which is both: (*bilinear*) for all $g \in \mathbb{G}$ and $a, b \leftarrow \mathbb{Z}_p$, $\mathbf{e}(g^a, g^b) = \mathbf{e}(g, g)^{ab}$; and (*non-degenerate*) if g generates $\mathbb{G}$, then $\mathbf{e}(g, g) \neq 1$. We will focus on the Computational Diffie-Hellman assumption in these groups.

Assumption 3.1 (CDH [25]) *Let g generate a group $\mathbb{G}$ of prime order $p \in \Theta(2^\lambda)$. For all PPT adversaries $\mathcal{A}$, the following probability is negligible in λ : $\Pr[a, b, \leftarrow \mathbb{Z}_p; z \leftarrow \mathcal{A}(g, g^a, g^b) : z = g^{ab}]$.*

4 Generic Constructions for Simple Predicates

Let $\mathcal{M}$ be a finite message space. We say that a predicate $P : \mathcal{M}^* \times \mathcal{M} \rightarrow \{0, 1\}$ is a *simple predicate* if the following properties hold:

⁴As acknowledged in Section 2.2 of Boneh-Freeman [12], our definitional notion is stronger than and predates the “weak context hiding” notion of [12]. Indeed, the fact that [12] uses our framework lends support to its generality, and the fact that they could not achieve our context hiding notion highlights its difficulty. Their “weak” definition, which is equivalent to [16], only ensures privacy when the original signatures remain hidden. In their system, signature derivation is deterministic and therefore once the original signatures become public it is easy to tell where the derived signature came from. Our signatures achieve full context hiding so that derived signatures remain private no matter what information is revealed. This is considerably harder and is not known how to do for the lattice-based signatures in Boneh-Freeman.

1. P is false whenever its left input is a tuple of length greater than 1,
2. P is a closed predicate (i.e., P is equal to its closure P^* ; see Section 2.1.)
3. For all $m \in \mathcal{M}$, $P(m, m) = 1$.

In this section, we present and discuss generic approaches to computing on authenticated data with respect to *any* simple predicate P . Note that the quoting of substrings or subsequences (i.e., redacting) are examples of simple predicates.

We begin with two inefficient constructions. The first takes a brute force approach that constructs long signatures that are easy to verify. The second takes an accumulator approach that constructs shorter signatures at the cost of less efficient verification. We conclude by discussing the limitations of a generic NIZK proof of knowledge approach.

4.1 A Brute Force Construction From Any Signature Scheme

Let (G, S, V) be a signature scheme with a deterministic signing algorithm.⁵ One can construct a P -homomorphic signature scheme for any simple predicate P as follows:

KeyGen (1^λ) : The setup algorithm runs $G(1^\lambda) \rightarrow (pk, sk)$ and outputs this key pair.

Sign $(sk, m \in \mathcal{M})$: While **Sign** is simply a special case of the **SignDerive** algorithm, we will explicitly provide both algorithms here for clarity purposes.

The signature σ is the tuple $(S(sk, m), U = \{S(sk, m') \mid m' \in P^0(\{m\})\})$.

SignDerive (pk, σ, m, m') : The derived signature is computed as follows. First check that $P(m, m') = 1$. If not, then output $\perp$. Otherwise, parse $\sigma = (\sigma_1, \dots, \sigma_k)$ where σ_i corresponds to message m_i . If for any i , $V(pk, m_i, \sigma_i) = 0$, then output $\perp$. Otherwise, the signature is comprised as the set containing σ_i for all m_i such that $P(m', m_i) = 1$. Again, by default, let the first sub-signature of the output be the signature on m' .

Verify (pk, m, σ) : Parse $\sigma = (\sigma_1, \dots, \sigma_k)$. Output $V(pk, m, \sigma_1)$.

Efficiency Discussion The efficiency of the above approach depends on the message space and the predicate P . For instance, the brute force approach for signing a message of n characters, where $P(m, m')$ outputs 1 if and only if m' is a substring of m , will result in $O(n^2)$ sub-signatures (one for each of the $O(n^2)$ substrings). If one wanted to “quote” subgraphs from a graph, this approach is intractable, as a graph of n nodes will generate an exponential in n number of subgraphs.

Theorem 4.1 (Security from Any Signature) *If (G, S, V) is a secure deterministic signature scheme, then the above signature scheme is unforgeable and context-hiding.*

Proof of the above theorem is rather straightforward. The context-hiding property follows from the uniqueness of the signatures generated by the honest signing algorithms. The unforgeability property follows from the fact that an adversary cannot obtain a signature on any message not derivable from those she queried or one could use this signature to directly break the regular unforgeability of the underlying signature scheme. The correctness property is actually the most complex to verify: it requires the two restrictions on the predicate P made above.

⁵Given a signature scheme with a probabilistic signing algorithm, one can convert it to a scheme with a deterministic signing algorithm by: (1) including a pseudorandom function (PRF) seed as part of the secret key and (2) during the signing algorithm, applying this PRF to the message and using the output as the randomness in the signature. Given any signature scheme, one can also construct a PRF.

4.2 An Accumulator-based Construction

Assumption 4.2 (RSA [47]) *Let k be the security parameter. Let a positive integer N be the product of two random k -bit primes p, q . Let e be a randomly chosen positive integer less than and relatively prime to $\phi(N) = (p-1)(q-1)$. Then no PPT algorithm given (N, e) and a random $y \in \mathbb{Z}_N^*$ as input can compute x such that $x^e \equiv y \pmod{N}$ with non-negligible probability.*

Lemma 4.3 (Shamir [50]) *Given $x, y \in \mathbb{Z}_n$ together with $a, b \in \mathbb{Z}$ such that $x^a = y^b$ and $\gcd(a, b) = 1$, there is an efficient algorithm for computing $z \in \mathbb{Z}_n$ such that $z^a = y$.*

Theorem 4.4 (Prime Number Theorem) *Define $\pi(x)$ as the number of primes no larger than x . For $x > 1$,*

$$\pi(x) > \frac{x}{\lg x}.$$

Consider the following RSA accumulator solution which supports short signatures, but the computation required to derive a new signature is expensive. Let P be any univariate predicate with the above restrictions.

We now describe the algorithms. While **Sign** is simply a special case of the **SignDerive** algorithm, we will explicitly provide both algorithms here for clarity purposes.

KeyGen(1^λ) : The setup algorithm chooses N as a 20λ -bit RSA modulus and a random value $a \in \mathbb{Z}_N$. It also chooses a hash function H_p that maps arbitrary strings to 2λ -bit prime numbers, e.g., [33], which we treat as a random oracle.⁶ Output the public key $pk = (H_p, N, a)$ and keep as the secret key sk , the factorization of N .

Sign($sk, m \in \mathcal{M}$) : Let $U = P^0(\{m\}) = \{m' \mid m' \in \mathcal{M} \text{ and } P(m, m') = 1\}$. Compute and output the signature as

$$\sigma := a^{1/(\prod_{u_i \in U} H_p(u_i))} \pmod{N}.$$

SignDerive(pk, σ, m, m') : The derivation is computed as follows. First check that $P(m, m') = 1$. If not, then output $\perp$. Otherwise, let $U' = P^0(\{m'\})$. Compute and output the signature as

$$\sigma' := \sigma^{\prod_{u_i \in U - U'} H_p(u_i)} \pmod{N}.$$

Thus, the signature is of the form $a^{1/\prod_{u_i \in U'} H_p(u_i)} \pmod{N}$.

Verify(pk, m, σ) : Accept if and only if $a = \sigma^{\prod_{u_i \in U} H_p(u_i)} \pmod{N}$ where $U = P^0(m)$.

Efficiency Discussion In the above scheme, signatures require only one element in $\mathbb{Z}_N^*$. However, the cost of signing depends on P and the size of the message space. For example, computing an ℓ -symbol quote from an n -symbol message requires $O(n(n-\ell))$ evaluations of $H_p()$ and $O(n(n-\ell))$ modular exponentiations. The prime search component of H_p will likely be the dominating factor. Verification requires $O(\ell^2)$ evaluations of $H_p()$ and $O(\ell^2)$ modular exponentiations, for an ℓ -symbol quote. Thus, this scheme optimizes on space, but may require significant computation.

Theorem 4.5 (Security under RSA) *If the RSA assumption holds, then the above signature scheme is unforgeable and context-hiding in the random oracle model.*

We provide a proof of above theorem by showing the following lemmas.

⁶We choose our modulus and hash output lengths to obtain λ -bit security based on the recent estimates of [52].

Lemma 4.6 (Context-Hiding) *The homomorphic signature scheme from §4.2 is strongly context-hiding.*

Proof. This property is derived from the fact that a signature on any given message is deterministic. Let the public key PK be (H_p, N, a) and challenge be any m, m' where $P(m, m') = 1$. Let $U = P^0(m)$ and $U' = P^0(m')$. Observe that

$$\begin{aligned} \text{Sign}(sk, m) &= \sigma = a^{1/\prod_{u \in U} H_p(u)} \mod N \\ \text{Sign}(sk, m') &= \sigma_0 = a^{1/\prod_{u' \in U'} H_p(u')} \mod N \\ \text{SignDerive}(pk, (\sigma, m), m') &= \sigma^{\prod_{u \in U - U'} H_p(u)} \mod N \\ &= \left[a^{1/\prod_{u \in U} H_p(u)} \right]^{\prod_{u \in U - U'} H_p(u)} \mod N \\ &= a^{1/\prod_{u' \in U'} H_p(u')} \mod N \\ &= \sigma_0 \end{aligned}$$

Because $\text{Sign}(sk, m')$ and $\text{SignDerive}(pk, (\sigma, m), m')$ are identical, for any adversary $\mathcal{A}$, the probability that $\mathcal{A}$ distinguishes the two is exactly $1/2$, and so the advantage in the strong context hiding game is 0. $\square$

Lemma 4.7 (Unforgeability) *If the RSA assumption holds, then the Section 4.2 homomorphic signature scheme is unforgeable in the **Unforg** game in the random oracle model.*

Proof. Our reduction only works on certain types of RSA challenges, as in [33]. In particular, this reduction only attempts to solve RSA challenges (N, e^*, y) where e^* is an odd prime. Fortunately, good challenges will occur with non-negligible probability. We know that e^* is less than and relatively prime to $\phi(N) < N$, which implies it cannot be 2. We also know, by Theorem 4.4, that the number of primes that are less than N is at least $\frac{N}{\lg N}$. Thus, a loose bound on the probability of e^* being a prime is $\geq (\frac{N}{\lg N})/N = \frac{1}{\lg N} = \frac{1}{20\lambda}$.

Now, we describe the reduction. Our proof first applies Lemma A.4, which allows us to only consider adversaries $\mathcal{A}$ that ask queries to *Sign* oracle in the **NHU** game. Moreover, suppose adversary $\mathcal{A}$ queries the random oracle H_p on at most s unique inputs. Without loss of generality, we will assume that all queries to this deterministic oracle are unique and that whenever *Sign* is called on message M , then H_p is automatically called with all unique substrings of M . Suppose an adversary $\mathcal{A}$ can produce a forgery with probability ϵ in the **NHU** game; then we can construct an adversary $\mathcal{B}$ that breaks the RSA assumption (with odd prime e^*) with probability ϵ/s minus a negligible amount as follows.

On input an RSA challenge (N, e^*, y) , $\mathcal{B}$ proceeds as follows:

Setup $\mathcal{B}$ chooses 2λ -bit distinct prime numbers $e_1, e_2, \dots, e_{s-1}$ at random, where all $e_i \neq e^*$. Denote this set of primes as E . Next, $\mathcal{B}$ makes a random guess of $i^* \in [1, s]$ and saves this value for later. Then it sets

$$a := y^{\prod_{e_i \in E} e_i}.$$

Finally, $\mathcal{B}$ give the public key $PK = (N, a)$ to $\mathcal{A}$ and will answer its queries to random oracle H_p interactively as described below.

Queries Proceeding adaptively, $\mathcal{B}$ answers the oracle and sign queries made by $\mathcal{A}$ as follows:

1. $H_p(x)$: When $\mathcal{A}$ queries the random oracle for the j th time, $\mathcal{B}$ responds with e^* if $j = i^*$, with e_j if $j < i^*$ and e_{j-1} otherwise. Recall that we stipulated that each call to H_p was unique. Denote x^* as the input where $H_p(x^*) = e^*$.
2. $Sign(M)$: Let $U = P^0(M)$. If $x^* \in U$, then $\mathcal{B}$ aborts the simulation. Otherwise, $\mathcal{B}$ calls H_p on all elements of U not previously queried to H_p . Let $\mathbf{primes}(U)$ denote the set of primes derived by calling H_p on the strings of U . Then, it computes the signature as $\sigma := y^{\prod_{e_i \in (E - \mathbf{primes}(U))} e_i} \bmod N$ and returns (M, σ) .

Response Eventually, $\mathcal{A}$ outputs a valid message-signature pair (M, σ) , where M is not a derivative of an element returned by $Sign$. If M was not queried to H_p or if $M \neq x^*$, then $\mathcal{B}$ aborts the simulation. Otherwise, let $U = P^0(x^*) - \{x^*\}$ and $\mathbf{primes}(U)$ denote the set of primes derived by calling H_p on the strings of U . It holds that $a^{1/\prod_{e_i \in \mathbf{primes}(U)} e_i} = y^{\prod_{e_i \in E - \mathbf{primes}(U)} e_i} = \sigma^{e^*} \bmod N$. Since $y, \sigma \in \mathbb{Z}_N$ and $\gcd(e^*, \prod_{e_i \in E - \mathbf{primes}(U)} e_i) = 1$ (recall, they are all distinct primes), then $\mathcal{B}$ can apply the efficient algorithm from Lemma 4.3 to obtain a value $z \in \mathbb{Z}_N$ such that $z^{e^*} = y \bmod N$. $\mathcal{B}$ outputs z as the solution to the RSA challenge.

Analysis We now argue that any successful adversary $\mathcal{A}$ against our scheme will have success in the game presented by $\mathcal{B}$. To do this, we first define a sequence of games, where the first game models the real security game and the final game is exactly the view of the adversary when interacting with $\mathcal{B}$. We then show via a series of claims that if $\mathcal{A}$ is successful against Game j , then it will also be successful against Game $j + 1$.

Game 1: The same as Game **NHU**, with the exception that at the beginning of the game $\mathcal{B}$ guesses an index $1 \leq i^* \leq s$ and e^* is the response of the i^* th query to H_p .

Game 2: The same as Game 1, with the exception that $\mathcal{A}$ fails if any output of H_p is repeated.

Game 3: The same as Game 2, with the exception that $\mathcal{A}$ fails if it outputs a valid forgery (M, σ) where M was not queried to H_p .

Game 4: The same as Game 3, with the exception that $\mathcal{A}$ fails if it outputs a valid forgery (M, σ) where $M \neq x^*$.

Notice that Game 4 is exactly the view of the adversary when interacting with $\mathcal{B}$. We complete this argument by linking the probability of $\mathcal{A}$'s success in these games via a series of claims. The only non-negligible probability gap comes between Games 3 and 4, where there is a factor $1/s$ loss.

Define $\mathbf{Adv}_{\mathcal{A}}[\text{Game } x]$ as the advantage of adversary $\mathcal{A}$ in Game x .

Claim 4.8 *If H_p is a truly random function, then*

$$\mathbf{Adv}_{\mathcal{A}}[\text{Game } 1] = \mathbf{Adv}_{\mathcal{A}}[\text{Game } \mathbf{NHU}].$$

Proof. The value e^* was chosen independently at random by the RSA challenger, just as H_p would have done. $\square$

Claim 4.9 *If H_p is a truly random function, then*

$$\mathbf{Adv}_{\mathcal{A}}[\text{Game 2}] = \mathbf{Adv}_{\mathcal{A}}[\text{Game 1}] - \frac{2s^2\lambda}{2^{2\lambda}}.$$

Proof. Consider the probability of a repeat occurring when s 2λ -bit primes are chosen at random. By Theorem 4.4, we know that there are at least $2^{2\lambda}/(2\lambda)$ 2λ -bit primes. Thus, a repeat will occur with probability $< \sum^s s/(2^{2\lambda}/2\lambda) = 2s^2\lambda/2^{2\lambda}$, which is negligible since s must be polynomial in λ . $\square$

Claim 4.10 *If H_p is a truly random function, then*

$$\mathbf{Adv}_{\mathcal{A}}[\text{Game 3}] = \mathbf{Adv}_{\mathcal{A}}[\text{Game 2}] - \frac{2\lambda}{2^{2\lambda}}.$$

Proof. If M was never queried to H_p , then σ can only be a valid forgery if $\mathcal{A}$ guessed the 2λ -bit prime that H_p would respond with on input M . By Theorem 4.4, there are at least $2^{2\lambda}/2\lambda$ such primes and thus the probability of $\mathcal{A}$'s correct guess is at most $2\lambda/2^{2\lambda}$, which is negligible. $\square$

Claim 4.11

$$\mathbf{Adv}_{\mathcal{A}}[\text{Game 4}] = \frac{\mathbf{Adv}_{\mathcal{A}}[\text{Game 3}]}{s}.$$

Proof. At this point in our series of games, we conclude that $\mathcal{A}$ forges on one of the s queries to H_p and that $1 \leq i^* \leq s$ was chosen at random. Thus, the probability that $\mathcal{A}$ forges on the i^* th query is $1/s$. $\square$

This completes our proof. $\square$

4.3 On the Limitations of Using a Generic NIZK Proof of Knowledge Approach

Another general approach that one might be tempted to try is to use an NIZK [8] proof of knowledge system to generate a signature on m' by proving that one knows a signature on some m such that $P(m, m')$ holds. Unfortunately, this approach has the standard drawback of generality in that it requires circuit-based (non black-box) reductions. In particular, the statements to prove in non-interactive zero-knowledge require transforming the circuits of the signature scheme and the quoting predicate into an instance of Hamiltonian circuit or 3-SAT. Even if one were to tailor an NIZK proof of knowledge for these specific statements and therefore avoid costly reductions, another problem emerges with re-quoting. When a quote is re-quoted, then the same process happens for both the original signature scheme circuit, the predicate, *and* the proof system. Aside from the inefficiency, using standard NIZKPoK systems would leak information about the size of the original message and quotes, and therefore would not satisfy our context hiding property⁷.

⁷Using non-interactive CS-proofs [39] in the random oracle model may reduce the size of the proof, but we do not know how to avoid leaking the size of the theorem statement which also violates the context hiding property.

5 A Powers-of-2 Construction for Quoting Substrings

We now provide our main construction for quoting substrings in a text document. It achieves the best time/space efficiency trade-off to our knowledge for this problem. We will have two different types of signatures called Type I and Type II, where a Type I signature can be quoted down to another Type I or Type II signature. A Type II signature cannot be quoted any further, but will be a shorter signature. The quoting algorithm will allow us to quote anything that is a substring of the original message. We point out that the Type I, II signatures of this system conform to the general framework given in Section 2. In particular, we can view a message M as a pair $(t, m) \in \{0, 1\}, \{0, 1\}^*$. The bit t will identify the message as being Type I or Type II (assume $t = 1$ signifies Type I signatures) and m will be the quoted substring. The predicate

$$P(M = (t, m), M' = (t', m')) = \begin{cases} 1 & \text{if } t = 1 \text{ and } m' \text{ is a substring of } m; \\ 0 & \text{otherwise.} \end{cases}$$

The bit t' will indicate whether the new message is Type I or II (i.e., whether the system can quote further.) We note that this description allows an attacker to distinguish between any Type I signature from any Type II signature since the “type bit” of the messages will be different and thus they will technically be two different messages even if the substring components are equal. For this reason we will only need to prove context hiding between messages of Type I or Type II, but not across types. In general, flipping the bit t will not result in a valid signature of a different type on the same core message, because the format will be wrong; however, moving from a Type I to a Type II on the same core message is not considered a forgery since Type II signatures can be legally derived from Type I.

For presentational clarity, we will split the description of our quoting algorithm into two quoting algorithms for quoting to Type I and to Type II signatures; likewise we will split the description of our verification algorithm into two separate verification algorithms, one for each type of signature. The type of signature used or created (i.e., bit t) will be implicit in the description.

Notation: We use notation $m_{i,j}$ to denote the substring of m of length j starting at position i .

Intuition: We begin by giving some intuition. We design Type I signatures that allow re-quoting and Type II signatures that cannot be further quoted, but are ultra-short. For an original message of length n , our signature structure should be able to accommodate starting at any position $1 \leq i \leq n$ and quoting any length $1 \leq \ell \leq (n - i + 1)$ substring.⁸

To (roughly) see how this works for a message of length n , visualize $(n + 1)$ columns with $(\lfloor \lg n \rfloor + 2)$ rows as in Figure 1. The columns correspond to the characters of the message, so if the 14-character message is “abcdefghijklmn” then there are 15 columns, with a character in between each column. The rows correspond to the numbers $\lg n$ down to 0, plus an extra row at the bottom.⁹ Each location in the matrix (except along the bottom-most row) contains one or more out-going arrows. We’ll establish rules for when these arrows exist and where each arrow ends shortly.

A Type II quote will trace a $(\lg n + 1)$ -length path on these arrows through this matrix starting in a row (with outgoing arrows) of the column that begins the quote and ending in the lowest row of the first column after the quote ends. The starting row corresponds to the largest power of two less than or equal to the length of the desired quote. E.g., to quote “bcdef”, start in row 2

⁸Technically, our predicate $P(m, m')$ will take the quote from the first occurrence of substring m' in m , but for the moment imagine that we allowed quoting from anywhere in m .

⁹The lowest row is intentionally not assigned a number. The second lowest row is row 0. We do this so that row i can correspond to a jump of length 2^i .

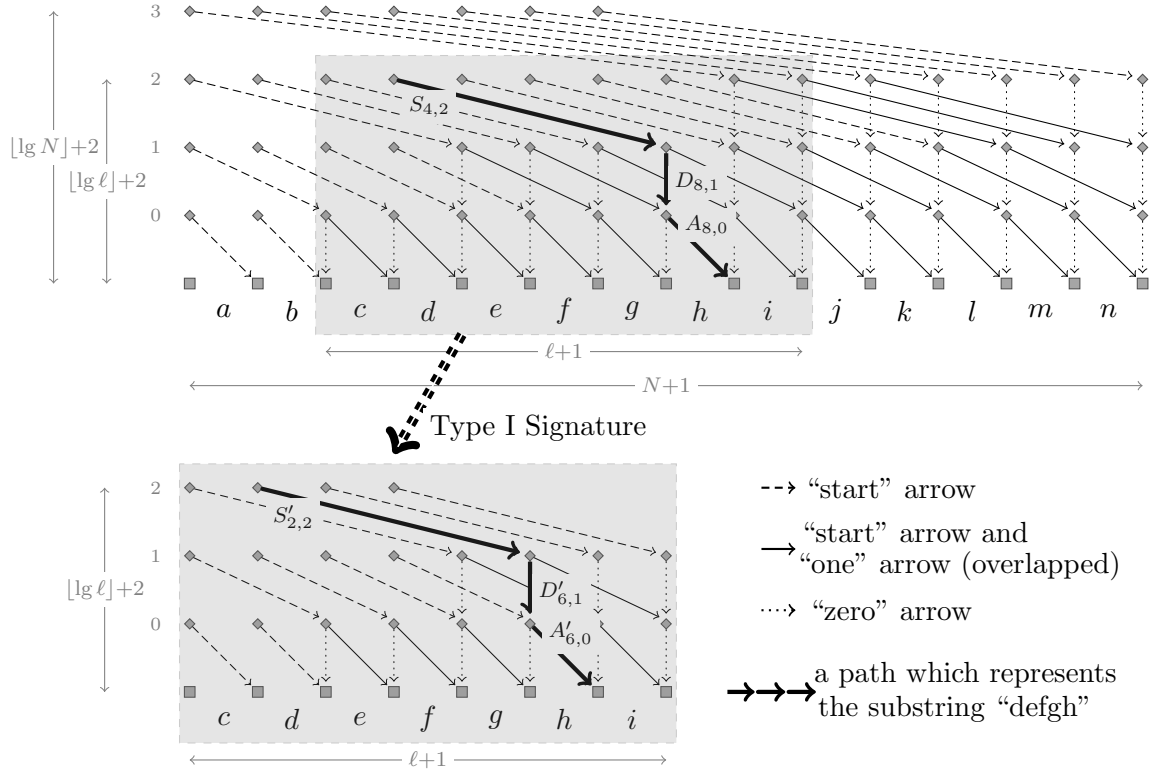


Figure 1: The top diagram represents a signature on “abcdefghijklmn” with length $N = 14$. Each arrow corresponds to some group elements in the construction. Logically, whenever the elements corresponding to an arrow are included in a quoted signature, the characters underneath this arrow are included in the quoted message. The bold path through the top diagram shows how to construct a Type II signature on “defgh”; it is very short, but cannot be re-quoted. The gray box in this figure shows how to construct a Type I signature on “cdefghi” of length $\ell = 7$; it includes all the arrows in the lower figure and can be re-quoted. A technical challenge is to enforce that following the arrows is the only way to form a valid signature. Details are below.

immediately to the left of ‘b’ (because $2^2 = 4$ is the largest power of two less than 5) and end in row 0 immediately to the right of ‘f’. Intuitively, taking an arrow over a character includes it in the quote. A Type II quote on “defgh” is illustrated in Figure 1.

A technical challenge is to make this a $O(\lg n)$ -length path rather than a $O(n)$ -length path. To do this, the key insight is to view the length of any possible quote as the sum of powers of two and to allow arrows that correspond to covering the quote in pieces of size corresponding to one operand of the sum at a time. Each location (i_c, i_r) in the matrix (except the bottom-most row) contains:

- a “start” arrow: an arrow that goes down one row and over 2^{i_r} columns ending in $(i_c + 2^{i_r}, i_r - 1)$, if this end point is in the matrix. This adds all characters from position i_c to $i_c + 2^{i_r} - 1$ to the quoted substring; effectively adding the largest power-of-two-length prefix of the quote characters. This arrow indicates that the quote starts here. These are represented as $S_{i,j}, \widetilde{S}_{i,j}$ pairs in our construction.
- a “one” arrow: operate similarly to start arrows and used to include characters after a start

arrow includes the quote prefix. These are represented as $A_{i,j}, \widetilde{A_{i,j}}$ pairs in our construction.

- a “zero” arrow: an arrow that goes straight down one row ending in $(i_c, i_r - 1)$. This does not add any characters to the quoted substring. These are represented as $D_{i,j}, \widetilde{D_{i,j}}$ pairs in our construction.

A Type II quote always starts with a start arrow and then contains one and zero arrows according to the binary representation of the length of the quote. In our example of original message “abcdefghijklmn”, we have 15 columns and 5 rows. We will logically divide our desired substring of “bcdef” (length $5 = 2^2 + 2^0 = 4 + 1$) into its powers-of-two components “bcde” (length $4 = 2^2$) and “f” (length $1 = 2^0$). To form the Type II quote, we start in row 2 (since $4 = 2^2$) of column 2 (to the left of ‘b’) and take the start arrow ($S_{2,2}$) to row 1 of column 7, take the zero arrow ($D_{7,1}$) to row 0 of column 7, and then take the one arrow ($A_{7,0}$) to the lowest row of column 8. The arrows “pass over” the characters “bcdef”. Figure 1 illustrates this for quote “defgh”.

For a quote of length ℓ , the elements on this $O(\lg \ell)$ -length path of arrows form a very short Type II signature. For Type I signatures, we include all the elements corresponding to all arrows that make connections within the columns corresponding to the quote. We illustrate this in Figure 1. This allows quoting of quotes with a signature size of $O(\ell \lg \ell)$.

It is essential for security that the signature structure and data algorithm enforce that the quoting algorithm be used and not allow an attacker to “splice” together a quote from different parts of the signature. We realize this by adding in random “chaining” variables. In order to cancel these out and get a well formed Type II quote a user must intuitively follow the prescribed procedure (i.e., following the arrows is the only way to form a valid quote.)

The Construction: We now describe our algorithms. While **Sign** is simply a special case of the **SignDerive** algorithm, we will explicitly provide both algorithms here for clarity purposes.

KeyGen(1^λ) : The algorithm selects a bilinear group $\mathbb{G}$ of prime order $p > 2^\lambda$ with generator g . Let L be the maximum message length supported and denote $n = \lfloor \lg(L) \rfloor$. Let $H : \{0, 1\}^* \rightarrow \mathbb{G}$ and $H_s : \{0, 1\}^* \rightarrow \mathbb{G}$ be the description of two hash functions that we model as random oracles. Choose random $z_0, \dots, z_{n-1}, \alpha \in \mathbb{Z}_p$. The secret key is $(z_0, \dots, z_{n-1}, \alpha)$ and the public key is:

$$PK = (H, H_s, g, g^{z_0}, \dots, g^{z_{n-1}}, \mathbf{e}(g, g)^\alpha).$$

Sign($sk, M = (t, m) \in \{0, 1\} \times \Sigma^{\leq L}$) : If $t = 1$, signatures produced by this algorithm are Type I as described below. If $t = 0$, the Type II signature can be obtained by running this algorithm and then running the Quote-Type II algorithm below to obtain a quote on the entire message. The message space is treated as $\ell \leq L$ symbols from alphabet Σ .

Recall: we use notation $m_{i,j}$ to denote the substring of m of length j starting at position i .

For $i = 3$ to $\ell + 1$ and $j = 0$ to $\lfloor \lg(i - 1) - 1 \rfloor$, choose random values $x_{i,j} \in \mathbb{Z}_p$. These will serve as our random “chaining” variables, and they should all “cancel” each other out in our short Type II signatures. By definition, set $x_{i,-1} := 0$ for all $i = 1$ to $\ell + 1$.

A signature is comprised of the following values for $i = 1$ to ℓ and $j = 0$ to $\lfloor \lg(\ell - i + 1) \rfloor$, for randomly chosen values $r_{i,j} \in \mathbb{Z}_p$:

$$\begin{aligned} &[\text{start arrow: start and include power } j] \\ &S_{i,j} = g^\alpha g^{-x_{i+2j,j-1}} H_s(m_{i,2j})^{r_{i,j}} \quad , \quad \widetilde{S_{i,j}} = g^{r_{i,j}} \end{aligned}$$

Together with the following values for $i = 3$ to ℓ and $j = 0$ to $\min(\lfloor \lg(i-1) \rfloor - 1, \lfloor \lg(\ell-i+1) \rfloor)$, for randomly chosen values $r'_{i,j} \in \mathbb{Z}_p$:

[one arrow: include power j and decrease j]

$$A_{i,j} = g^{x_{i,j}} g^{-x_{i+2j,j-1}} H(m_{i,2j})^{r'_{i,j}}, \quad \widetilde{A_{i,j}} = g^{r'_{i,j}}$$

Together with the following values for $i = 3$ to $\ell+1$ and $j = 0$ to $\lfloor \lg(i-1) \rfloor - 1$, for randomly chosen values $r''_{i,j} \in \mathbb{Z}_p$:

[zero arrow: decrease j]

$$D_{i,j} = g^{x_{i,j}} g^{-x_{i,j-1}} g^{z_j r''_{i,j}}, \quad \widetilde{D_{i,j}} = g^{r''_{i,j}}$$

We provide an example of how to form Type II signatures from this construction shortly. To see why our $A_{i,j}$ and $D_{i,j}$ values start at $i = 3$, note that Type II quotes at position i of length $2^0 = 1$ symbol include only the $S_{i,0}$ value, where the $x_{i,0-1}$ term is 0 by definition. Type II quotes at position i of length $2^1 = 2$ symbols include the $S_{i,1}$ value plus an additional $D_{i+2,0}$ term to cancel out the $x_{i+2,0}$ value (leaving only $x_{i+2,-1} = 0$.) Quotes at position i of length $2^1 + 1 = 3$ symbols include the $S_{i,1}$ value plus an additional $A_{i+2,0}$ term to cancel out the $x_{i+2,0}$ value (leaving only $x_{i+3,-1} = 0$.) Since we index strings from position 1, the first position to include an $A_{i,j}$ or $D_{i,j}$ value is $i + 2 = 3$.

SignDerive($pk, \sigma, M = (t, m), M' = (t', m')$) : If $P(M, M') = 0$, output $\perp$. Otherwise, if $t' = 1$, output Quote-Type I(PK, σ, m, m'); if $t' = 0$, output Quote-Type II(PK, σ, m, m'), where these algorithms are defined below.

Quote-Type I(pk, σ, m, m') : The quote algorithm takes a Type I signature and produces another Type I signature that maintains the ability to be quoted again. Intuitively, this operation will simply find a substring m' in m , keep only the components associated with this substring and re-randomize them all (both the $x_{i,j}$ and $r_{i,j}$ terms in every component.)

If m' is not a substring of m , then output $\perp$. Otherwise, let $\ell' = |m'|$. Determine the first index k at which substring m' occurs in m . Parse σ as a collection of $S_{i,j}, \widetilde{S_{i,j}}, A_{i,j}, \widetilde{A_{i,j}}, D_{i,j}, \widetilde{D_{i,j}}$ values, exactly as would come from **Sign** with $\ell = |m|$.

First, we choose re-randomization values (to re-randomize the $x_{i,j}$ terms of σ .) For $i = 2$ to $\ell' + 1$ and $j = 0$ to $\lfloor \lg(i-1) \rfloor - 1$, choose random values $y_{i,j} \in \mathbb{Z}_p$. Set $y_{i,-1} := 0$ for all $i = 1$ to $\ell' + 1$. Later, we will choose $t_{i,j}$ values to re-randomize the $r_{i,j}$ terms of σ .

The quote signature σ' is comprised of the following values:

For $i = 1$ to ℓ' and $j = 0$ to $\lfloor \lg(\ell' - i + 1) \rfloor$, for randomly chosen $t_{i,j} \in \mathbb{Z}_p$:

$$S'_{i,j} = S_{i+k-1,j} \cdot g^{-y_{i+2j,j-1}} H_s(m_{i+k-1,2j})^{t_{i,j}}, \quad \widetilde{S'_{i,j}} = \widetilde{S_{i+k-1,j}} \cdot g^{t_{i,j}}$$

Together with the following values for $i = 3$ to ℓ' and $j = 0$ to $\min(\lfloor \lg(i-1) \rfloor - 1, \lfloor \lg(\ell' - i + 1) \rfloor)$, for randomly chosen $t'_{i,j} \in \mathbb{Z}_p$:

$$A'_{i,j} = A_{i+k-1,j} \cdot g^{y_{i,j}} g^{-y_{i+2j,j-1}} H(m_{i+k-1,2j})^{t'_{i,j}}, \quad \widetilde{A'_{i,j}} = \widetilde{A_{i+k-1,j}} \cdot g^{t'_{i,j}}$$

Together with the following values for $i = 3$ to $\ell' + 1$ and $j = 0$ to $\lfloor \lg(i - 1) - 1 \rfloor$, for randomly chosen $t''_{i,j} \in \mathbb{Z}_p$:

$$D'_{i,j} = D_{i+k-1,j} \cdot g^{y_{i,j}} g^{-y_{i,j-1}} g^{z_j t''_{i,j}}, \quad \widetilde{D'_{i,j}} = \widetilde{D_{i+k-1,j}} \cdot g^{t''_{i,j}}$$

Quote-Type II(pk, σ, m, m') : The quote algorithm takes a Type I signature and produces a Type II signature. If $P(m, m') \neq 1$, then output $\perp$.

A quote is computed from one start value and logarithmically many subsequent pieces depending on the bits of $|m'|$. All signature pieces must be re-randomized to prevent content-hiding attacks.

Consider the length ℓ' written as a binary string. Let β' be the largest index of $\ell' = |m'|$ that is set to 1, where *we start counting with zero as the least significant bit*. That is, set $\beta' = \lfloor \lg(\ell') \rfloor$. Select random values $v, v_{\beta'-1}, \dots, v_0 \in \mathbb{Z}_p$. Set the start position as $B := S_{k,\beta'}$ and $k' := k + 2^{\beta'}$. Then, from $j = \beta' - 1$ down to 0, proceed as follows:

- If the j th bit of ℓ' is 1, set $B := B \cdot A_{k',j} \cdot H(m_{k',2^j})^{v_j}$, set $k' := k' + 2^j$, and $Z_j := \widetilde{A_{k',j}} \cdot g^{v_j}$;
- If the j th bit of ℓ' is 0, set $B := B \cdot D_{k',j} \cdot g^{z_j v_j}$ and $Z_j := \widetilde{D_{k',j}} \cdot g^{v_j}$.

To end, re-randomize as $B := B \cdot H_s(m_{k,2^\beta})^v$ and $\widetilde{S} := \widetilde{S_{k,\beta}} \cdot g^v$; output the quote as

$$\sigma' = (B, \widetilde{S}, Z_{\beta-1}, \dots, Z_0)$$

Verify($pk, M = (t, m), \sigma$) : If $t = 1$, output Verify-Type I(pk, m, σ). Otherwise, output Verify-Type II(pk, m, σ), where these algorithms are defined immediately below.

Verify-Type I(pk, m, σ) : Parse σ as the set of $S_{i,j}, \widetilde{S_{i,j}}, A_{i,j}, \widetilde{A_{i,j}}, D_{i,j}, \widetilde{D_{i,j}}$. Let $\ell = |m|$.

Let $X_{i,j}$ denote $e(g, g)^{x_{i,j}}$. We can compute these values as follows. The value $X_{i,-1} = 1$, since for all $i = 1$ to $\ell + 1$, $x_{i,-1} = 0$. For $i = 3$ to $\ell + 1$ and $j = 0$ to $\lfloor \lg(i - 1) - 1 \rfloor$, we compute $X_{i,j}$ in the following manner: Let $I = i - 2^{j+1}$ and $J = j + 1$. Next, compute $X_{i,j} = (e(g, g)^\alpha \cdot e(H_s(m_{I,2^J}), \widetilde{S_{I,J}})) / e(S_{I,J}, g)$. The verification accepts if and only if all of the following hold:

- for $i = 3$ to ℓ and $j = 0$ to $\min(\lfloor \lg(i - 1) - 1 \rfloor, \lfloor \lg(\ell - i + 1) \rfloor)$,

$$e(A_{i,j}, g) = X_{i,j} / X_{i+2^j, j-1} \cdot e(H(m_{i,2^j}), \widetilde{A_{i,j}})$$

- and for $i = 3$ to $\ell + 1$ and $j = 0$ to $\lfloor \lg(i - 1) - 1 \rfloor$, $e(D_{i,j}, g) = X_{i,j} / X_{i,j-1} \cdot e(g^{z_j}, \widetilde{D_{i,j}})$.

Verify-Type II(pk, m, σ) : We give the verification algorithm for Type II signatures. Parse σ as $(B, \widetilde{S}, Z_{\beta-1}, \dots, Z_0)$. Let $\ell = |m|$ and β be the index of the highest bit of ℓ that is set to 1. If σ does not include exactly β Z_i values, reject. Set $C := 1$ and $k = 1$. From $j = \beta - 1$ down to 0, proceed as follows:

- If the j th bit of ℓ is 1, set $C := C \cdot e(H(m_{k,2^j}), Z_j)$ and $k := k + 2^j$;
- If the j th bit of ℓ is 0, set $C := C \cdot e(g^{z_j}, Z_j)$.

Accept if and only if $e(B, g) = e(g, g)^\alpha \cdot e(H_s(m_{1,2^\beta}), \widetilde{S}) \cdot C$.

Theorem 5.1 (Security under CDH) *If the CDH assumption holds in $\mathbb{G}$, then the above quotable signature scheme is selectively quote unforgeable and context-hiding in the random oracle model.*

Efficiency Discussion This construction presents the best known balance between time and space complexity. The quotable (Type I) signatures require $O(\ell \lg \ell)$ elements in $\mathbb{G}$ for a message of length ℓ . The group elements in both types of signatures are elements of $\mathbb{G}$, and not the target group $\mathbb{G}_T$. Typically, elements of the base group are significantly smaller than elements of the target group. Computing quotes requires $O(\ell \lg \ell)$ modular exponentiations for a quote of length ℓ for re-randomization. Similarly, verification also requires $O(\ell \lg \ell)$ pairings.

The non-quotable (Type II) signatures require only $O(\lg \ell)$ elements in $\mathbb{G}$. Computing quotes is very efficient as it requires only $O(\lg \ell)$ modular exponentiations for a quote of length ℓ for re-randomization. Similarly, verification requires only $O(\lg \ell)$ pairings.

Removing the Random Oracle and Obtaining Full Security There are a few different options for adapting the above construction to the standard model. We observe that our signatures share many properties with the private keys of hierarchical identity-based encryption (HIBE) schemes. To remove the random oracle, while remaining under a selective definition, we can use the Boneh-Boyen techniques [9] to instantiate $H(m) = g^m h$, where $h \in \mathbb{G}$ is added to the public key and there is a method for mapping the message space to $\mathbb{Z}_p$. Similarly, we can remove the random oracle by instantiating H with the Waters hash [55] and applying his proof techniques. This can be viewed as a full security construction with a reduction to the concrete security parameter by roughly a factor of $(1/O(q))^{\lg \ell}$, where q is the number of signing queries and ℓ is the length of the quote. A direction for achieving full security is to use the recent “Dual System” techniques introduced by Waters [56]. One obstacle in adapting the Waters system is that it contains “tags” in the private key structure, which would likely make our re-randomization step difficult for our context hiding property. Lewko and Waters [37] recently removed the tags, which may make their techniques and construction more suitable for our application. One drawback in using their HIBE techniques to construct signatures is that even the signatures resulting from their construction require (slightly non-standard) *decisional* complexity assumptions. Thus, it is unknown how to balance time/space efficiently while achieving full security in the standard model from a simple computational assumption such as CDH.

5.1 Security Analysis

We now provide a proof of Theorem 5.1 by showing the following lemmas.

Lemma 5.2 (Strong Context-Hiding) *The Section 5 quotable signature scheme is strongly context-hiding.*

Proof. Given any two challenge messages $M = (t, m), M' = (t', m')$ such that $P(M, M') = 1$, we claim that whether $t' = 1$ or 0, **SignDerive**(pk, σ, M', M) has an identical distribution to that of **Sign**(sk, M), which implies that the two distributions are statistically close.

$$\begin{aligned} & \{(SK, \sigma \leftarrow \mathbf{Sign}(SK, M), \mathbf{Sign}(SK, M'))\}_{SK, M, M'} \\ & \{(SK, \sigma \leftarrow \mathbf{Sign}(SK, M), \mathbf{SignDerive}(PK, \sigma, M, M'))\}_{SK, M, M'} \end{aligned}$$

Let ℓ, ℓ' denote $|m|$ and $|m'|$ respectively. Let $\Gamma = \min(\lfloor \lg(i-1) - 1 \rfloor, \lfloor \lg(\ell - i + 1) \rfloor)$. **Sign**(SK, M) is composed of the following values:

$$\begin{aligned} S_{i,j} &= g^\alpha g^{-x_{i+2j,j-1}} H_s(m_{i,2j})^{r_{i,j}}, & \widetilde{S}_{i,j} &= g^{r_{i,j}}, & \text{for } i = 1 \text{ to } \ell \text{ and } j = 0 \text{ to } \lfloor \lg(\ell - i + 1) \rfloor \\ A_{i,j} &= g^{x_{i,j}} g^{-x_{i+2j,j-1}} H(m_{i,2j})^{r'_{i,j}}, & \widetilde{A}_{i,j} &= g^{r'_{i,j}}, & \text{for } i = 3 \text{ to } \ell \text{ and } j = 0 \text{ to } \Gamma \\ D_{i,j} &= g^{x_{i,j}} g^{-x_{i,j-1}} g^{z_j r''_{i,j}}, & \widetilde{D}_{i,j} &= g^{r''_{i,j}}, & \text{for } i = 3 \text{ to } \ell + 1 \text{ and } j = 0 \text{ to } \lfloor \lg(i-1) - 1 \rfloor \end{aligned}$$

for randomly chosen $r_{i,j}, r'_{i,j}, r''_{i,j}, x_{i,j} \in \mathbb{Z}_p$.

Case where $t' = 1$ (Type I Signatures). Let $\Gamma' = \min(\lfloor \lg(i-1) - 1 \rfloor, \lfloor \lg(\ell' - i + 1) \rfloor)$. When $t' = 1$, **Sign**(SK, M') is composed of the following values:

$$\begin{aligned} S''_{i,j} &= g^\alpha g^{-x'_{i+2j,j-1}} H_s(m'_{i,2j})^{v_{i,j}}, & \widetilde{S''_{i,j}} &= g^{v_{i,j}}, & \text{for } i = 1 \text{ to } \ell' \text{ and } j = 0 \text{ to } \lfloor \lg(\ell' - i + 1) \rfloor \\ A''_{i,j} &= g^{x'_{i,j}} g^{-x'_{i+2j,j-1}} H(m'_{i,2j})^{v'_{i,j}}, & \widetilde{A''_{i,j}} &= g^{v'_{i,j}}, & \text{for } i = 3 \text{ to } \ell' \text{ and } j = 0 \text{ to } \Gamma' \\ D''_{i,j} &= g^{x'_{i,j}} g^{-x'_{i,j-1}} g^{z_j v''_{i,j}}, & \widetilde{D''_{i,j}} &= g^{v''_{i,j}}, & \text{for } i = 3 \text{ to } \ell' + 1 \text{ and } j = 0 \text{ to } \lfloor \lg(i-1) - 1 \rfloor \end{aligned}$$

for randomly chosen $v_{i,j}, v'_{i,j}, v''_{i,j}, x'_{i,j} \in \mathbb{Z}_p$.

And **SignDerive**(PK, σ, M, M') is Quote-Type I(PK, σ, m, m'), which is comprised of the following:

$$\begin{aligned} S'_{i,j} &= g^\alpha g^{-w_{i+2j,j-1}} H_s(m'_{i,2j})^{r_{I,j}+t_{i,j}}, & \widetilde{S'_{i,j}} &= g^{r_{I,j}+t_{i,j}}, & \text{for } i = 1 \text{ to } \ell' \text{ and } j = 0 \text{ to } \lfloor \lg(\ell' - i + 1) \rfloor \\ A'_{i,j} &= g^{w_{i,j}} g^{-w_{i+2j,j-1}} H(m'_{i,2j})^{r'_{I,j}+t'_{i,j}}, & \widetilde{A'_{i,j}} &= g^{r'_{I,j}+t'_{i,j}}, & \text{for } i = 3 \text{ to } \ell' \text{ and } j = 0 \text{ to } \Gamma' \\ D'_{i,j} &= g^{w_{i,j}} g^{-w_{i,j-1}} g^{z_j(r''_{I,j}+t''_{i,j})}, & \widetilde{D'_{i,j}} &= g^{r''_{I,j}+t''_{i,j}}, & \text{for } i = 3 \text{ to } \ell' + 1 \text{ and } j = 0 \text{ to } \lfloor \lg(i-1) - 1 \rfloor \end{aligned}$$

for randomly chosen $t_{i,j}, t'_{i,j}, t''_{i,j}, y_{i,j} \in \mathbb{Z}_p$, where m' occurs at position k as a substring of m , $I = i + k - 1$ and $w_{i,j} = x_{I,j} + y_{i,j}$.

Since all exponents have been independently re-randomized, one can see by inspection that **SignDerive**(pk, σ, M', M) has identical distribution as that of **Sign**(sk, M').

Case where $t' = 0$ (Type II Signatures). Parse $m' = m'_\beta m'_{\beta-1} \dots m'_0$ where m'_j is of length 2^j or a null string where $\beta = \lfloor \lg(\ell') \rfloor$. ℓ'_i denotes i -th bit of ℓ' when we start counting with zero as the least significant bit. m' occurs at position k of m . **Sign**(SK, M') = $(B, \tilde{S}, Z_{\beta-1}, \dots, Z_0)$ is the following, for random $u, u_i \in \mathbb{Z}_p$:

$$\begin{aligned} B &= g^\alpha \cdot H_s(m'_\beta)^u \prod_{j < \beta, \ell'_j=1} H(m'_j)^{u_j} \prod_{j' < \beta, \ell'_{j'}=0} g^{z_{j'} u_{j'}} \\ \tilde{S} &= g^u, \quad Z_j = g^{u_j} \end{aligned}$$

Let each m'_j start at position s_j in m' . **SignDerive**(PK, σ, M, M') = Quote-Type II(PK, σ, m, m') is $(B', \tilde{S}', Z'_{\beta-1}, \dots, Z'_0)$ such that

$$\begin{aligned} B' &= g^\alpha \cdot H_s(m'_\beta)^{r_{k,\beta}+v} \prod_{j < \beta, \ell'_j=1} H(m'_j)^{r'_{k+s_j-1,j}+v_j} \prod_{j' < \beta, \ell'_{j'}=0} g^{z_{j'}(r''_{k+s_{j'}-1,j'}+v_{j'})} \\ \tilde{S}' &= g^{r_{k,\beta}+v}, \quad Z'_j = g^{r''_{k+s_j-1,j}+v_j} \end{aligned}$$

for randomly chosen $v, v_j \in \mathbb{Z}_p$. Since all exponents have been independently re-randomized, one can see by inspection that **SignDerive**(PK, σ, M, M') has identical distribution as that of **Sign**(sk, M').

Thus, the our powers-of-2 construction is strongly context-hiding. $\square$

Lemma 5.3 (Unforgeability) *If the CDH assumption holds in $\mathbb{G}$, then the Section 5 quotable signature scheme is selectively unforgeable in the **Unforg** game in the random oracle model.*

Proof. (Sketch) We first apply Lemma A.4, which allows us to only consider adversaries $\mathcal{A}$ that asks queries to *Sign* oracle in the simpler **NHU** game.

Suppose an adversary $\mathcal{A}$ can produce a forgery with probability ϵ in the selective **NHU** unforgeability game; then we can construct an adversary $\mathcal{B}$ that breaks the CDH assumption with probability ϵ plus a negligible amount.

We are now ready to describe $\mathcal{B}$ which solves the CDH problem. On input the CDH challenge (g, g^a, g^b) , $\mathcal{B}$ begins to run $\mathcal{A}$ and proceeds as follows:

Selective Disclosure $\mathcal{A}$ first announces the message M^* on which he will forge.

Setup Let L be the maximum size of any message and let $n = \lfloor \lg(L) \rfloor$. Let $M^* = (t^*, m^*)$ and $\ell^* = |m^*|$ and let β be the highest bit of ℓ^* set to 1 (numbering the least significant bit as zero). Set $\mathbf{e}(g, g)^\alpha := \mathbf{e}(g^a, g^b)$, which implicitly sets the secret key $\alpha = ab$.

For $i = 0$ to $n - 1$, choose a random $v_i \in \mathbb{Z}_p$ and set

$$g^{z_i} = \begin{cases} g^{bv_i} & \text{if the } i\text{th bit of } \ell^* \text{ is 1;} \\ g^{v_i} & \text{otherwise.} \end{cases}$$

Finally, $\mathcal{B}$ give the public key $PK = (g, g^{z_0}, \dots, g^{z_{n-1}}, \mathbf{e}(g, g)^\alpha)$ to $\mathcal{A}$ and will answer its queries to random oracles H and H_s interactively as described below.

Random Oracle Queries Proceeding adaptively, $\mathcal{A}$ may make any of the following queries which $\mathcal{B}$ will answer as follows:

1. $H(x)$: The random oracle is answered as follows. If the query has been made before, return the same response as before. Otherwise, imagine dividing up m^* into a sequence of segments whose lengths are decreasing powers of two; that is, the first segments would be of length 2^β where β is the largest power of two less than ℓ^* , the second segment would contain the next largest power of two, etc. Let $m_{(j)}^*$ denote the segment of m^* corresponding to power j . If no such segment exists, let $m_{(j)}^* = \perp$. Select a random $\gamma \in \mathbb{Z}_p$ and return the response as:

$$H(x) = \begin{cases} g^\gamma & \text{if } |x| = 2^j \text{ and } j < \beta \text{ and } m_{(j)}^* = x \\ & (x \text{ is on the selective path}); \\ g^{b\gamma} & \text{otherwise} \\ & (x \text{ is not on the selective path}). \end{cases}$$

Note that $H(m_{(j)}^*)$ is set according to the first method for all segments of m^* *except* the first segment $m_{(\beta)}^*$.

2. $H_s(x)$: The random oracle is answered as follows. If the query has been made before, return the same response as before. Select a random $\delta \in \mathbb{Z}_p$ and return the response as:

$$H_s(x) = \begin{cases} g^\delta & \text{if } |x| = 2^\beta \text{ and } m_{(\beta)}^* = x; \\ g^{b\delta} & \text{otherwise.} \end{cases}$$

Note that $H_s(m_{(j)}^*)$ is set according to the first method *only* for the first segment of m^* .

Signature and Quote Queries

Sign (M): Let $M = (t, m)$ and $\ell = |m|$. Recall that β^* is highest bit of ℓ^* set to 1 and that we are counting up from zero as the least significant bit.

We describe how to create signatures.

1. When $t = 1$ and m^* is not a substring of m (Type I Signature Generation):

Here $m_{i,j}$ denotes the substring m of length j starting at position i . It will help us to first establish the variables $X_{i,j}$, which will be set to 1 if on the selective forgery path and 0 otherwise. We give a set of “rules” defining terms and make a few observations. Then we describe how the reduction algorithm creates the signatures.

Rules.

For $i = 1$ up to $\ell + 1$,

For $j = \lfloor \lg(\ell - i + 1) \rfloor$ down to -1 ,

- (a) If $j + 1 = \beta^*$ and $m_{i-2^{j+1}, 2^{j+1}} = m_{(j+1)}^*$, then set $X_{i,j} = 1$.
- (b) Else, if $j + 1 < \beta^*$ and $(j + 1)$ th bit of ℓ^* is 1 and $m_{i-2^{j+1}, 2^{j+1}} = m_{(j+1)}^*$ and $X_{i-2^{j+1}, j+1} = 1$, then set $X_{i,j} = 1$.
- (c) Else if $j + 1 < \beta^*$ and $(j + 1)$ th bit of ℓ^* is 0 and $X_{i, j+1} = 1$, then set $X_{i,j} = 1$.
- (d) Else set $X_{i,j} = 0$.

Observations. Before we show how $\mathcal{B}$ will simulate the signatures, we make a set of useful observations.

- (a) For all i and $j \geq \beta^*$, $X_{i,j} = 0$.
- (b) For all i , $X_{i,-1} = 0$. Otherwise, $m_{i-\ell^*, \ell^*} = m^*$.
- (c) For all i, j , if $X_{i,j} = 1$ and $X_{i, j-1} = 0$, then the j th bit of ℓ^* is 1. If the j th bit were 0, then $X_{i, j-1}$ would have been set to 1 by Rule 1c.
- (d) For all i, j , if $X_{i,j} = 0$ and $X_{i, j-1} = 1$, then the j th bit of ℓ^* is 1. If the j th bit were 0, then the only way to set $X_{i, j-1}$ to 1 would be by Rule 1c, however, $X_{i,j} = 0$ so Rule 1c does not apply.
- (e) For all i, j , if $X_{i,j} = 1$ and $X_{i+2^j, j-1} = 0$, then $H(m_{i, 2^j}) = g^{b\gamma}$ for some known $\gamma \in \mathbb{Z}_p$. Otherwise, $X_{i+2^j, j-1}$ would have been set by Rule 1b to be 1.
- (f) For all i, j , if $X_{i,j} = 0$ and $X_{i+2^j, j-1} = 1$, then $H(m_{i, 2^j}) = g^{b\gamma}$ for some known $\gamma \in \mathbb{Z}_p$. If $X_{i+2^j, j-1} = 1$ and $X_{i,j} = 0$, then $X_{i+2^j, j-1}$ was set to be 1 either by Rule 1a or Rule 1c. If it were Rule 1a, then $j = \beta^*$ and it follows from the programming of the random oracle that $H(m_{i, 2^j}) = g^{b\gamma}$. If it were Rule 1c, then the j th bit of ℓ^* is 0, meaning $m_{(j)}$ cannot be on the selective path and therefore again $H(m_{i, 2^j}) = g^{b\gamma}$.
- (g) For all i, j , if $X_{i+2^j, j-1} = 0$, then $H_s(m_{i, 2^j}) = g^{b\delta}$ for some known $\delta \in \mathbb{Z}_p$. If $j \neq \beta^*$, this follows immediately from the programming of the random oracle. Otherwise, if $j = \beta^*$, then the only way for $X_{i+2^j, j-1} = 0$ would be if $m_{(\beta)} \neq m_{(\beta)}^*$ by Rule 1a. Thus, it also follows that $H_s(m_{i, 2^j}) = g^{b\delta}$.

Signature Components. Next, for $i = 1$ to $\ell + 1$ and $j = 0$ to $\lfloor \lg(\ell - i + 1) \rfloor$, choose a random $x'_{i,j} \in \mathbb{Z}_p$ and logically set $x_{i,j} := x'_{i,j} + X_{i,j} \cdot (ab)$. For $i = 1$ to $\ell + 1$, set $x_{i,-1} := 0$ (as consistent with Observation 1b.)

A signature is comprised of the following values:

Start. For $i = 1$ to ℓ and $j = 0$ to $\lfloor \lg(\ell - i + 1) \rfloor$:

- (a) If $X_{i+2j,j-1} = 0$, then it follows by Observation 1g that $H_s(m_{i,2j}) = g^{b\delta}$ for some known $\delta \in \mathbb{Z}_p$, so choose random $s_{i,j} \in \mathbb{Z}_p$, implicitly set $r_{i,j} := -a/\delta + s_{i,j}$ and set

$$\begin{aligned} S_{i,j} &= g^{-x_{i+2j,j-1}} g^{b\delta s_{i,j}} \\ &= g^\alpha g^{-x_{i+2j,j-1}} H_s(m_{i,2j})^{r_{i,j}} \\ \widetilde{S_{i,j}} &= g^{-a/\delta + s_{i,j}} = g^{r_{i,j}} \end{aligned}$$

- (b) Else $X_{i+2j,j-1} = 1$, so choose random $r_{i,j} \in \mathbb{Z}_p$ and with $x_{i+2j,j-1} := x'_{i+2j,j-1} + ab$ set

$$\begin{aligned} S_{i,j} &= g^{-x'_{i+2j,j-1}} H_s(m_{i,2j})^{r_{i,j}} \\ &= g^\alpha g^{-x_{i+2j,j-1}} H_s(m_{i,2j})^{r_{i,j}} \\ \widetilde{S_{i,j}} &= g^{r_{i,j}} \end{aligned}$$

Across. Together with the following values for $i = 3$ to ℓ and $j = 0$ to $\min(\lfloor \lg(i-1) - 1 \rfloor, \lfloor \lg(\ell - i + 1) \rfloor)$:

- (a) If $X_{i,j} = 1$ and $X_{i+2j,j-1} = 1$, choose random $r'_{i,j} \in \mathbb{Z}_p$ with implicitly set $x_{i,j} = x'_{i,j} + ab$ and $x_{i+2j,j-1} = x'_{i+2j,j-1} + ab$ and set

$$\begin{aligned} A_{i,j} &= g^{x'_{i,j}} g^{-x'_{i+2j,j-1}} H(m_{i,2j})^{r'_{i,j}} \\ &= g^{x_{i,j}} g^{-x_{i+2j,j-1}} H(m_{i,2j})^{r'_{i,j}} \\ \widetilde{A_{i,j}} &= g^{r'_{i,j}} \end{aligned}$$

- (b) Else, if $X_{i,j} = 1$ and $X_{i+2j,j-1} = 0$, then $H(m_{i,2j}) = g^{b\gamma}$ for some known $\gamma \in \mathbb{Z}_p$ by Observation 1e. Choose random $s'_{i,j} \in \mathbb{Z}_p$ with implicitly set $x_{i,j} = x'_{i,j} + ab$, $x_{i+2j,j-1} = x'_{i+2j,j-1}$ and $r'_{i,j} := -a/\gamma + s'_{i,j}$ and set

$$\begin{aligned} A_{i,j} &= g^{x'_{i,j}} g^{-x_{i+2j,j-1}} g^{b\gamma s'_{i,j}} \\ &= g^{x_{i,j}} g^{-x_{i+2j,j-1}} H(m_{i,2j})^{r'_{i,j}} \\ \widetilde{A_{i,j}} &= g^{r'_{i,j}} \end{aligned}$$

- (c) Else, if $X_{i,j} = 0$ and $X_{i+2j,j-1} = 1$, then $H(m_{i,2j}) = g^{b\gamma}$ for some known $\gamma \in \mathbb{Z}_p$ by Observation 1f. Choose random $s'_{i,j} \in \mathbb{Z}_p$ with implicitly set $x_{i,j} = x'_{i,j}$, $x_{i+2j,j-1} = x'_{i+2j,j-1} + ab$ and $r'_{i,j} := a/\gamma + s'_{i,j}$ and set

$$\begin{aligned} A_{i,j} &= g^{x_{i,j}} g^{-x'_{i+2j,j-1}} g^{b\gamma s'_{i,j}} \\ &= g^{x_{i,j}} g^{-x_{i+2j,j-1}} H(m_{i,2j})^{r'_{i,j}} \\ \widetilde{A_{i,j}} &= g^{r'_{i,j}} \end{aligned}$$

- (d) Else, $X_{i,j} = 0$ and $X_{i+2j,j-1} = 0$, so choose random $r'_{i,j} \in \mathbb{Z}_p$ and set

$$A_{i,j} = g^{x_{i,j}} g^{-x_{i+2j,j-1}} H(m_{i,2j})^{r'_{i,j}}, \quad \widetilde{A_{i,j}} = g^{r'_{i,j}}$$

Down. Together with the following values for $i = 3$ to $\ell + 1$ and $j = 0$ to $\lfloor \lg(i-1) - 1 \rfloor$:

- (a) If $X_{i,j} = 1$ and $X_{i,j-1} = 1$, choose random $r''_{i,j} \in \mathbb{Z}_p$ with implicitly set $x_{i,j} = x'_{i,j} + ab$ and $x_{i,j-1} = x'_{i,j-1} + ab$ and set

$$\begin{aligned} D_{i,j} &= g^{x'_{i,j}} g^{-x'_{i,j-1}} g^{z_j r''_{i,j}} = g^{x_{i,j}} g^{-x_{i,j-1}} g^{z_j r''_{i,j}} \\ \widetilde{D_{i,j}} &= g^{r''_{i,j}} \end{aligned}$$

- (b) Else, if $X_{i,j} = 1$ and $X_{i,j-1} = 0$, then the j th bit of ℓ^* is 1 by Observation 1c. Thus $z_j = bv_j$, so choose random $s''_{i,j} \in \mathbb{Z}_p$ with implicitly set $x_{i,j} = x'_{i,j} + ab$, $x_{i,j-1} = x'_{i,j-1}$ and $r''_{i,j} := -a/v_j + s''_{i,j}$ and set

$$\begin{aligned} D_{i,j} &= g^{x'_{i,j}} g^{-x_{i,j-1}} g^{bv_j s''_{i,j}} = g^{x_{i,j}} g^{-x_{i,j-1}} g^{z_j r''_{i,j}} \\ \widetilde{D_{i,j}} &= g^{-a/v_j + s''_{i,j}} = g^{r''_{i,j}} \end{aligned}$$

- (c) Else, if $X_{i,j} = 0$ and $X_{i,j-1} = 1$, then the j th bit of ℓ^* is 1 by Observation 1d. Thus $z_j = bv_j$, so choose random $s''_{i,j} \in \mathbb{Z}_p$ with implicitly set $x_{i,j} = x'_{i,j}$, $x_{i,j-1} = x'_{i,j-1} + ab$ and $r''_{i,j} := a/v_j + s''_{i,j}$ and set

$$\begin{aligned} D_{i,j} &= g^{x'_{i,j}} g^{-x_{i,j-1}} g^{bv_j s''_{i,j}} = g^{x_{i,j}} g^{-x_{i,j-1}} g^{z_j r''_{i,j}} \\ \widetilde{D_{i,j}} &= g^{a/v_j + s''_{i,j}} = g^{r''_{i,j}} \end{aligned}$$

- (d) Else, $X_{i,j} = 0$ and $X_{i,j-1} = 0$, so choose random $r''_{i,j} \in \mathbb{Z}_p$ and set

$$D_{i,j} = g^{x_{i,j}} g^{-x_{i,j-1}} g^{z_j r''_{i,j}}, \quad \widetilde{D_{i,j}} = g^{r''_{i,j}}$$

2. When $t = 0$ and $m \neq m^*$ (Type II Signature Generation):

Let $\ell = |m|$, and $\beta = \lfloor \lg(\ell) \rfloor$. ℓ_i^* denotes i -th bit of ℓ^* when we start counting with zero as the least significant bit, and ℓ_i denotes i -th bit of ℓ .

Parse m^* as $m_{\beta^*}^* m_{\beta^*-1}^* \dots m_0^*$ where m_i^* is a string of length 2^i or a null string. m_i is of length 2^i if $\ell_i = 0$, and is null otherwise. Similarly, parse m as $m_{\beta} m_{\beta-1} \dots m_0$.

$\mathcal{B}$ constructs $(B, \tilde{S}, Z_{\beta-1}, \dots, Z_0)$ in the following way:

- If $m_{\beta} \neq m_{\beta^*}^*$, then $H_s(m_{\beta}) = g^{b\delta}$ for a δ which is known to $\mathcal{B}$.
 - (a) $\mathcal{B}$ sets $\tilde{S} := g^{-a/\delta+r}$ for a randomly chosen r and $B := g^{b\delta r}$.
 - (b) For $j = \beta - 1$ down to 0, $Z_j := g^{r_j}$ for a randomly chosen r_j , and
 - If $\ell_j = 1$, then $B := B \cdot H(m_j)^{r_j}$.
 - If $\ell_j = 0$, then $B := B \cdot g^{z_j r_j}$.
- Otherwise, if $\beta = \beta^*$ and $m_{\beta} = m_{\beta^*}^*$, there exists $j_s < \beta$ such that
 - $\ell_{j_s} \neq \ell_{j_s}^*$, or
 - $\ell_{j_s} = \ell_{j_s}^* = 1$ and $H(m_{j_s}) \neq H(m_{j_s}^*)$.

so $\mathcal{B}$ can construct a signature $(B, \tilde{S}, Z_{\beta-1}, \dots, Z_0)$ in the following way.

- (a) $\mathcal{B}$ sets $\tilde{S} := g^{r_c}$ for a randomly chosen r_c and $B := g^{\delta r_c}$.
- (b) For $j = \beta - 1$ down to $j_s + 1$ and $j = j_s - 1$ to 0, $Z_j := g^{r_j}$ for randomly chosen r_j , and
 - If $\ell_j = 1$, then $B := B \cdot H(m_j)^{r_j}$.
 - If $\ell_j = 0$, then $B := B \cdot g^{z_j r_j}$.
- (c) For $j = j_s$,

- If $\ell_j = 1$, whether $\ell_j^* = 0$ or not, $\mathcal{B}$ knows γ such that $H(m_j) = g^{b\gamma}$. $\mathcal{B}$ sets $Z_j = g^{-a/\gamma+r_j}$ for a randomly chosen r_j , and $B := B \cdot g^{b\gamma r_j}$.
- If $\ell_j = 0$ and $\ell_j^* = 1$, then $\mathcal{B}$ knows v such that $g^{z_j} = g^{bv}$. $\mathcal{B}$ sets $Z_j = g^{-a/v+r_j}$ for a randomly chosen r_j , and $B := B \cdot g^{bvr_j}$.

$\mathcal{B}$ returns $(B, \tilde{S}, Z_{\beta-1}, \dots, Z_0)$.

Response Eventually, $\mathcal{A}$ outputs a valid signature σ^* on $M^* = (t^*, m^*)$. Recall that $\ell^* = |m^*|$ and $\beta = \lfloor \lg(\ell^*) \rfloor$. Here ℓ_i^* denotes i -th bit of ℓ^* when we start counting with zero as the least significant bit. Parse m^* as $m_\beta^* m_{\beta-1}^* \dots m_0^*$ where m_i^* is a string of length 2^i (when $\ell_i^* = 1$) or a null string (when $\ell_i^* = 0$).

Because of the selective disclosure and setup, $\mathcal{B}$ knows the following exponents:

- γ such that $H_s(m_\beta^*) = g^\gamma$.
- δ_j such that $H(m_{s_j, 2^j}^*) = g^{\delta_j}$ when $\ell_j^* = 1$ and $j \neq \beta$.
- z_j when $\ell_j^* = 0$.

t^* is either 1 or 0.

- If $t^* = 1$,
 s_i denotes the position where m_i^* starts. $\mathcal{B}$ can compute the information of some $x_{i,j}$ with the following components of σ^* .

$$- S_{1,\beta} = g^\alpha g^{-x_{1+2\beta, \beta-1}} H_s(m_\beta^*)^{r_c}, \quad \widetilde{S_{1,\beta}} = g^{r_{1,\beta}}$$

$\mathcal{B}$ knows γ such that $H_s(m_\beta^*) = g^\gamma$, so $\mathcal{B}$ can compute $g^\alpha g^{-x_{1+2\beta, \beta-1}} = S_{1,\beta} / \widetilde{S_{1,\beta}}^\gamma$.

- For $j = \beta - 1$ down to 0,

$$* \text{ when } \ell_j = 1, \quad A_{s_j, j} = g^{x_{s_j, j}} g^{-x_{s_{j-1}, j-1}} H(m_j^*)^{r'_{s_j, j}}, \quad \widetilde{A_{s_j, j}} = g^{r'_{s_j, j}}$$

$\mathcal{B}$ knows δ such that $H(m_j^*) = g^\delta$, so $\mathcal{B}$ can compute $g^{x_{s_j, j}} g^{-x_{s_{j-1}, j-1}} = A_{s_j, j} / \widetilde{A_{s_j, j}}^\delta$.

$$* \text{ when } \ell_j = 0, \quad D_{s_j, j} = g^{x_{s_j, j}} g^{-x_{s_{j-1}, j-1}} g^{z_j r''_{s_j, j}}, \quad \widetilde{D_{s_j, j}} = g^{r''_{s_j, j}}$$

$\mathcal{B}$ knows z_j , so $\mathcal{B}$ can compute $g^{x_{s_j, j}} g^{-x_{s_{j-1}, j-1}} = D_{s_j, j} / \widetilde{D_{s_j, j}}^{z_j}$.

so $\mathcal{B}$ can compute $g^{x_{s_j, j}} g^{-x_{s_{j-1}, j-1}}$.

$\mathcal{B}$ has the values of $g^{x_{s_j, j}} g^{-x_{s_{j-1}, j-1}}$ for $j = \beta - 1$ down to 0 and $g^\alpha g^{-x_{1+2\beta, \beta-1}}$, so can compute

$$g^\alpha g^{-x_{1+2\beta, \beta-1}} \prod_{j=0}^{\beta-1} g^{x_{s_j, j}} g^{-x_{s_{j-1}, j-1}} = g^\alpha g^{-x_{s_{-1}, -1}} = g^\alpha$$

- If $t^* = 0$,
 $\mathcal{B}$ parses σ^* as $(B, \tilde{S}, Z_{\beta-1}, \dots, Z_0)$, with

$$\tilde{S} = g^c, \quad Z_{\beta-1} = g^{c_{\beta-1}}, \quad \dots, \quad Z_0 = g^{c_0}$$

for some $c, c_{\beta-1}, \dots, c_0 \in \mathbb{Z}_p$.

$$B = g^\alpha \cdot H_s(m_\beta^*)^c \prod_{j < \beta, \ell_j^* = 1} H(m_j^*)^{c_j} \prod_{j' < \beta, \ell_{j'}^* = 0} (g^{z_{j'}})^{c_{j'}}$$

because the signature is valid.

- $\mathcal{B}$ knows γ such that $H_s(m_\beta^*) = g^\gamma$. $\mathcal{B}$ sets $C := \tilde{S}^\gamma$.

- From $j = \beta - 1$ down to 0, $\mathcal{B}$ proceeds as:
 - * If $\ell_j = 1$, $\mathcal{B}$ knows δ_j such that $H(m_j^*) = g^{\delta_j}$. $\mathcal{B}$ sets $C := C \cdot Z_j^{\delta_j}$;
 - * If $\ell_j = 0$, $\mathcal{B}$ knows z_j . $\mathcal{B}$ sets $C := C \cdot Z_j^{z_j}$.

Then

$$C = H_s(m_\beta^*)^c \prod_{j < \beta, \ell_j^* = 1} H(m_j^*)^{c_j} \prod_{j' < \beta, \ell_{j'}^* = 0} (g^{z_{j'}})^{c_{j'}}$$

so $\mathcal{B}$ can compute $B/C = g^\alpha$.

Thus, whether t^* is 1 or 0, $\mathcal{B}$ can solve for $g^\alpha = g^{ab}$ and correctly answer to the CDH challenge.

Analysis The distribution of the above game and the security game are identical. Thus, whenever $\mathcal{A}$ is successful in a forgery against our scheme, $\mathcal{B}$ will solve the CDH challenge. $\square$

6 A Construction for Subset Predicates based on ABE

In this section we briefly point out a surprising connection to Attribute Based Encryption (ABE). We show that existing constructions for Ciphertext-Policy ABE [7, 36, 57] naturally lead to context hiding quotable signatures for arbitrary message subsets (as opposed to the substring predicate considered in the previous section). In particular, a message will be a sets of strings (strings can be used to encode elements for different types of sets), and the predicate $P(\vec{m}, m') = 1$ iff $m' \subseteq m_i$ for some $m_i \in \vec{m}$. Observe that this disallows “collusions” between two different signatures where m' is a subset of the union of multiple messages, but not any single one. (Otherwise, this would be trivially realizable from standard signatures schemes.)

Our main tool is an observation of Naor that shows that secret keys in Identity Based Encryption [11] can function as signatures. Recall that in (ciphertext-policy) *attribute based encryption* an authority provides secret keys to a user based on the user’s list of attributes. The main challenge in building such systems is preventing collusion attacks: two (or more) users with distinct sets of attributes should be unable to create a secret key for a combination of their attributes.

If we treat words in a message $m \in \Sigma^*$ as attributes, that is, we treat a message $m = (a_1, \dots, a_\ell) \in \Sigma^\ell$ as a sequence of attributes $a_1, \dots, a_\ell$, then we can define the signature on m as a set of ℓ secret keys corresponding to the ℓ attributes in the message. Verifying the signature can be done by trying to decrypt some test ciphertext using the secret keys in the signature. Now, given a signature on m we derive a signature on a subset of the words in m by simply removing the secret keys corresponding to words not in the subset. For context hiding we need to re-randomize the resulting set of secret keys. (Not all CP-ABE schemes may support the removal and re-randomization of secret keys in this manner, but the schemes of [7, 36, 57] do.)

Since ABE security prevents collusion attacks, it is straight forward to show that these signatures are unforgeable in the sense of Definition 2.3. Moreover, due to the re-randomization of secret keys, a derived signature is sampled from the same distribution as a fresh signature and is independent from the given signature. This implies strong context hiding in sense of Definition 2.4.

This unexpected connection between quoting and ABE leads to the following theorem, stated informally.

Theorem 6.1 (informal) *The Ciphertext-Policy ABE systems in [7, 36, 57] translate using Naor’s transformation into a signature scheme supporting quoting for arbitrary subsets of a message. (Selective) security of the CP-ABE systems imply (selective) unforgeability and context hiding.*

In other words, when the ABE scheme provides adaptive (resp, selective) security, then the resulting signature scheme achieves adaptive (resp., selective) unforgeability. The (third) ABE scheme of [57] provides selective security from the d-BDH assumption. Adaptive security is proven in [7], but only in the generic group model. While [36] proves adaptive security under certain static assumptions using composite order groups.

7 Computing Weighted Averages and Fourier Transforms

So far we only constructed schemes for *univariate* predicates P . We now give an example where one computes on multiple signed messages. Let p be a prime, n a positive integer, and $\mathcal{T}$ a set of tags. The message space $\mathcal{M}$ consists of pairs:

$$\mathcal{M} := \mathcal{T} \times \mathbb{F}_p^n$$

Now, define the predicate P as follows: $P(\varepsilon, m) = 1$ for all $m \in \mathcal{M}$ and¹⁰

$$P\left(((t_1, \mathbf{v}_1), \dots, (t_k, \mathbf{v}_k)) , (t, \mathbf{v}) \right) = 1 \iff \begin{cases} t = t_1 = \dots = t_k, \text{ and} \\ \mathbf{v} \in \text{span}(\mathbf{v}_1, \dots, \mathbf{v}_k) \end{cases}$$

Thus, given signatures on vectors $\mathbf{v}_1, \dots, \mathbf{v}_k$ grouped together by the tag t , anyone can create a signature on a linear combination of these vectors. This can be done iteratively so that given signed linear combinations, new signed linear combinations can be created. Unforgeability means that if the adversary obtains signatures on vectors $\mathbf{v}_1, \dots, \mathbf{v}_k$ for particular tag $t \in \mathcal{T}$ then he cannot create a signature on a vector outside the linear span of $\mathbf{v}_1, \dots, \mathbf{v}_k$.

Signature schemes for this predicate P are presented in [14, 13, 12, 15, 2] while schemes over $\mathbb{Z}$ (rather than $\mathbb{F}_p$) are presented in [27]. These schemes were originally designed to secure network coding where context hiding is not needed since there are no privacy requirements for the sender (in fact, the sender is explicitly transmitting all his data to the recipient). The question then is how to construct a system for predicate P above that is both unforgeable and context hiding. Fortunately, we do not need to look very far. The linearly homomorphic signature schemes in [14] can be shown to be context hiding. We state this in the following theorem.

Theorem 7.1 *If the CDH assumption holds in group $\mathbb{G}$, then the signature scheme $\mathbf{NCS}_1$ from [14] is unforgeable and context-hiding in the random oracle model, assuming tags are generated independently at random by the unforgeability challenger when responding to Sign queries.*

Unforgeability is Theorem 6 in [14], which holds only when tags $t_i \in \mathcal{T}$ are generated independently at random by the signer. While context hiding has not been considered before for this scheme, it is not difficult to show that the scheme is context hiding. The scheme is unique in the sense that every vector $\mathbf{v}$ has a unique valid signature.¹¹ It is easy to show that any homomorphic unique signature must be context hiding and hence $\mathbf{NCS}_1$ is context hiding.

Viewed in this way, the scheme $\mathbf{NCS}_1$ gives the ability to carry out authenticated addition on signed data. Consider a server that stores signed data samples $s_1, \dots, s_n$ in $\mathbb{F}_p$. The signature on sample s_i is actually a signature on the vector $(s_i | \mathbf{e}_i) \in \mathbb{F}_p^{n+1}$, where $\mathbf{e}_i$ is the i th unit vector in $\mathbb{F}_p^n$. The server stores (i, s_i) and a signature on $(s_i | \mathbf{e}_i)$. (The vector $\mathbf{e}_i$ need not be stored with the

¹⁰Recall, the signature on ϵ is the output the KeyGen algorithm.

¹¹Recall that in *unique* signatures [38] in addition to the regular unforgeability requirement there is an additional uniqueness property: for any honestly-generated public key pk and any message m in the message space, there do not exist values σ_1, σ_2 such that $\sigma_1 \neq \sigma_2$ and yet $\mathbf{Verify}(pk, m, \sigma_1) = \mathbf{Verify}(pk, m, \sigma_2) = 1$.

data and can be reconstructed from i when needed.) Using **SignDerive**, the server can compute a signature σ on the sum $(\sum_{i=1}^n s_i, 1, \dots, 1)$. Since the schemes are context hiding, the server can publish the sum $\sum_{i=1}^n s_i \bmod p$ and the signature σ on the sum and (provably) reveal no other information on the original data. The “augmentation” $(1, \dots, 1)$ proves that the published message really is the claimed sum of the original samples (the tag t prevents mix-and-match attacks between different data sets). We can similarly publish a sum of any required subset.

More generally, the server can publish an authenticated inner product of the samples $\mathbf{s} := (s_1, \dots, s_n)$ with any public vector $\mathbf{c} \in \mathbb{F}_p^n$ without leaking additional information about the samples. This is needed, for example, to publish a weighted average of the original data set. Similarly, recall that the Fourier transform of the data $(s_1, \dots, s_n)$ is a specific linear operator (represented by a specific $n \times n$ matrix) applied to this vector. Therefore, we can publish signed Fourier coefficients of the data. If we only publish a subset of the Fourier coefficients then, by context hiding, we are guaranteed that no other information about $(s_1, \dots, s_n)$ is revealed.

Acknowledgments

We are grateful to the anonymous reviewers for their helpful comments.

References

- [1] Giuseppe Ateniese, Daniel H. Chou, Breno de Medeiros, and Gene Tsudik. Sanitizable signatures. In *ESORICS '05*, volume 3679 of LNCS, pages 159–177, 2005.
- [2] Nuttapong Attrapadung and Benoit Libert. Homomorphic network coding signatures in the standard model. In *Public Key Cryptography - PKC 2011*, volume 6571, page 17, 2011.
- [3] Mihir Bellare, Oded Goldreich, and Shafi Goldwasser. Incremental cryptography: The case of hashing and signing. In *CRYPTO '94*, volume 839 of LNCS, pages 216–233, 1994.
- [4] Mihir Bellare, Daniele Micciancio, and Bogdan Warinschi. Foundations of group signatures: Formal definitions, simplified requirements, and a construction based on general assumptions. In *EUROCRYPT*, pages 614–629, 2003.
- [5] Mihir Bellare and Gregory Neven. Transitive signatures based on factoring and RSA. In *ASIACRYPT '02*, volume 2501 of LNCS, pages 397–414, 2002.
- [6] Mihir Bellare and Gregory Neven. Transitive signatures: New schemes and proofs. *IEEE Transactions on Information Theory*, 51:2133–2151, 2005.
- [7] John Bethencourt, Amit Sahai, and Brent Waters. Ciphertext-policy attribute-based encryption. In *IEEE Symposium on Security and Privacy*, pages 321–334, 2007.
- [8] Manuel Blum, Alfredo De Santis, Silvio Micali, and Giuseppe Persiano. Noninteractive zero-knowledge. *SIAM Journal of Computing*, 20(6):1084–1118, 1991.
- [9] Dan Boneh and Xavier Boyen. Efficient selective-ID secure identity-based encryption without random oracles. In *Advances in Cryptology – EUROCRYPT '04*, volume 3027, pages 223–238, 2004.
- [10] Dan Boneh, Xavier Boyen, and Hovav Shacham. Short group signatures. In *CRYPTO '04*, volume 3152 of LNCS, pages 45–55, 2004.

- [11] Dan Boneh and Matthew K. Franklin. Identity-based encryption from the Weil pairing. *SIAM J. Comput.*, 32(3), 2003.
- [12] Dan Boneh and David Freeman. Homomorphic signatures for polynomial functions. In *Proc. of Eurocrypt*, 2011. Cryptology ePrint Archive, Report 2011/018.
- [13] Dan Boneh and David Freeman. Linearly homomorphic signatures over binary fields and new tools for lattice-based signatures. In *Proc. of PKC*, volume 6571 of *LNCS*, pages 1–16, 2011. Cryptology ePrint Archive, Report 2010/453.
- [14] Dan Boneh, David Freeman, Jonathan Katz, and Brent Waters. Signing a linear subspace: Signature schemes for network coding. In *Public-Key Cryptography — PKC '09*, volume 5443 of *Springer LNCS*, pages 68–87, 2009.
- [15] Dan Boneh and Michael Hamburg. Generalized identity based and broadcast encryption schemes. In *ASIACRYPT*, pages 455–470, 2008.
- [16] Christina Brzuska, Heike Busch, Özgür Dagdelen, Marc Fischlin, Martin Franz, Stefan Katzenbeisser, Mark Manulis, Cristina Onete, Andreas Peter, Bertram Poettering, and Dominique Schröder. Redactable signatures for tree-structured data: definitions and constructions. In *Applied Cryptography and Network Security (ACNS) '08*, volume 6123 of *LNCS*, pages 87–104, 2010.
- [17] Christina Brzuska, Marc Fischlin, Tobias Freudenreich, Anja Lehmann, Marcus Page, Jakob Schelbert, Dominique Schröder, and Florian Volk. Security of sanitizable signatures revisited. In *Public Key Cryptography*, volume 5443 of *LNCS*, pages 317–336, 2009.
- [18] Christina Brzuska, Marc Fischlin, Anja Lehmann, and Dominique Schröder. Santizable signatures: How to partially delegate control for authenticated data. In *BIOSIG 2009*, pages 117–128, 2009.
- [19] Christina Brzuska, Marc Fischlin, Anja Lehmann, and Dominique Schröder. Unlinkability of sanitizable signatures. In *Public Key Cryptography (PKC) '10*, volume 6056 of *LNCS*, pages 444–461, 2010.
- [20] Jan Camenisch and Anna Lysyanskaya. Signature schemes and anonymous credentials from bilinear maps. In *Advances in Cryptology – CRYPTO '04*, volume 3152, pages 56–72, 2004.
- [21] Ran Canetti, Shai Halevi, and Jonathan Katz. A forward-secure public-key encryption scheme. In *EUROCRYPT*, pages 255–271, 2003.
- [22] Ee-Chien Chang, Chee Liang Lim, and Jia Xu. Short redactable signatures using random trees. In *CT-RSA '09: Proceedings of the The Cryptographers' Track at the RSA Conference 2009 on Topics in Cryptology*, pages 133–147, 2009.
- [23] Denis Charles, K Jain, and K Lauter. Signatures for network coding. *International Journal of Information and Coding Theory*, 1(1):3–14, 2009.
- [24] David Chaum and Eugène van Heyst. Group signatures. In *EUROCRYPT*, volume 547 of *LNCS*, pages 257–265, 1991.
- [25] Whitfield Diffie and Martin Hellman. New directions in cryptography. *IEEE Transactions on Information Theory*, 22:644–654, 1976.

- [26] Christina Fragouli and Emina Soljanin. *Network Coding Fundamentals*. Now Publishers Inc., Hanover, MA, USA, 2007.
- [27] Rosario Gennaro, Jonathan Katz, Hugo Krawczyk, and Tal Rabin. Secure network coding over the integers. In *Public Key Cryptography — PKC '10*, volume 6056 of *Springer LNCS*, pages 142–160, 2010.
- [28] Craig Gentry. *A fully homomorphic encryption scheme*. PhD thesis, Stanford University, 2009.
- [29] Oded Goldreich, Shafi Goldwasser, and Silvio Micali. How to construct random functions (extended abstract). In *FOCS*, pages 464–479, 1984.
- [30] Shafi Goldwasser, Silvio Micali, and Ronald L. Rivest. A digital signature scheme secure against adaptive chosen-message attacks. *SIAM Journal of Computing*, 17(2):281–308, 1988.
- [31] Stuart Haber, Yasuo Hatano, Yoshinori Honda, William Horne, Kunihiko Miyazaki, Tomas Sander, Satoru Tezoku, and Danfeng Yao. Efficient signature schemes supporting redaction, pseudonymization, and data deidentification. In *ASIACCS '08*, pages 353–362, 2008.
- [32] Alejandro Hevia and Daniele Micciancio. The provable security of graph-based one-time signatures and extensions to algebraic signature schemes. In *ASIACRYPT '02*, volume 2501 of *LNCS*, pages 379–396, 2002.
- [33] Susan Hohenberger and Brent Waters. Realizing hash-and-sign signatures under standard assumptions. In *EUROCRYPT '09*, volume 5479 of *LNCS*, pages 333–350, 2009.
- [34] Robert Johnson, David Molnar, Dawn Song, and David Wagner. Homomorphic signature schemes. In *CT-RSA*, pages 244–262. Springer-Verlag, 2002.
- [35] M. Krohn, M. Freedman, and D. Mazieres. On-the-fly verification of rateless erasure codes for efficient content distribution. In *Proc. of IEEE Symposium on Security and Privacy*, pages 226–240, 2004.
- [36] Allison B. Lewko, Tatsuaki Okamoto, Amit Sahai, Katsuyuki Takashima, and Brent Waters. Fully secure functional encryption: Attribute-based encryption and (hierarchical) inner product encryption. In *EUROCRYPT*, 2010.
- [37] Allison B. Lewko and Brent Waters. New techniques for dual system encryption and fully secure HIBE with short ciphertexts. In *TCC '10*, volume 5978 of *LNCS*, pages 455–479, 2010.
- [38] Anna Lysyanskaya. Unique signatures and verifiable random functions from the DH-DDH separation. In *CRYPTO*, pages 597–612, 2002.
- [39] Silvio Micali. Computationally sound proofs. *SIAM J. Comput.*, 30(4):1253–1298, 2000.
- [40] Silvio Micali and Ronald L. Rivest. Transitive signature schemes. In *CT-RSA '02*, volume 2271 of *LNCS*, pages 236–243, 2002.
- [41] Kunihiko Miyazaki, Goichiro Hanaoka, and Hideki Imai. Digitally signed document sanitizing scheme based on bilinear maps. In *ASIACCS '06: Proceedings of the 2006 ACM Symposium on Information, computer and communications security*, pages 343–354, 2006.

- [42] Kunihiro Miyazaki, Mitsuru Iwamura, Tsutomu Matsumoto, Ryoichi Sasaki, Hiroshi Yoshiura, Satoru Tezuka, and Hideki Imai. Digitally signed document sanitizing scheme with disclosure condition control. *IEICE Transactions on Fundamentals*, E88-A(1):239–246, 2005.
- [43] Kunihiro Miyazaki, Seiichi Susaki, Mitsuru Iwamura, Tsutomu Matsumoto, Ryoichi Sasaki, and Hiroshi Yoshiura. Digital document sanitizing problem. *IEICE Technical Report*, 103(195(IEEC2003 12-29)):61–67, 2003.
- [44] David Naccache. Is theoretical cryptography any good in practice? CHES 2010 invited talk, 2010. available at www.iacr.org/workshops/ches/ches2010.
- [45] Gregory Neven. A simple transitive signature scheme for directed trees. *Theoretical Computer Science*, 396(1-3):277–282, 2008.
- [46] Ronald Rivest. Two signature schemes. Slides from talk given at Cambridge University, 2000. <http://people.csail.mit.edu/rivest/Rivest-CambridgeTalk.pdf>.
- [47] Ronald L. Rivest, Adi Shamir, and Leonard Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Comm. of the ACM*, 21(2):120–126, February 1978.
- [48] Ronald L. Rivest, Adi Shamir, and Yael Tauman. How to leak a secret: Theory and applications of ring signatures. In *Essays in Memory of Shimon Even*, pages 164–186, 2006.
- [49] Siamak Fayyaz Shahandashti, Mahmoud Salmasizadeh, and Javad Mohajer. A provably secure short transitive signature scheme from bilinear group pairs. In *Security and Communication Networks*, volume 3352 of *LNCS*, page 6076, 2005.
- [50] Adi Shamir. On the generation of cryptographically strong pseudorandom sequences. *ACM Transaction on Computer Systems*, 1:38–44, 1983.
- [51] N. P. Smart and F. Vercauteren. Fully homomorphic encryption with relatively small key and ciphertext sizes. In *Public Key Cryptography — PKC '10*, volume 6056 of *Springer LNCS*, pages 420–443, 2010.
- [52] Nigel Smart. ECRYPT2 Yearly Report on Algorithms and Keysizes (2008-2009), Revision 1.0, July 27, 2009. Edited by Smart. Available at <http://www.ecrypt.eu.org/documents/D.SPA.7.pdf>.
- [53] Ron Steinfeld, Laurence Bull, and Yuliang Zheng. Context extraction signatures. In *Information Security and Cryptology (ICISC)*, volume 2288 of *LNCS*, pages 285–304, 2001.
- [54] Marten van Dijk, Craig Gentry, Shai Halevi, and Vinod Vaikuntanathan. Fully homomorphic encryption over the integers. In *Advances in Cryptology — EUROCRYPT '10*, volume 6110 of *Springer LNCS*, pages 24–43, 2010.
- [55] Brent Waters. Efficient identity-based encryption without random oracles. In *Advances in Cryptology — EUROCRYPT '05*, volume 3494, pages 320–329, 2005.
- [56] Brent Waters. Dual system encryption: Realizing fully secure IBE and HIBE under simple assumptions. In *Advances in Cryptology — CRYPTO '09*, volume 5677, pages 619–636, 2009.
- [57] Brent Waters. Ciphertext-policy attribute-based encryption: an expressive, efficient, and provably secure realization. In *Public Key Cryptography — PKC '11*, pages 53–70, 2011.

- [58] Lei Wei, Scott E. Coull, and Michael K. Reiter. Bounded vector signatures and their applications. In *ASIACCS '11*, pages 277–285, 2011.
- [59] Xun Yi. Directed transitive signature scheme. In *CT-RSA '07*, volume 4377 of LNCS, page 129144, 2007.
- [60] Fang Zhao, Ton Kalker, Muriel Médard, and Keesook Han. Signatures for content distribution with network coding. In *Proc. Intl. Symp. Info. Theory (ISIT)*, 2007.

A A Computational Definition of Context Hiding

Let $(\mathbf{KeyGen}, \mathbf{SignDerive}, \mathbf{Verify})$ be a P -homomorphic signature scheme for predicate P and message $\mathcal{M}$. Consider the following game to model context hiding:

Setup: The challenger runs the algorithm $(pk, sk) \leftarrow \mathbf{KeyGen}(1^\lambda)$ to obtain the public key pk and the secret key sk , and gives pk to the adversary.

Query Phase 1: Proceeding adaptively, the adversary may query any of the three oracles from the unforgeability game:

- $\mathbf{Sign}(m \in \mathcal{M})$: (same as in the unforgeability game)
- $\mathbf{SignDerive}(i \in \mathbb{Z}, m')$: (same as in the unforgeability game)
- $\mathbf{Reveal}(i \in \mathbb{Z})$: (same as in the unforgeability game)

Challenge: At some point, the adversary issues a challenge (m, m') where $P(m, m') = 1$ for any $m, m' \in \mathcal{M}$. The challenger computes the following three values: $\sigma \leftarrow \mathbf{Sign}(sk, m)$, $\sigma_0 \leftarrow \mathbf{Sign}(sk, m')$ and $\sigma_1 \leftarrow \mathbf{SignDerive}(pk, \sigma, m, m')$. The challenger then picks a random $b \in \{0, 1\}$ and returns (σ, σ_b) to the adversary. Note: there are no restrictions on m, m' other than that they be in the message space; in particular, they could be equal and one or both could have been previously signed.

Query Phase 2: Proceeding adaptively, the adversary may query the oracles from Phase 1.

Output: Eventually, the adversary will output a bit b' and is said to win if $b = b'$.

We define $\mathbf{Adv}_{\mathcal{A}}^{\mathbf{CH}}$ to be the probability that adversary $\mathcal{A}$ wins in the above game minus $\frac{1}{2}$.

Definition A.1 (Context Hiding) For a predicate P and message space $\mathcal{M}$, a P -homomorphic signature scheme $(\mathbf{KeyGen}, \mathbf{Sign}, \mathbf{SignDerive}, \mathbf{Verify})$ is context hiding if for all probabilistic polynomial time adversaries $\mathcal{A}$, $\mathbf{Adv}_{\mathcal{A}}^{\mathbf{CH}}$ is negligible in λ .

A.1 Relation to Strong Context Hiding

Lemma A.2 A homomorphic signature scheme that is strongly context hiding is context hiding.

Proof. (Sketch) Let $\Pi = (\mathbf{KeyGen}, \mathbf{SignDerive}, \mathbf{Verify})$ be a homomorphic signature scheme and let A be an adversary that has advantage $\mathbf{Adv}_A^{\mathbf{CH}}(\Pi) = p(\lambda)$ in the context-hiding game. The advantage probability for A is taken over the random coins of the key generation, random coins of the $\mathbf{Sign}$ and $\mathbf{SignDerive}$ operations used in the first query phase, the random coins used by algorithm A , and the random coins used by the rest of the experiment. Therefore by an averaging argument,

there must exist *some particular* key pair $(PK, SK) \leftarrow \mathbf{KeyGen}(1^\lambda; z_1)$, some *particular* random tape z_q for the **Sign** and **SignDerive** operations used in the first query phase, some *particular* random coins z_A for A , and some *particular* message pair (m, m') output by A over which the probability of A winning the context-hiding game in this case is at least $p(\lambda)$. Let the values of the random tapes be given as non-uniform advice.

We show how this information can be used to construct a (non-uniform) adversary A' that distinguishes $\{(SK, \sigma, \mathbf{Sign}(SK, m'))\}$ from $\{(SK, \sigma, \mathbf{SignDerive}(PK, \sigma, m, m'))\}$ with probability $p(\lambda)$ for the triple $((PK, SK), m, m')$. Thus, if Π is strongly context hiding, then $p(\lambda)$ must be exponentially small, and so Π must also be context-hiding.

The adversary A' works as follows: On input the challenge tuple (SK, σ, σ') , A' begins to run the context-hiding experiment for $A(PK; z_A)$. A' answers the queries that A asks by using SK and the random tape z_q to run **Sign** and **SignDerive**. When A outputs a challenge message pair (m, m') (which must occur by construction), then A' answers with (σ, σ') . A' answers the second-phase queries of A using SK and fresh random coins. Finally, when A outputs b' , A' echoes this answer as output and halts.

First observe that A' performs a perfect simulation of the context-hiding game. When the input pair (σ, σ') corresponds to $(\mathbf{Sign}(SK, m), \mathbf{Sign}(SK, m'))$, then A' simulates the context-hiding game for $b = 0$. In the other case, A' simulates the context-hiding game for $b = 1$. Therefore, A' distinguishes

$$\begin{aligned} & \{(SK, \mathbf{Sign}(SK, m), \mathbf{Sign}(SK, m'))\}_{SK, m, m'} \\ & \{(SK, \mathbf{Sign}(SK, m), \mathbf{SignDerive}(PK, \sigma, m, m'))\}_{SK, m, m'} \end{aligned}$$

with probability $p(\lambda)$. □

A.2 Simplified Unforgeability Under Strong Context Hiding

We now show how the strong context hiding property can help simplify the security argument for unforgeability. In particular, we introduce a weaker notion of unforgeability in which the adversary only makes calls to the **Sign** oracle and immediately receives a signature.

— Game $\mathbf{NHU}(\Pi, \mathcal{A}, \lambda, P)$: This game is the same as the $\mathbf{Unforg}(\Pi, \mathcal{A}, \lambda, P)$ game with the exception that only the following query is allowed:

— $\text{Sign}(m \in \mathcal{M})$: the oracle computes $\sigma \leftarrow \mathbf{Sign}(SK, m)$, adds m to Q and returns σ .

Note, the only difference between game $\mathbf{NHU}$ and the standard unforgeability game for a signature scheme is that in this game, the adversary only wins if it produces a forgery on a signature m^* such that for all $m \in Q$, $P(m, m^*) = 0$, whereas in the standard unforgeability game, the adversary wins if it produces a signature on *any* message that is not in Q .

Definition A.3 A quoteable signature scheme Π is **NHU-unforgeable** if for all efficient adversaries $\mathcal{A}$, it holds that $\Pr[\mathbf{NHU}(\Pi, \mathcal{A}, \lambda, P) = 1] < \text{negl}(\lambda)$ for some negligible function λ .

Lemma A.4 A signature scheme that is **NHU-unforgeable** and strongly context hiding is **Unforg-unforgeable**.

Proof. Our plan is to present a series of hybrid experiments that are meant to simplify the quoteable unforgeability game.

Hybrid $H_1(\Pi, \mathcal{A}, \lambda, P)$ Consider the first hybrid experiment H_1 which is the same as the unforgeability game **Unforg** $(\Pi, \mathcal{A}, \lambda, P)$, with the exception that all *Sign* and *SignDerive* queries are lazily evaluated. That is, when $\mathcal{A}$ makes a query, the experiment responds in the following way:

- *Sign*(m): generate a handle i and record information $(i, ?, m, \epsilon)$ in T and return i
- *SignDerive*(i, m'): retrieve $(i, z, m, \cdot)$ from T , return $\perp$ if it does not exist or if $P(m, m') \neq 1$, generate a new handle i' , record $(i', ?, m', i)$ in T , and return i'
- *Reveal*(i): retrieve (i, z, m, i_1) from T (returning $\perp$ if it does not exist). If $z \neq ?$, then return z . Otherwise, if $i_1 = \epsilon$, then compute $\sigma \leftarrow \mathbf{Sign}(SK, m)$, replace the entry (i, z, m, ϵ) with (i, σ, m, ϵ) , and return σ . Finally, if $i_1 \neq \epsilon$, then recursively call $z_1 \leftarrow \mathbf{Reveal}(i_1)$, obtain $(i_1, \cdot, m_1, \cdot)$ from T and compute $\sigma \leftarrow \mathbf{SignDerive}(PK, z_1, m_1, m)$. Replace the entry with (i, σ, m, i_1) , and return σ .

Claim A.5 $\Pr[H_1(\Pi, \mathcal{A}, \lambda, P) = 1] = \Pr[\mathbf{Unforg}(\Pi, \mathcal{A}, \lambda, P) = 1]$.

This claim follows by inspection. For any query that is eventually revealed, the same operations are performed in both H_1 and the original game. For any query that is never revealed, no operation in H_1 is performed; but this does not affect the view of the adversary, and therefore does not affect the output of the adversary.

Hybrid $H_{2,i}(\Pi, \mathcal{A}, \lambda, P)$ The second hybrid is the same as H_1 except that the first i queries to *Reveal* are answered using *Reveal₂* described below, and the remaining queries are answered as per H_1 : *(The only difference is that $\mathbf{Sign}(SK, m_1)$ is used in place of $\mathbf{SignDerive}(PK, z_1, m_1, m)$ in the second to last sentence.)*

- *Reveal₂*(i): retrieve (i, z, m, i_1) from T (returning $\perp$ if it does not exist). If $z \neq ?$, then return z . Otherwise, if $i_1 = \epsilon$, then compute $\sigma \leftarrow \mathbf{Sign}(SK, m)$, replace the entry (i, z, m, ϵ) with (i, σ, m, ϵ) , and return σ . Finally, if $i_1 \neq \epsilon$, then recursively call $z_1 \leftarrow \mathbf{Reveal}(i_1)$, obtain $(i_1, \cdot, m_1, \cdot)$ from T and compute $\sigma \leftarrow \mathbf{Sign}(SK, m_1)$. Replace the entry with (i, σ, m, i_1) , and return σ .

Claim A.6 $H_{2,0}(\Pi, \mathcal{A}, \lambda, P)$ is identically distributed to $H_1(\Pi, \mathcal{A}, \lambda, P)$.

By inspection.

Claim A.7 $H_{2,i}(\Pi, \mathcal{A}, \lambda, P)$ is identically distributed to $H_{2,i-1}(\Pi, \mathcal{A}, \lambda, P)$ for $i \geq 1$.

This claim follows via the strong context-hiding property of the signature scheme because this property guarantees $\mathbf{Sign}(SK, m')$ and $\mathbf{SignDerive}(PK, \sigma, m, m')$ are statistically close.

Suppose that $\mathcal{A}$ makes $\ell = \text{poly}(\lambda)$ queries. Observe that $H_{2,\ell}(\Pi, \mathcal{A}, \lambda, P)$ only evaluates **Sign**, and only does so on messages that are immediately returned to the adversary. Thus, $H_{2,\ell}$ is syntactically equivalent to the **NHU** game. Since the $H_{2,\ell}$ game enables $\mathcal{A}$ to produce a forgery with the same probability as **Unforg** $(\Pi, \mathcal{A}, \lambda, P)$, we have that $\mathbf{Unforg}(\Pi, \mathcal{A}, \lambda, P) = \mathbf{NHU}(\Pi, \mathcal{A}, \lambda, P)$ which completes the lemma. $\square$

Batch Verification of Short Signatures

Jan Camenisch
IBM Research - Zürich
jca@zurich.ibm.com

Susan Hohenberger
Johns Hopkins University
susan@cs.jhu.edu

Michael Østergaard Pedersen
Miracle A/S
mop@miracleas.dk

August 11, 2011

Abstract

With computer networks spreading into a variety of new environments, the need to authenticate and secure communication grows. Many of these new environments have particular requirements on the applicable cryptographic primitives. For instance, a frequent requirement is that the communication overhead inflicted be small and that many messages be processable at the same time. In this paper, we consider the suitability of public key signatures in the latter scenario. That is, we consider signatures that are 1) short and 2) where many signatures from (possibly) different signers on (possibly) different messages can be verified quickly. Prior work focused almost exclusively on batching signatures from the same signer.

We propose the first batch verifier for messages from many (certified) signers without random oracles and with a verification time where the dominant operation is independent of the number of signatures to verify. We further propose a new signature scheme with very short signatures, for which batch verification for many signers is also highly efficient. Combining our new signatures with the best known techniques for batching certificates from the same authority, we get a fast batch verifier for certificates and messages combined. Although our new signature scheme has some restrictions, it is very efficient and still practical for some communication applications.

1 Introduction

As the world moves towards pervasive computing and communication, devices from vehicles to dog collars will soon be expected to communicate with their environments. For example, many governments and industry consortia are currently planning for the future of *intelligent cars* that constantly communicate with each other and the transportation infrastructure to prevent accidents and to help alleviate traffic congestion [16, 50]. Raya and Hubaux suggest that vehicles will transmit safety messages every 300ms to all other vehicles within a minimum range of 110 meters [49], which in turn may retransmit these messages.

For such pervasive systems to work properly, there are many competing constraints [16, 50, 37, 49]. For one, there are physical limitations, such as a limited spectrum allocation for specific types of communications and the potential roaming nature of devices, that require that messages be kept very short and (security) overhead be minimal [37]. Yet for messages to be trusted by their recipients, they need to be authenticated in some fashion, so that entities spreading false information can be held accountable. Thus, some short form of authentication must be added. Furthermore, different messages from many different signers may need to be verified and processed

quickly (e.g., every 300ms [49]). Another possible constraint that these authentications remain anonymous or pseudonymous, we leave as an exciting open problem.

In this work, we consider the suitability of public key signatures to the needs of pervasive communication applications. Generating one signature every 300ms is not a problem for current systems, but transmitting and/or verifying 100+ messages per second might pose a problem. Using RSA signatures for example seems attractive as they are verified quickly, however, one would need approximately 3000 bits to represent a signature on a message plus the certificate (i.e., the public key and signature on that public key) which might be too much for some applications (see Section 8.2 of [49]). While many new schemes based on bilinear maps can provide the same security with significantly smaller signatures, they take significantly more time to verify (e.g., [9, 6, 13, 53]). Thus, it is not immediately clear what the proper tradeoff between message length and verification time is for many pervasive communication applications. However, in some applications, there is evidence that doing a *small* amount of additional computation is more advantageous than sending longer messages. For example, Landsiedel, Wehrle, and Götz showed that for applications using Mica2 sensors transmitting data consumes significantly more battery power than keeping the CPU active [40]. Barr and Asanović note that the wireless transmission of just a single bit, can use more than 1000 times the energy required for a 32 bit computation [3].

1.1 Our Contributions

Now, if one wants both, short signatures and short verification times, it seems that one needs to improve on the verification of the bilinear-map based schemes or try to reduce the signature size of a fast signature scheme such as RSA. In this paper we take the first route and investigate the known batch-verification techniques and to what extent they are applicable to bilinear-map based schemes, whereas for example Gentry provides a method for compressing Rabin signatures in [28]. We note that while these two techniques are not mutually exclusive (in fact Gentry mentions that the compressed Rabin signatures can be aggregated [28]), compressing signatures has not been the focus of our work. More precisely, the main contributions of this paper are:

1. We instantiate the general batch verification definitions of Bellare, Garay, and Rabin [4] to the case of signatures from many signers. We also do this for a weaker notion of batch verification called *screening* and show the relation of these notions to the one of aggregate signatures. Surprisingly, for most known aggregate signature schemes a batching algorithm is provably *not* obtained by aggregating many signatures and then verifying the aggregate.
2. We present a batch verifier for the Π -IBS scheme [19]. (More precisely, this is the IBS scheme implicitly defined by the Chatterjee-Sarkar hierarchical IBE [19] and it can also be viewed as a generalized version of the Boyen-Waters IBS [11] as we will discuss later.) To our knowledge, this is the first batch verifier for a signature scheme without random oracles. Let z be the additional security parameter required by the Π -IBS scheme. When identities and messages are k bits, viewed as z chunks of k/z bits each, our algorithm verifies n Π -IBS signatures using only $(z + 3)$ pairings. Individually verifying n signatures would cost $3n$ pairings.
3. We present a new signature scheme, Π -Sig, derived from the Camenisch-Lysyanskaya signature scheme [13], which is secure in the random oracle model. Π -Sig signatures require only one-third the space of the original CL signatures—on par with the shortest signatures known [9]—, but users may only issue one signature per period (e.g., users might only be allowed to sign

one message per 300ms). We present a batch verifier for these signatures from many different signers that verifies n signatures using only three total pairings, instead of the $5n$ pairings required by n original CL signatures. Yet, our batch verifier has the restriction that it can only batch verify signatures made during the same period. Π -Sig signatures form the core of the only public key authentication, known to us, that is extremely short and highly efficient to verify in bulk.

4. Often signatures and certificates need to be verified together. This happens implicitly in IBS schemes. To achieve this functionality with the Π -Sig signature scheme, we can issue signatures with Π -Sig and certificates with the Boneh, Lynn and Shacham scheme [9]. Then we can batch the Π -Sig signatures (on any message from any signer) using a new batch verifier proposed herein; and we can batch the BLS certificates (on any public key from the same authority) using a known batch verifier that can batch verify n signatures from the *same* signer using only two pairings.

1.2 Batch Verification Overview

We start by a some historical remarks and then later present the schemes relevant to this paper in more detail.

Batch cryptography was introduced in 1989 by Fiat [26] for a variant of RSA. Later, in 1994, Naccache, M'Raihi, Vaudenay and Raphaeli [48] gave the first efficient batch verifier for DSA signatures, however an interactive batch verifier presented in an early version of their paper was broken by Lim and Lee [43]. In 1995 Lai and Yen proposed a new method for batch verification of DSA and RSA signatures [39], but the RSA batch verifier was broken five years later by Boyd and Pavlovski [10]. In 1998, Harn presented two batch verification techniques for DSA and RSA [32, 33] but both were later broken [10, 35, 36]. The same year, Bellare, Garay and Rabin took the first systematic look at batch verification [4] and presented three generic methods for batching modular exponentiations, called the *random subset test*, the *small exponents test* and the *bucket test* which are similar to the ideas from [48, 39]. They showed how to apply these methods to batch verification of DSA signatures and also introduced a weaker form of batch verification called *screening*. Later, Cheon and Lee introduced two new methods called the *sparse exponents test* and the *complex exponents test* [22], which they claim to be about twice as fast as the small exponents test. In 2000, Boyd and Pavlovski published some attacks against different batch verification schemes, mostly ones based on the small exponents test and related tests [10]. These attacks do not invalidate the proof of security for the small exponents test, but rather show how the small exponents test is often used in a wrong way. However, the authors also describe methods to repair some broken schemes based on this test. In 2001, Hoshino, Masayuki and Kobayashi [34] pointed out that the problem discovered by Boyd and Pavlovski [10] might not be critical for batch verification of signatures, but only when using batch verification to verify for example zero-knowledge proofs. In 2004 Yoon, Cheon and Kim proposed a new ID-based signature scheme with batch verification [21], but their security proof is for aggregate signatures and does not meet the definition of batch verification by Bellare et al. [4]; hence their title is somewhat misleading. Other schemes for batch verification based on bilinear maps were proposed [17, 54, 55, 56] but all were later broken by Cao, Lin and Xue [15]. In 2006, a method was proposed for identifying invalid signatures in RSA-type batch signatures [42], but Stanek [52] showed that this method is flawed. Shacham and Boneh gave a practical application of batch verification by using a modified version of Fiat's batch verifier for RSA to improve the

efficiency of SSL handshakes on a busy server [51]. Ferrara, Green, Hohenberger, and Pedersen gave performance measurements for the schemes herein, and also showed how to batch verify other types of signatures, such as group and ring signatures [25]. Law and Matt pointed out some IBS schemes that batch well, and also give methods for identifying invalid signatures in a batch [41].

Bellare, Garay and Rabin Testing Techniques. Let g generate a group of prime order. In 1998, Bellare, Garay and Rabin described some tests [4], for verifying equations of the form $y_i = g^{x_i}$ for $i = 1$ to n . Obviously if one just multiplies these equations together and checks if $\prod_{i=1}^n y_i = g^{\sum_{i=1}^n x_i}$, it is easy to produce two pairs (x_1, y_1) and (x_2, y_2) such that the product of them verifies correctly, but each individual verification does not, e.g., by submitting the pairs $(x_1 - \alpha, y_1)$ and $(x_2 + \alpha, y_2)$ instead. Let us review three fixes to this broken initial proposal.

Random Subset Test: The idea here is to pick a random subset of these pairs (x_i, y_i) and multiply them together, hoping to split up the pairs that were specifically crafted to cancel each other out. Repeating this test ℓ times, picking a new random subset every time, results in the probability of accepting invalid pairs being $2^{-\ell}$.

Small Exponents Test: Instead of picking a random subset every time, one can choose exponents δ_i of (a small number of) ℓ bits and compute $\prod_{i=1}^n y_i^{\delta_i} = g^{\sum_{i=1}^n x_i \delta_i}$. Bellare et al. prove that this test results in the probability of accepting a bad pair being $2^{-\ell}$. The size of ℓ is a tradeoff between efficiency and security and hence it is difficult to give an exact recommendation for it. It all depends on the application and how critical it is not to accept even a single invalid signature. For just a rough check that all signatures are correct 20 bits seems reasonable. In a higher security setting we should probably be using around 64 bits.

Bucket Test: This method is even more efficient than the small exponents test for large values of n . The idea is to repeat a test called the *atomic bucket test* m times. The atomic bucket test works by first putting the n instances one wants to verify into M buckets at random. This results in M new instances of the same problem, which are then checked using the small exponents test with security parameter m . After repeating the atomic bucket test m times, the probability of accepting a bad pair in the original n instances is at most 2^{-m} .

1.3 Efficiency of Prior Work and our Contributions

Efficiency will be given as an abstract cost for computing different functions. We begin by discussing prior work on RSA, DSA, and BLS signatures mostly for single signers, and then discuss our new work on Π -IBS, Π -Sig and BLS signatures for many signers. Note that Lim [44] provides a number of efficient methods for doing m -term exponentiations and Granger and Smart [31] give improvements over the naive method for computing a product of pairings, which is why we state them explicitly.

$m\text{-MultPairCost}_{\mathbb{G}, \mathbb{H}}^s$	s m -term pairings $\prod_{i=1}^m \mathbf{e}(g_i, h_i)$ where $g_i \in \mathbb{G}$, $h_i \in \mathbb{H}$.
$m\text{-MultExpCost}_{\mathbb{G}}^s(k)$	s m -term exponentiations $\prod_{i=1}^m g^{a_i}$ where $g \in \mathbb{G}$, $ a_i = k$.
$\text{PairCost}_{\mathbb{G}, \mathbb{H}}^s$	s pairings $\mathbf{e}(g_i, h_i)$ for $i = 1 \dots s$, where $g_i \in \mathbb{G}$, $h_i \in \mathbb{H}$.
$\text{ExpCost}_{\mathbb{G}}^s(k)$	s exponentiations g^{a_i} for $i = 1 \dots s$ where $g \in \mathbb{G}$, $ a_i = k$.
$\text{GroupTestCost}_{\mathbb{G}}^s$	Testing whether or not s elements are in the group $\mathbb{G}$.
$\text{HashCost}_{\mathbb{G}}^s$	Hashing s values into the group $\mathbb{G}$.
MultCost^s	s multiplications in one or more groups.

If $s = 1$ we will omit it. Throughout this paper we assume that n is the number of message/signature pairs and ℓ_b is a security parameter such that the probability of accepting a batch that contains an invalid signature is at most $2^{-\ell_b}$.

RSA* is a modified version of RSA by Boyd and Pavlovski [10]. The difference to normal RSA is that the verification equation accepts a signature σ as valid if $\alpha\sigma^e = m$ for some element $\alpha \in \mathbb{Z}_m^*$ of order no more than 2, where m is the product of two primes. The signatures are usually between 1024 – 2048 bits and the same for the public key. A single signer batch verifier for this signature scheme with cost $n\text{-MultExpCost}_{\mathbb{Z}_m}^2(\ell_b) + \text{ExpCost}_{\mathbb{Z}_m}(k)$, where k is the number of bits in the public exponent e , can be found in [10]. Note that verifying n signatures by verifying each signature individually only costs $\text{ExpCost}_{\mathbb{Z}_m}^n(k)$, so for small values of e ($|e| < 2\ell_b/3$) the naive method is a faster way to verify RSA signatures and it can also handle signatures from multiple signers. Bellare et al. [4] presents a screening algorithm for RSA that assumes distinct messages from the same signer and costs $2n + \text{ExpCost}_{\mathbb{Z}_m}(k)$.

DSA** is a modified version of DSA from [48] compatible with the *small exponents test* from [10]. There are two differences to normal DSA. First there is no reduction modulo q , so the signatures are 672 bits instead of 320 bits and second, individual verification should check both a signature σ and $-\sigma$ and accept if one of them holds. Messages and public keys are both 160 bits long. Using the small exponents test the cost is $n\text{-MultExpCost}_{\mathbb{G}}(\ell_b) + \text{ExpCost}_{\mathbb{G}}^2(160) + \text{HashCost}_{\mathbb{G}}^n + \text{MultCost}^{2n+1}$ multiplications. This method works for a single signer only.

II-IBS is an IBS scheme derived from the Chatterjee and Sarkar HIBE scheme [19] for which we provide a batch verifier without random oracles in Section 4. An interesting property of this scheme is that the identity does not need to be verified separately. Identities and messages are k bits divided into z logical chunks, each of k/z bits, where z is a security parameter, and a signature is three bilinear group elements. The computational effort required depends on the number of messages and the security parameters. Let $M = n\text{-MultExpCost}_{\mathbb{G}_T}(\ell_b) + n\text{-MultExpCost}_{\mathbb{G}}^3(\ell_b) + \text{PairCost}_{\mathbb{G},\mathbb{G}}^3 + \text{GroupTestCost}_{\mathbb{G}}^{3n} + \text{MultCost}^3$ and refer to the table below for efficiency of the scheme.

$$\begin{array}{ll} n \leq 2z : & M + 2n\text{-MultPairCost}_{\mathbb{G},\mathbb{G}} + z\text{-MultExpCost}_{\mathbb{G}}^{2n}(\frac{k}{z}) + \text{ExpCost}_{\mathbb{G}}^{2n}(\ell_b) \\ n > 2z : & M + z\text{-MultPairCost}_{\mathbb{G},\mathbb{G}} + \text{ExpCost}_{\mathbb{G}}^{2n}(\frac{k}{z} + \ell_b) + \text{MultCost}^{zn} \end{array}$$

The naive application of II-IBS to verify n signatures costs $\text{PairCost}_{\mathbb{G},\mathbb{G}}^{3n} + z\text{-MultExpCost}_{\mathbb{G}}^{2n}(\frac{k}{z}) + \text{MultCost}^{4n}$. Also note that in many security applications we do not need to transmit the identity as a separate parameter, as it is already included in the larger protocol. For example, the identity may be the hardware address of the network interface card.

BLS is the signature scheme by Boneh, Lynn and Shacham [9]. We discuss batch verifiers for BLS signatures based on the small exponents test. For a screening algorithm, aggregate signatures by Boneh, Gentry, Lynn and Shacham [8] can be used. The signature is only one group element in a bilinear group and the same for the public key. For different signers the cost of batch verification is $n\text{-MultPairCost}_{\mathbb{G},\mathbb{G}} + n\text{-MultExpCost}_{\mathbb{G}}(\ell_b) + \text{PairCost}_{\mathbb{G},\mathbb{G}} + \text{ExpCost}_{\mathbb{G}_T}^n(\ell_b) + \text{GroupTestCost}_{\mathbb{G}}^n + \text{HashCost}_{\mathbb{G}}^n$, but for single signer it is only $n\text{-MultExpCost}_{\mathbb{G}}^2(\ell_b) + \text{PairCost}_{\mathbb{G},\mathbb{G}}^2 + \text{GroupTestCost}_{\mathbb{G}}^n + \text{HashCost}_{\mathbb{G}}^n$.

Π -Sig is a new variant of Camenisch and Lysyanskaya signatures [13] presented in Section 5 designed specifically to enable efficient batch verification. The signature is only one bilinear group element and the same for the public key. Batch verification costs $n \cdot \text{MultExpCost}_{\mathbb{G}}^2(\ell_b) + n \cdot \text{MultExpCost}_{\mathbb{G}}(|w| + \ell_b) + \text{PairCost}_{\mathbb{G}, \mathbb{G}}^3 + \text{GroupTestCost}_{\mathbb{G}}^n + \text{HashCost}_{\mathbb{G}}^n$, where w is the output of a hash function. However, the scheme has some additional restrictions.

Small Exponents and Bucket Tests. Recall the various testing techniques covered in Section 1.2. Our batch verifiers in this paper make use of the small exponents test, but since the bucket test uses the small exponents test as a subroutine, we note that we can also use the bucket test to further speed up verification of many signatures.

2 Definitions

Recall that a *digital signature scheme* is a tuple of algorithms $(\text{Gen}, \text{Sign}, \text{Verify})$ that also is *correct* and *secure*. The correctness property states that for all $\text{Gen}(1^\ell) \rightarrow (pk, sk)$, the algorithm $\text{Verify}(pk, m, \text{Sign}(sk, m)) = 1$.

There are two common notions of security. Goldwasser, Micali and Rivest [30] defined a scheme to be *unforgeable* as follows: Let $\text{Gen}(1^\ell) \rightarrow (pk, sk)$. Suppose (m, σ) is output by a p.p.t. adversary $\mathcal{A}$ with access to a signing oracle $\mathcal{O}_{sk}(\cdot)$ and input pk . Then the probability that m was *not* queried to $\mathcal{O}_{sk}(\cdot)$ and yet $\text{Verify}(pk, m, \sigma) = 1$ is negligible in ℓ . An, Dodis and Rabin [1] proposed the notion of *strong unforgeability*, where if $\mathcal{A}$ outputs a pair (m, σ) such that $\text{Verify}(pk, m, \sigma) = 1$, then except with negligible probability at some point the signing oracle $\mathcal{O}_{sk}(\cdot)$ was queried on m and outputted signature σ exactly. In other words, an adversary cannot create a new signature even for a previously signed message. Our batch verification definitions work with either notion. The signatures used in Section 4 meet the GMR [30] definition, while those in Section 5 meet the stronger ADR [1] definition.

Now, we consider the case where we want to quickly verify a set of signatures on (possibly) different messages by (possibly) different signers. The input is $\{(t_1, m_1, \sigma_1), \dots, (t_n, m_n, \sigma_n)\}$, where t_i specifies the verification key against which σ_i is purported to be a signature on message m_i . We extend the definitions of Bellare, Garay and Rabin [4] to deal with multiple signers. And this is an important point that wasn't a concern with only a single signer: *one or more of the signers may be maliciously colluding*.

Definition 2.1 (Batch Verification of Signatures) *Let ℓ be the security parameter. Suppose $(\text{Gen}, \text{Sign}, \text{Verify})$ is a signature scheme, $k, n \in \text{poly}(\ell)$, and $(pk_1, sk_1), \dots, (pk_k, sk_k)$ are generated independently according to $\text{Gen}(1^\ell)$. Let $PK = \{pk_1, \dots, pk_k\}$. Then we call probabilistic Batch a batch verification algorithm when the following conditions hold:*

- *If $pk_{t_i} \in PK$ and $\text{Verify}(pk_{t_i}, m_i, \sigma_i) = 1$ for all $i \in [1, n]$, then $\text{Batch}((pk_{t_1}, m_1, \sigma_1), \dots, (pk_{t_n}, m_n, \sigma_n)) = 1$.*
- *If $pk_{t_i} \in PK$ for all $i \in [1, n]$ and $\text{Verify}(pk_{t_i}, m_i, \sigma_i) = 0$ for some $i \in [1, n]$, then $\text{Batch}((pk_{t_1}, m_1, \sigma_1), \dots, (pk_{t_n}, m_n, \sigma_n)) = 0$ except with probability negligible in ℓ , taken over the randomness of Batch.*

Note that Definition 2.1 requires that signing keys be generated honestly, but then they can be later held by an adversary. In practice, users could register their keys and prove some necessary properties of the keys at registration time [2].

On Differences between Batch Verification, Screening and Aggregate Signatures. As we discussed in the introduction, when doing our literature search on batch verification, we often came across works (e.g., [21, 23]) which confuse the goals of aggregate signatures and batch verification or claim to do batch verification when, in fact, they often meet a weaker guarantee called *screening* [4]. Let us clarify these distinct notions.

Informally, in both batch verification and screening, the goal is an algorithm that takes as input n distinct signatures and verifies them quickly. In batch verification, the batch of signatures should verify if and only if all individual signatures are valid. In the screening security model, an honest signer is protected in the sense that an attacker cannot cause her to become bound to a message that she did not sign, even if the attacker controls all other signers; however, honest verifiers are not totally protected from dishonest signers in the sense that a dishonest signer might be able to devise a batch of signatures that pass the screening test, but do not all individually verify.

The goal in aggregate signatures is an algorithm that takes as input n distinct signatures and compresses them to save bandwidth. It happens that the security definition of aggregate signatures [8] implies screening, while neither definition implies batch verification. We first give the formal definitions of screening and aggregate signatures, and then discuss a scheme which satisfies these notions, but *not* batch verification.

Definition 2.2 (Screening of Signatures) *Let ℓ be the security parameter. Suppose $(\text{Gen}, \text{Sign}, \text{Verify})$ is a signature scheme, $n \in \text{poly}(\ell)$ and $(pk_0, sk_0) \leftarrow \text{Gen}(1^\ell)$. Let $\mathcal{O}_{sk_0}(\cdot)$ be an oracle that on input m outputs $\sigma = \text{Sign}(sk_0, m)$. Then for all p.p.t. adversaries $\mathcal{A}$, we call probabilistic Screen a screening algorithm when $\mu(\ell)$ defined as follows is a negligible function:*

$$\begin{aligned} \Pr[(pk_0, sk_0) \leftarrow \text{Gen}(1^\ell), (pk_1, sk_1) \leftarrow \text{Gen}(1^\ell), \dots, (pk_n, sk_n) \leftarrow \text{Gen}(1^\ell), \\ D \leftarrow \mathcal{A}^{\mathcal{O}_{sk_0}(\cdot)}(pk_0, (pk_1, sk_1), \dots, (pk_n, sk_n)) : \\ \text{Screen}(D) = 1 \wedge (pk_0, m, \sigma) \in D \wedge m \notin Q] = \mu(\ell), \end{aligned}$$

where Q is the set of queries that $\mathcal{A}$ made to $\mathcal{O}_{sk_0}(\cdot)$ and for all $(pk_a, b, c) \in D$, $a \in \{0, \dots, n\}$.

The above definition is generalized to the multiple-signer case from the single-signer screening definition of Bellare, Garay and Rabin [4]. We now describe the security notion for aggregate signatures; the correctness property should be obvious.

Definition 2.3 (Aggregate Signatures Security [8]) *Let ℓ be the security parameter. Suppose $(\text{Gen}, \text{Sign}, \text{Verify})$ is a signature scheme, $n \in \text{poly}(\ell)$ and $(pk_0, sk_0) \leftarrow \text{Gen}(1^\ell)$. Let $\mathcal{O}_{sk_0}(\cdot)$ be an oracle that on input m outputs $\sigma = \text{Sign}(sk_0, m)$. Then for all p.p.t. adversaries $\mathcal{A}$, we call probabilistic AggVerify an aggregate-verification algorithm when $\mu(\ell)$ defined as follows is a negligible function:*

$$\begin{aligned} \Pr[(pk_0, sk_0) \leftarrow \text{Gen}(1^\ell); (pk_1, \dots, pk_n, m_0, \dots, m_n, \sigma) \leftarrow \mathcal{A}^{\mathcal{O}_{sk_0}(\cdot)}(pk_0) : \\ \text{AggVerify}((pk_0, \dots, pk_n), (m_0, \dots, m_n), \sigma) = 1 \wedge m_0 \notin Q] = \mu(\ell), \end{aligned}$$

where Q is the set of queries that $\mathcal{A}$ made to $\mathcal{O}_{sk_0}(\cdot)$.

As mentioned above, screening is the (maximum) guarantee that some aggregate signatures offer if one were to attempt to batch verify a group of signatures by first aggregating them together and then executing the aggregate-verification algorithm. We now give an example of a construction which can satisfy both Definitions 2.2 and 2.3, but which provably does not satisfy Definition 2.1. Consider the aggregate signature scheme of Boneh, Gentry, Lynn and Shacham [8] based on the BLS signatures [9]. First, we review the BLS signatures. Let $\mathbb{G} = \langle g \rangle$ be a group of prime order q that provided for a bilinear map $\mathbf{e} : \mathbb{G} \times \mathbb{G} \rightarrow \mathbb{G}_T$. To generate a key pair, choose a random $sk \in \mathbb{Z}_q$ and set $pk = g^{sk}$. A signature on message m is $\sigma = H(m)^{sk}$, where $H : \{0, 1\}^* \rightarrow \mathbb{G}$ is a hash function. To verify a signature σ on a message m , one checks that $\mathbf{e}(\sigma, g) = \mathbf{e}(H(m), pk)$. Given a group of message-signature pairs $(m_1, \sigma_1), \dots, (m_n, \sigma_n)$ (all purportedly from the same signer) where each message is distinct, BGLS aggregate them as $A = \prod_{i=1}^n \sigma_i$. Then all signatures can be verified in aggregate (i.e., screened) by testing that $\mathbf{e}(A, g) = \mathbf{e}(\prod_{i=1}^n H(m_i), pk)$. This scheme is *not*, however, a batch verification scheme since, for any $a \neq 1 \in \mathbb{G}$, the two *invalid* message-signature pairs $P_1 = (m_1, a \cdot H(m_1)^{sk})$ and $P_2 = (m_2, a^{-1} \cdot H(m_2)^{sk})$ will verify under Definition 2.2 (as BGLS observe [8]), but will not verify under Definition 2.1. Indeed, for some pervasive computing applications only guaranteeing screening would be disastrous, because only P_1 may be relevant information to forward to the next entity – and it won’t verify once it arrives! Also recall the e-mail scenario from Section 1. If we only did screening on the server, a user could send n messages with invalid signatures (to different receivers) that would screen correctly. The sender could then later claim that he did not send one of the messages and indeed the signature will not verify unless one can get hold of *all* n messages! To be fair, batch verification is not what aggregate schemes were designed to do.

Let’s make one final observation about the relationship between batch verification and screening. Let $D = \{(t_1, m_1, \sigma_1), \dots, (t_n, m_n, \sigma_n)\}$. We note that while $\text{Screen}(D) = 1$ does *not* guarantee that $\text{Verify}(pk_{t_i}, m_i, \sigma_i)$ for all i ; it does guarantee that the holder of sk_{t_i} authenticated m_i . That is, for all i , the holder of sk_{t_i} helped to create σ_i , which may or may not be a valid signature for m_i . Thus, a screening scheme can be employed to hold users accountable for the messages they “sign” in a set D such that $\text{Screen}(D) = 1$, but to do this the entire set D must be recorded or retransmitted to a third party. In the authenticated email scenario, where the mailserver is verifying the signatures on emails for many different users, releasing D (in the event of disputes) raises serious privacy issues. One could consider releasing a non-interactive zero-knowledge proof of knowledge of D such that $\text{Screen}(D) = 1$, although the naive approach will require $O(|D|)$ space and $O(|D|)$ time to verify.

3 Algebraic Setting and Group Membership

Bilinear Groups. Let BSetup be an algorithm that, on input the security parameter 1^ℓ , outputs the parameters for a bilinear map as $(q, g, \mathbb{G}, \mathbb{G}_T, \mathbf{e})$, where $\mathbb{G}$ and $\mathbb{G}_T$ are groups of prime order $q \in \Theta(2^\ell)$. The efficient mapping $\mathbf{e} : \mathbb{G} \times \mathbb{G} \rightarrow \mathbb{G}_T$ is both: (*bilinear*) for all $g \in \mathbb{G}$ and $a, b \leftarrow \mathbb{Z}_q$, $\mathbf{e}(g^a, g^b) = \mathbf{e}(g, g)^{ab}$; and (*non-degenerate*) if g generates $\mathbb{G}$, then $\mathbf{e}(g, g) \neq 1$. Following prior work, we write $\mathbb{G}$ and $\mathbb{G}_T$ in multiplicative notation (although $\mathbb{G}$ is often also denoted as an additive group). This bilinear map is called a *symmetric bilinear map*. A more general version of the bilinear map is the *asymmetric bilinear map* $\mathbf{e} : \mathbb{G}_1 \times \mathbb{G}_2 \rightarrow \mathbb{G}_T$, where $\mathbb{G}_1$ and $\mathbb{G}_2$ are distinct groups, possibly without efficient isomorphisms between them. Getting into details about how these bilinear maps are constructed is not the purpose of this paper, so we just give a very brief overview required for reasoning about the efficiency of our schemes.

$\mathbb{G}_1$ and $\mathbb{G}_2$ are groups of points on some curve and $\mathbb{G}_T$ is a subgroup of a multiplicative group over a related finite field. All groups have the same order q . Let E be an elliptic curve. We denote the group of points on E defined over $\mathbb{F}_p$ as $E(\mathbb{F}_p)$. $\mathbb{G}_1$ (or $\mathbb{G}$ in the symmetric setting) is a subgroup of $E(\mathbb{F}_p)$, $\mathbb{G}_2$ is usually defined as a subgroup of $E(\mathbb{F}_{p^k})$ where k is the embedding degree and $\mathbb{G}_T$ is usually a subgroup of $E(\mathbb{F}_{p^k}^*)$. Let's look at the size of the group elements. For simplicity we will assume that we are aiming for security comparable to 1024 bit RSA. Note that although a point (x, y) on a curve consist of two elements x and y in the underlying field, the size of groups elements are equivalent to the size of elements in the underlying field. The reason is that we only need to represent the x coordinate and the least significant bit of y in order to reconstruct y when needed.

So what is the minimum size the group elements can have? First of all the group order q must be large enough to resist attacks on discrete logarithms, such as the Pollard- ρ attack, which means that $q \geq 160$. Second, the MOV attack states that solving the discrete logarithm problem on a curve, reduces to solving it over the corresponding finite field [46], which means that the bitlength of p^k must be around 1024, which has implications for the size of $\mathbb{G}_T$. The best known curves in the symmetric setting works with $|p| = 512$ and $k = 2$, and hence elements of $\mathbb{G}$ will be 512 bits, while elements of $\mathbb{G}_T$ will be 1024 bits. In the asymmetric setting one can choose $|p| = 160$ and $k = 6$ which results in elements of size 160 bits in $\mathbb{G}_1$, while elements of $\mathbb{G}_2$ and $\mathbb{G}_T$ will be 960 bits. In some cases elements of $\mathbb{G}_2$ can be represented in the subfield $E(\mathbb{F}_{p^{k/2}})$ instead, resulting in elements of 480 bits [38].

Our constructions from Section 5 also work in the asymmetric setting which allows us to use a short representation of the signatures. The Π -IBS scheme from Section 4 can be modified to work in the asymmetric setting, but some parts of the signature will end up in the large group. We refer to the *efficiency note* paragraphs in Section 4 and 5 for a more detailed discussion.

Testing Membership in $\mathbb{G}$. In a *non-bilinear* setting, Boyd and Pavlovski [10] observed that the proofs of security for many previous batch verification or screening schemes *assumed* that the signatures (potentially submitted by a malicious adversary) were elements of an appropriate subgroup. For example, it was common place to assume that signatures submitted for batch DSA verification contained an element in a subgroup $\mathbb{G}$ of $\mathbb{Z}_p^*$ of prime order q . Boyd and Pavlovski [10] pointed out efficient attacks on many batching algorithms via exploiting this issue. Of course, group membership cannot be *assumed*, it must be *tested* and the work required by this test might well obliterate all batching efficiency gains. E.g., verifying that an element y is in $\mathbb{G}$ by testing if $y^q \bmod q = 1$; easily obliterates the gain of batching DSA signatures. Boyd and Pavlovski [10] suggest methods for overcoming this problem through careful choice of q .

In this paper, we will work in a bilinear setting, and we must be careful to avoid this common mistake in batch verification. Our proofs will require that elements of purported signatures are members of $\mathbb{G}$ and *not* $E(\mathbb{F}_p) \setminus \mathbb{G}$. The question is: how efficiently can this fact be verified? Determining whether some data represents a point on a curve is easy. The question is whether it is in the correct subgroup. Assume we have a bilinear map $e : \mathbb{G}_1 \times \mathbb{G}_2 \rightarrow \mathbb{G}_T$. In all the schemes we use, signatures are in $\mathbb{G}_1$, so this is the group we are interested in testing membership of.

If the order of $\mathbb{G}_1$ is q , one option is to verify that an element y is in $\mathbb{G}_1$ by checking that $y^q = 1$. While this might seem inefficient, it is actually not a problem in practice when working with pairing based schemes, since the time required for a single exponentiation is considerably less than the time required for computing a pairing. This has been verified experimentally by Ferrara et al. [25]. One area for improvement in batching, however, is to devise more efficient methods for

membership testing in bilinear groups. Chen, Cheng and Smart [20] provide more details on this.

Complexity Assumptions. In the coming sections, we will refer to the following complexity assumptions.

Assumption 3.1 (Computational Diffie-Hellman [24]) *Let g generate a group $\mathbb{G}$ of prime order $q \in \Theta(2^\ell)$. For all p.p.t. adversaries $\mathcal{A}$, the following probability is negligible in ℓ :*

$$\Pr[a, b, \leftarrow \mathbb{Z}_q; z \leftarrow \mathcal{A}(g, g^a, g^b) : z = g^{ab}].$$

Assumption 3.2 (Decisional Bilinear Diffie-Hellman [7]) *Let $\text{BSetup}(1^\ell) \rightarrow (q, g, \mathbb{G}, \mathbb{G}_T, \mathbf{e})$, where g generates $\mathbb{G}$. For all p.p.t. adversaries $\mathcal{A}$, the following probability is at most $1/2$ plus a negligible function in ℓ :*

$$\Pr[a, b, c, d \leftarrow \mathbb{Z}_q; x_0 \leftarrow \mathbf{e}(g, g)^{abc}; x_1 \leftarrow \mathbf{e}(g, g)^d; z \leftarrow \{0, 1\}; z' \leftarrow \mathcal{A}(g, g^a, g^b, g^c, x_z) : z = z'].$$

Assumption 3.3 (LRSW [45]) *Let $\text{BSetup}(1^\ell) \rightarrow (q, g, \mathbb{G}, \mathbb{G}_T, \mathbf{e})$. Let $X, Y \in \mathbb{G}$, $X = g^x$, and $Y = g^y$. Let $\mathcal{O}_{X,Y}(\cdot)$ be an oracle that, on input a value $m \in \mathbb{Z}_q^*$, outputs a triple $A = (a, a^y, a^{x+my})$ for a randomly chosen $a \in \mathbb{G}$. For all p.p.t. adversaries $\mathcal{A}^{(\cdot)}$, the following probability is negligible in ℓ :*

$$\begin{aligned} \Pr[(q, g, \mathbb{G}, \mathbb{G}_T, \mathbf{e}) \leftarrow \text{BSetup}(1^\ell); x \leftarrow \mathbb{Z}_q; y \leftarrow \mathbb{Z}_q; X = g^x; Y = g^y; \\ (m, a, b, c) \leftarrow \mathcal{A}^{\mathcal{O}_{X,Y}}(q, g, \mathbb{G}, \mathbb{G}_T, \mathbf{e}, X, Y) : m \notin Q \wedge m \in \mathbb{Z}_q^* \wedge \\ a \in \mathbb{G} \wedge b = a^y \wedge c = a^{x+my}] \end{aligned}$$

where Q is the set of queries that $\mathcal{A}$ made to $\mathcal{O}_{X,Y}(\cdot)$.

4 Batch Verification without Random Oracles

In this section, we present a method for batch verifying an identity-based signature scheme Π -IBS. This batch verification method can be executed in different modes, optimizing for the lowest runtime. Let n be the number of certificate/signature pairs, let 2^k be the number of users and let there be k bits per message. Let z be the additional security parameter required by the Π -IBS. Furthermore assume that the k bits are divided into z elements of k/z bits each. Then our batch verifier will verify n certificate/signature pairs with asymptotic complexity of the dominant operations roughly $\text{MIN}\{(2n + 3), (z + 3)\}$.

On the practical side, we note that as z grows there is a corresponding degradation in the concrete security of the IBS scheme (see [19] for a detailed discussion of these tradeoffs.) Setting $z = k/32$, however, seems a reasonable choice. Suppose we use SHA256 to hash all the messages ($k = 256$) and we choose the elements to be 32 bits ($k/z = 32$), then roughly when $n \geq 3$ batch verification becomes faster than individual verification.

4.1 Batch Verification for Π -IBS

We describe a batch verification algorithm for the Π -IBS scheme [19], where the number of pairings depends on the security parameter and not on the number of signatures and where no random oracles are necessary. The underlying Π -IBS signature scheme appears only implicitly in prior work, so let us clearly explain its origin. We begin with the observation by Boyen and Waters that an IBS scheme is realized by the key issuing algorithm of any (fully-secure) 2-level hierarchical identity-based encryption (HIBE) scheme [11].

In 2004, Boneh and Boyen described an efficient HIBE in the selective-ID security model [5]. In 2005, Waters described how to alter this scheme to make it fully-secure [53]. The IBS scheme that can be extracted from Waters 2-HIBE was proven secure under CDH in the standard model by Boyen and Waters [11]. In the conference version of this paper [12], we presented a batch verifier for this IBS scheme. Let n be the number of certificate/signature pairs, let 2^{k_1} be the number of users, and let k_2 be the bits per message. Then our batch verifier from the conference version can verify n certificate/signature pairs with asymptotic complexity of the dominant operations roughly $\text{MIN}\{(2n + 3), (k_1 + n + 3), (n + k_2 + 3), (k_1 + k_2 + 3)\}$. Suppose there are one billion users ($k_1 = 30$) and SHA256 is used to hash all the messages ($k_2 = 256$), then when $n \geq 31$ batching becomes faster than individual verification and at most 289 dominant operations will have to be performed regardless of n .

Fortunately, we are able to significantly improve the efficiency of these prior results. We begin by recalling that in 2005 Naccache [47] and Chatterjee and Sarkar [18] independently showed how to generalize the Waters IBE to optimize it for efficiency. In 2006 Chatterjee and Sarkar extended these ideas to Waters HIBE and the resulting HIBE was proven secure under DBDH in the standard model [19]. We call the IBS scheme implicitly defined by this generalized HIBE as Π -IBS. It is known to be secure under DBDH [19] and we conjecture that its security can be shown under CDH.

The Π -IBS scheme and its batch verification algorithm are both considerably more practical than the non-generalized version presented in our conference paper [12]. Indeed, the structure imposed by the generalization [47, 19] makes the Π -IBS scheme particularly well-suited for batch verification. We now explicitly describe the Π -IBS and then show how to batch verify these signatures.

We assume that the identities and messages are both bit-strings of length k represented by z blocks of k/z bits each. (If this is not the case, then let k be the larger bit-length and then pre-pad the shorter string with zeros.) Let $\text{BSetup}(1^\ell) \rightarrow (q, g, \mathbb{G}, \mathbb{G}_T, \mathbf{e})$.

Setup: First choose a secret $\alpha \in \mathbb{Z}_q$ and $h \in \mathbb{G}$ and calculate $A = \mathbf{e}(g, h)^\alpha$. Then pick two random integers $y'_1, y'_2 \in \mathbb{Z}_q$ and a random vector $y = (y_1, \dots, y_z) \in \mathbb{Z}_q^z$. The master secret key is $MK = h^\alpha$ and the public parameters are given as: $PP = (g, A, u'_1 = g^{y'_1}, u'_2 = g^{y'_2}, u_1 = g^{y_1}, \dots, u_z = g^{y_z})$.

We use the notation of Chatterjee and Sarkar [19] to define the following function. Let $v = (v_1, \dots, v_z)$, where each v_i is a (k/z) -bit string. For $i \in \{1, 2\}$, let:

$$U_i(v) = u'_i \prod_{j=1}^z u_j^{v_j}.$$

Extract: To create a private key for a user with identity $ID = (\kappa_1, \dots, \kappa_z)$, select $r \in \mathbb{Z}_q$ and return $K_{ID} = (h^\alpha \cdot U_1(ID)^r, g^{-r})$.

Sign: To sign a message $m = (m_1, \dots, m_z)$, where each m_i is a (k/z) -bit string, using private key $K = (K_1, K_2)$, select $s \in \mathbb{Z}_q$ and return

$$S = (K_1 \cdot U_2(m)^s, K_2, g^{-s}).$$

Verify: To verify a signature $S = (S_1, S_2, S_3)$ from identity $ID = \kappa_1, \dots, \kappa_z$ on message m , parse $m = (m_1, \dots, m_z)$, where each m_i is a (k/z) -bit string, and check that:

$$A = \mathbf{e}(S_1, g) \cdot \mathbf{e}(S_2, U_1(ID)) \cdot \mathbf{e}(S_3, U_2(m)).$$

If this equation holds, output *accept*; otherwise output *reject*.

We now introduce a batch verifier for this signature scheme. The basic idea is to adopt the small exponents test from [4] and to take advantage of the peculiarities of bilinear maps.

Batch Verify: Suppose we want to batch verify n purported signatures. Let κ_j^i and m_j^i denote the j 'th (k/z) -bit block of the identity of the i 'th signer and the message signed by the i 'th signer, respectively. Let $S^i = (S_1^i, S_2^i, S_3^i)$ denote the signature from the i 'th signer. First check that all the identities have the correct length and that $S_1^i, S_2^i, S_3^i \in \mathbb{G}$ for all i . If not; output *reject*. Otherwise generate a vector $\Delta = (\delta_1, \dots, \delta_n)$ where each δ_i is a random element of ℓ_b bits from $\mathbb{Z}_q$ and set

$$P = \mathbf{e}\left(\prod_{i=1}^n S_1^{i\delta_i}, g\right) \cdot \mathbf{e}\left(\prod_{i=1}^n S_2^{i\delta_i}, u'_1\right) \cdot \mathbf{e}\left(\prod_{i=1}^n S_3^{i\delta_i}, u'_2\right).$$

Depending on the values of z and n proceed as follows: if $n < 2z$ check whether

$$\prod_{i=1}^n A^{\delta_i} = P \cdot \prod_{i=1}^n \left(\mathbf{e}(S_2^{i\delta_i}, \prod_{j=1}^z u_j^{\kappa_j^i}) \cdot \mathbf{e}(S_3^{i\delta_i}, \prod_{j=1}^z u_j^{m_j^i}) \right) \quad (1)$$

holds, otherwise verify the equation

$$\prod_{i=1}^n A^{\delta_i} = P \cdot \prod_{j=1}^z \mathbf{e}\left(\prod_{i=1}^n (S_2^{i\kappa_j^i} \cdot S_3^{im_j^i})^{\delta_i}, u_j\right). \quad (2)$$

Output *accept* if the chosen equation holds; otherwise output *reject*.

Theorem 4.1 *The above algorithm is a batch verifier for the Π -IBS.*

Proof. Let $ID_i = (\kappa_1^i, \dots, \kappa_z^i)$. The requirement that all public keys are valid is trivially satisfied for an identity based scheme, once it has been verified that all identities have the correct length. First we show that $\text{Verify}(ID_1, M_1, S_1) = \dots = \text{Verify}(ID_n, M_n, S_n) = 1$ implies that $\text{Batch}((ID_1, M_1, S_1), \dots, (ID_n, M_n, S_n)) = 1$. This follows from the verification equation for the

Π -IBS scheme:

$$\prod_{i=1}^n A^{\delta_i} = \prod_{i=1}^n \left(\mathbf{e}(S_1^i, g) \cdot \mathbf{e}(S_2^i, U_1(ID_i)) \cdot \mathbf{e}(S_3^i, U_1(M_i)) \right)^{\delta_i} \quad (3)$$

$$\begin{aligned} &= \mathbf{e}\left(\prod_{i=1}^n S_1^{i\delta_i}, g\right) \cdot \prod_{i=1}^n \mathbf{e}(S_2^{i\delta_i}, u'_1 \prod_{j=1}^z u_j^{\kappa_j^i}) \cdot \prod_{i=1}^n \mathbf{e}(S_3^{i\delta_i}, u'_2 \prod_{j=1}^z u_j^{m_j^i}) \\ &= P \cdot \prod_{i=1}^n \left(\mathbf{e}(S_2^{i\delta_i}, \prod_{j=1}^z u_j^{\kappa_j^i}) \cdot \mathbf{e}(S_3^{i\delta_i}, \prod_{j=1}^z u_j^{m_j^i}) \right) . \end{aligned} \quad (4)$$

For the first part of the proof, all we need now is to show that equation 1 is equivalent to equation 2. Since for all i , $\text{Verify}(ID_i, M_i, S_i) = 1$, (S_1^i, S_2^i, S_3^i) are valid signatures and hence we can write $S_2^i = g^{b_i}$ and $S_3^i = g^{c_i}$ for some elements $b_i, c_i \in \mathbb{Z}_q$. Now we rewrite the part inside the parenthesis of equation 1 and get equation 2:

$$\begin{aligned} \prod_{i=1}^n \mathbf{e}(S_2^{i\delta_i}, \prod_{j=1}^z u_j^{\kappa_j^i}) \cdot \prod_{i=1}^n \mathbf{e}(S_3^{i\delta_i}, \prod_{j=1}^z u_j^{m_j^i}) &= \prod_{i=1}^n \left(\mathbf{e}(g^{b_i}, g^{\sum_{j=1}^z \kappa_j^i y_j}) \cdot \mathbf{e}(g^{c_i}, g^{\sum_{j=1}^z m_j^i y_j}) \right)^{\delta_i} \\ &= \prod_{i=1}^n \left(\mathbf{e}(g, g)^{\sum_{j=1}^z (\delta_i b_i \kappa_j^i y_j + \delta_i c_i m_j^i y_j)} \right) \\ &= \prod_{j=1}^z \left(\mathbf{e}(g, g)^{y_j \sum_{i=1}^n (\delta_i b_i \kappa_j^i + \delta_i c_i m_j^i)} \right) \\ &= \prod_{j=1}^z \mathbf{e}\left(\prod_{i=1}^n (S_2^{i\kappa_j^i} \cdot S_3^{im_j^i})^{\delta_i}, u_j\right) . \end{aligned}$$

We must now show the other direction. This proof is an application of the technique for proving the small exponents test in [4]. Batch verification accepts so we know that $S_1^i, S_2^i, S_3^i \in \mathbb{G}$ and hence we can write $S_1^i = g^{a_i}$, $S_2^i = g^{b_i}$ and $S_3^i = g^{c_i}$ for some $a_i, b_i, c_i \in \mathbb{Z}_q$. Also since $h \in \mathbb{G}$ we can write $h = g^d$ for some $d \in \mathbb{Z}_q$.

Since equation 3 is just an (inefficient) variant of the batch verification, we know that it holds, and we can rewrite it as:

$$\begin{aligned} \prod_{i=1}^n A^{\delta_i} &= \prod_{i=1}^n \left(\mathbf{e}(g^{a_i}, g) \cdot \mathbf{e}(g^{b_i}, g^{y'_1} g^{\sum_{j=1}^z y_j \kappa_j^i}) \cdot \mathbf{e}(g^{c_i}, g^{y'_2} g^{\sum_{j=1}^z y_j m_j^i}) \right)^{\delta_i} \\ &= \prod_{i=1}^n \mathbf{e}(g, g)^{\delta_i (a_i + b_i y'_1 + c_i y'_2 + b_i \sum_{j=1}^z y_j \kappa_j^i + c_i \sum_{j=1}^z y_j m_j^i)} \\ &= \mathbf{e}(g, h)^{\sum_{i=1}^n \delta_i d^{-1} (a_i + b_i y'_1 + c_i y'_2 + b_i \sum_{j=1}^z y_j \kappa_j^i + c_i \sum_{j=1}^z y_j m_j^i)} \\ &\Rightarrow \sum_{i=1}^n \delta_i \alpha - \sum_{i=1}^n \delta_i d^{-1} \left(a_i + b_i y'_1 + c_i y'_2 + b_i \sum_{j=1}^z y_j \kappa_j^i + c_i \sum_{j=1}^z y_j m_j^i \right) \equiv 0 \pmod{q} . \end{aligned}$$

Setting $\beta_i = \alpha - d^{-1} \left(a_i + b_i y'_1 + c_i y'_2 + b_i \sum_{j=1}^z y_j \kappa_j^i + c_i \sum_{j=1}^z y_j m_j^i \right)$ this can be written as:

$$\sum_{i=1}^n \delta_i \beta_i \equiv 0 \pmod{q} . \quad (5)$$

Assume that $\text{Batch}((ID_1, M_1, S_1), \dots, (ID_n, M_n, S_n)) = 1$, but for at least one i it is the case that $\text{Verify}(ID_i, M_i, S_i) = 0$. Assume wlog that this is true for $i = 1$, which means that $\beta_1 \neq 0$. Since q is a prime then β_1 has an inverse γ_1 such that $\beta_1 \gamma_1 \equiv 1 \pmod{q}$. This and equation 5 gives us:

$$\delta_1 \equiv -\gamma_1 \sum_{i=2}^n \delta_i \beta_i \pmod{q} . \quad (6)$$

Given (ID_i, M_i, S_i) , where $i = 1 \dots n$, let E be an event that $\text{Verify}(ID_1, M_1, S_1) = 0$ holds but that $\text{Batch}((ID_1, M_1, S_1), \dots, (ID_n, M_n, S_n)) = 1$, or in other words, that we break batch verification. Note that we do not make any assumptions about the remaining values. Let $\Delta' = \delta_2, \dots, \delta_n$ denote the last $n - 1$ values of Δ and let $|\Delta'|$ be the number of possible values for this vector. Equation 6 says that given a fixed vector Δ' there is exactly one value of δ_1 that will make event E happen, or in other words that the probability of E given a randomly chosen δ_1 is $\Pr[E|\Delta'] = 2^{-\ell_b}$. So if we pick δ_1 at random and sum over all possible choices of Δ' we get $\Pr[E] \leq \sum_{i=1}^{|\Delta'|} (\Pr[E|\Delta'] \cdot \Pr[\Delta'])$. Plugging in the values, we get: $\Pr[E] \leq \sum_{i=1}^{2^{\ell_b(n-1)}} (2^{-\ell_b} \cdot 2^{-\ell_b(n-1)}) = 2^{-\ell_b}$. $\square$

Efficiency Note. The signature for Π -IBS consists of three group elements, but since it is identity-based there is no public key, and we assume that the identity is given "for free" e.g., it could be the hardware address of the network interface card. Hence the size of the signature that verifies both the message and the identity depends only on the size of these group elements. We have described the scheme in the symmetric bilinear setting $\mathbf{e} : \mathbb{G} \times \mathbb{G} \rightarrow \mathbb{G}_T$ because the original scheme does not work in the asymmetric bilinear setting $\mathbf{e} : \mathbb{G}_1 \times \mathbb{G}_2 \rightarrow \mathbb{G}_T$. However, by switching the order of the elements in the first pairing and modifying the public parameters accordingly, the scheme also works in the asymmetric bilinear setting.

In the symmetric bilinear setting, elements must be around 512 bits for security comparable to 1024 bits RSA, which gives us a total signature size of 1536 bits. In the asymmetric bilinear setting the elements S_2 and S_3 can be represented using 160 bits, whereas S_1 needs 512 bits. So all in all we can represent the signature on the message and the identity using only 832 bits. However, it might not be efficient to test membership of the group $\mathbb{G}_2$, which is needed for batch verification.

5 Faster Batch Verification with Restrictions

In this section, we present a second method for batch verifying signatures together with their accompanying certificates. We propose using the BLS signature scheme [9] for the certificates and a modified version of the CL signature scheme [13] for signing messages. This method requires only two pairings to verify n certificates (from the same authority) and three pairings to verify n signatures (from possibly different signers). The cost for this significant efficiency gain is some usage restrictions, although as we will discuss, these restrictions may not be a problem for some of the applications we have in mind.

Certificates: We use a batch verifier for BLS signatures from the same authority as described in Section 5.1. The scheme is secure under CDH in the random oracle model. To verify n BLS certificates costs $n\text{-MultExpCost}_{\mathbb{G}}^2(\ell_b) + \text{PairCost}_{\mathbb{G},\mathbb{G}}^2 + \text{GroupTestCost}_{\mathbb{G}}^n + \text{HashCost}_{\mathbb{G}}^n$, using the Section 1.2 notation.

Signatures: We describe a new signature scheme $\Pi\text{-Sig}$ with a batch verifier in Section 5.2. The scheme is secure under the LRSW assumption in the plain model when the size of the message space is a polynomial and in the random oracle model when the size of the message space is super-polynomial. We assume that there are discrete time or location identifiers $\phi \in \Phi$. A user can issue at most one signature per ϕ (e.g., this might correspond to a device being allowed to broadcast at most one message every 300ms) and only signatures from the same ϕ can be batch verified together. To verify n $\Pi\text{-Sig}$ signatures, costs $n\text{-MultExpCost}_{\mathbb{G}}^2(\ell_b) + n\text{-MultExpCost}_{\mathbb{G}}(|w| + \ell_b) + \text{PairCost}_{\mathbb{G},\mathbb{G}}^3 + \text{GroupTestCost}_{\mathbb{G}}^n + \text{HashCost}_{\mathbb{G}}^n$, where w is the output of a hash function.

5.1 Batch Verification of BLS Signatures

We describe a batch verifier for *many signers* for the Boneh, Lynn, and Shacham signatures [9] described in Section 2, using the small exponents test [4], which requires distinct messages.

Batch Verify: Given purported signatures σ_i from n users on *distinct* messages M_i for $i = 1 \dots n$, first check that all public keys pk_i where $i \in [1, n]$ are valid, and that $\sigma_i \in \mathbb{G}$ for all i . If not; output *reject*. Otherwise compute $h_i = H(M_i)$ and generate a vector $\delta = (\delta_1, \dots, \delta_n)$ where each δ_i is a random element of ℓ_b bits from $\mathbb{Z}_q$. Check that $\mathbf{e}(\prod_{i=1}^n \sigma_i^{\delta_i}, g) = \prod_{i=1}^n \mathbf{e}(h_i, pk_i)^{\delta_i}$. If this equation holds, output *accept*; otherwise output *reject*.

Theorem 5.1 *The algorithm above is a batch verifier for BLS signatures.*

Proof. First we show that $\text{Verify}(pk_1, M_1, S_1) = \dots = \text{Verify}(pk_n, M_n, S_n) = 1$ implies that $\text{Batch}((pk_1, M_1, S_1), \dots, (pk_n, M_n, S_n)) = 1$. This follows from the verification equation for the BLS scheme:

$$\prod_{i=1}^n \mathbf{e}(\sigma_i, g)^{\delta_i} = \prod_{i=1}^n \mathbf{e}(h_i, pk_i)^{\delta_i} \Leftrightarrow \mathbf{e}(\prod_{i=1}^n \sigma_i^{\delta_i}, g) = \prod_{i=1}^n \mathbf{e}(h_i, pk_i)^{\delta_i} \quad (7)$$

We must now show the other direction. This proof is again an application of the technique for proving the small exponents test in [4]. Batch verification accepts so we know that $\sigma_i \in \mathbb{G}$ and hence we can write $\sigma_i = g^{c_i}$ for some $c_i \in \mathbb{Z}_q$. We also know that $h_i \in \mathbb{G}$ so we write it as $h_i = g^{r_i}$. Recall that $pk_i = g^{x_i}$. We know that equation 7 holds, so we can rewrite it as:

$$\begin{aligned} \prod_{i=1}^n \mathbf{e}(\sigma_i, g)^{\delta_i} &= \prod_{i=1}^n \mathbf{e}(h_i, pk_i)^{\delta_i} = \prod_{i=1}^n \mathbf{e}(g, g)^{\delta_i r_i x_i} \\ &\Rightarrow \mathbf{e}(g, g)^{\sum_{i=1}^n \delta_i c_i} = \mathbf{e}(g, g)^{\sum_{i=1}^n \delta_i r_i x_i} \\ &\Rightarrow \sum_{i=1}^n \delta_i c_i - \sum_{i=1}^n \delta_i r_i x_i \equiv 0 \pmod{q} \end{aligned}$$

Setting $\beta_i = c_i - r_i x_i$ this is equivalent to:

$$\sum_{i=1}^n \delta_i \beta_i \equiv 0 \pmod{q}$$

The rest of the proof follows from the last part of the proof of Theorem 4.1. $\square$

Single Signer for BLS. However, BLS [9] previously observed that if we have a single signer with public key v , the verification equation can be written as $\mathbf{e}(\prod_{i=1}^n \sigma_i^{\delta_i}, g) = \mathbf{e}(\prod_{i=1}^n h_i^{\delta_i}, v)$ which reduces the load to only two pairings.

Theorem 5.2 ([9]) *The algorithm above is a single-signer, batch verifier for BLS signatures.*

5.2 A New Signature Scheme Π -Sig

In this section we introduce a new signature scheme secure under the LRSW assumption [45], which is based on the Camenisch-Lysyanskaya signature scheme [13].

The Original CL Scheme. Recall the Camenisch and Lysyanskaya signature scheme [13]. Let $\text{BSetup}(1^\ell) \rightarrow (q, g, \mathbb{G}, \mathbb{G}_T, \mathbf{e})$. Choose the secret key $sk = (x, y) \in \mathbb{Z}_q^2$ at random and set $X = g^x$ and $Y = g^y$. The public key is $pk = (X, Y)$. To sign a message $m \in \mathbb{Z}_q^*$, choose a random $a \in \mathbb{G}$ and compute $b = a^y$, $c = a^x b^{xm}$. Output the signature (a, b, c) . To verify, check whether $\mathbf{e}(X, a) \cdot \mathbf{e}(X, b)^m = \mathbf{e}(g, c)$ and $\mathbf{e}(a, Y) = \mathbf{e}(g, b)$ holds.

Π -Sig: A version of the CL Scheme Allowing Batch Verification. Our goal is to batch-verify CL signatures made by different signers. That is, we need to consider how to verify equations of the form $\mathbf{e}(X, a) \cdot \mathbf{e}(X, b)^m = \mathbf{e}(g, c)$ and $\mathbf{e}(a, Y) = \mathbf{e}(g, b)$. The fact that the values X , a , b , and c are different for each signature seems to prevent efficient batch verification. Thus, we need to find a way such that many different signers share some of these values. Obviously, X and c need to be different. Now, depending on the application, all the signers can use the same value a by choosing a as the output of some hash function applied to, e.g., the current time period or location. We then note that all signers can use the same b in principle, i.e., have all of them share the same Y as it is sufficient for each signer to hold only one secret value (i.e., $sk = x$). Indeed, the only reason that the signer needs to know Y is to compute b . However, it turns out that if we define b such that $\log_a b$ is not known, the signature scheme is still secure. So, for instance, we can derive b in a similar way to a using a second hash function. Thus, all signers will virtually sign using the same Y per time period (but a different one for each period).

We note that the idea of sharing some value between the signers in order to efficiently perform some operation on the signatures is not new. Gentry and Ramzan present an identity based aggregate signature scheme [29] in which signatures can only be aggregated if all signers agree on some dummy message that none of them have used before.

Let us now describe the resulting scheme. Let $\text{BSetup}(1^\ell) \rightarrow (q, g, \mathbb{G}, \mathbb{G}_T, \mathbf{e})$. Let $\phi \in \Phi$ denote the current time period or location, where $|\Phi|$ is polynomial. Let $\mathcal{M}$ be the message space, for now let $\mathcal{M} = \{0, 1\}^*$. Let $H_1 : \Phi \rightarrow \mathbb{G}$, $H_2 : \Phi \rightarrow \mathbb{G}$, and $H_3 : \mathcal{M} \times \Phi \rightarrow \mathbb{Z}_q$ be different hash functions.

KeyGen: Choose a random $x \in \mathbb{Z}_q$ and set $X = g^x$. Set $sk = x$ and $pk = X$.

Sign: If this is the first call to Sign during period $\phi \in \Phi$, then on input message $m \in \mathcal{M}$, set $w = H_3(m||\phi)$, $a = H_1(\phi)$, $b = H_2(\phi)$ and output the signature $\sigma = a^x b^{xw}$. Otherwise, abort.

Verify: On input message-period pair (m, ϕ) and purported signature σ , compute $w = H_3(m||\phi)$, $a = H_1(\phi)$ and $b = H_2(\phi)$, and check that $\mathbf{e}(\sigma, g) = \mathbf{e}(a, X) \cdot \mathbf{e}(b, X)^w$. If true, output *accept*; otherwise output *reject*.

Theorem 5.3 *Under the LRSW assumption in $\mathbb{G}$, the Π -Sig signature scheme is existentially unforgeable in the random oracle model for message space $\mathcal{M} = \{0, 1\}^*$.*

Proof. We show that if there exists a p.p.t. adversary $\mathcal{A}$ that succeeds with probability ε in forging Π -Sig signatures, then we can construct a p.p.t. adversary $\mathcal{B}$ that solves the LRSW problem with probability $\varepsilon \cdot |\Phi|^{-1} \cdot q_H^{-1}$ in the random oracle model, where q_H is the maximum number of oracle queries $\mathcal{A}$ makes to H_3 during any period $\phi \in \Phi$. Recall that $|\Phi|$ is a polynomial. Adversary $\mathcal{B}^{\mathcal{O}_{X,Y}(\cdot)}$ against LRSW operates as follows on input $(q, g, \mathbb{G}, \mathbb{G}_T, \mathbf{e}, X, Y)$. Let ℓ be the security parameter. We assume that Φ is pre-defined. Let q_H be the maximum number of queries $\mathcal{A}$ makes to H_3 during any period $\phi \in \Phi$.

1. *Setup:* Send the bilinear parameters $(q, g, \mathbb{G}, \mathbb{G}_T, \mathbf{e})$ to $\mathcal{A}$. Choose a random $w' \in \mathcal{M}$ and query $\mathcal{O}_{X,Y}(w')$ to obtain an LRSW instance (w', a', b', c') . Choose a random $\phi' \in \Phi$. Treat H_1, H_2, H_3 as random oracles. Allow $\mathcal{A}$ access to the hash functions H_1, H_2, H_3 .
2. *Key Generation:* Set $pk^* = X$. For $i = 1$ to n , choose a random $sk_i \in \mathbb{Z}_q$ and set $pk_i = g^{sk_i}$. Output to $\mathcal{A}$ the keys pk^* and all (pk_i, sk_i) pairs.
3. *Oracle queries:* $\mathcal{B}$ responds to $\mathcal{A}$'s hash and signing queries as follows. Choose random r_i and s_i in $\mathbb{Z}_q$ for each time period (except ϕ'). Set up H_1 and H_2 such that:

$$H_1(\phi_i) = \begin{cases} g^{r_i} & \text{if } \phi_i \neq \phi'; \\ a' & \text{otherwise.} \end{cases} \quad (8)$$

and

$$H_2(\phi_i) = \begin{cases} g^{s_i} & \text{if } \phi_i \neq \phi'; \\ b' & \text{otherwise.} \end{cases} \quad (9)$$

Pick a random j in the range $[1, q_H]$. Choose random $t_{l,i} \in \mathbb{Z}_q$, such that $t_{l,i} \neq w'$, for $l \in [1, q_H]$ and $i \in [1, |\Phi|]$. Set up H_3 such that:

$$H_3(m_l||\phi_i) = \begin{cases} t_{l,i} & \text{if } \phi_i \neq \phi' \text{ or } l \neq j; \\ w' & \text{otherwise.} \end{cases} \quad (10)$$

$\mathcal{B}$ records $m^* := m_j$. Finally, set the signing query oracle such that on the l th query involving period ϕ_i :

$$\mathcal{O}_{sk^*}(m_l||\phi_i) = \begin{cases} \text{abort} & \text{if } \phi_i = \phi' \text{ and } l \neq j; \\ c' & \text{else if } \phi_i = \phi' \text{ and } l = j; \\ X^{r_i} X^{(s_i)t_{l,i}} & \text{otherwise.} \end{cases} \quad (11)$$

4. *Output:* At some point $\mathcal{A}$ stops and outputs a purported forgery $\sigma \in \mathbb{G}$ for some (m_l, ϕ_i) . If $\phi_i \neq \phi'$, $\mathcal{B}$ did not guess the correct period and thus $\mathcal{B}$ outputs a random guess for the LRSW game. If $m_l = m^*$, or the Π -Sig signature does not verify, $\mathcal{A}$'s output is not a valid forgery and thus $\mathcal{B}$ outputs a random guess for the LRSW game. Otherwise, $\mathcal{B}$ outputs $(t_{l,i}, a', b', \sigma)$ as the solution to the LRSW game.

We now analyze $\mathcal{B}$'s success. If $\mathcal{B}$ is not forced to abort or issue a random guess, then we note that $\sigma = H_1(\phi_i)^x H_2(\phi_i)^{x \cdot H_3(m_l || \phi_i)}$. In this scenario $\phi_i = \phi'$ and $t_{l,i} \neq w'$. We can substitute as $\sigma = (a')^x (b')^{x \cdot (t_{l,i})}$. Thus, we see that $(t_{l,i}, a', b', \sigma)$ is indeed a valid LRSW instance. Thus, $\mathcal{B}$ succeeds at LRSW whenever $\mathcal{A}$ succeeds in forging Π -Sig signatures, except when $\mathcal{B}$ is forced to abort or issue a random guess. First, when simulating the signing oracle, $\mathcal{B}$ is forced to abort whenever it incorrectly guesses which query to H_3 , during period ϕ' , $\mathcal{A}$ will eventually query to $\mathcal{O}_{sk^*}(\cdot, \cdot)$. Since all outputs of H_3 are independently random, $\mathcal{B}$ will be forced to abort at most q_H^{-1} probability. Next, provided that $\mathcal{A}$ issued a valid forgery, then $\mathcal{B}$ is only forced to issue a random guess when it incorrectly guesses which period $\phi \in \Phi$ that $\mathcal{A}$ will choose to issue its forgery. Since, from the view of $\mathcal{A}$ conditioned on the event that $\mathcal{B}$ has not yet aborted, all outputs of the oracles are perfectly distributed as either random oracles (H_1, H_2, H_3) or as a valid Π -Sig signer $(\mathcal{O}_{sk^*})$. Thus, this random guess is forced with probability at most $|\Phi|^{-1}$. Thus, if $\mathcal{A}$ succeeds with ε probability, then $\mathcal{B}$ succeeds with probability $\varepsilon \cdot |\Phi|^{-1} \cdot q_H^{-1}$. $\square$

On Removing the Random Oracles. In the previous proof, notice that we treated hash functions H_1, H_2 and H_3 as independent random oracles which were (statically) programmed in $|\Phi|$, $|\Phi|$, and $|\Phi| \cdot |\mathcal{M}|$ points, respectively, where Φ is the set of time period identifiers and $\mathcal{M}$ is the signing message space. Recall that, as before, $|\Phi|$ is restricted to be polynomial in the security parameter. Now, for sufficiently short message spaces, e.g., ISO defined error messages, we can replace all three random oracles in the security proof of Π -Sig by concrete hash functions. Suppose that given a set of pairs $(x_1, y_1), \dots, (x_k, y_k)$, it is possible to efficiently sample a function $H : \{0, 1\}^\ell \rightarrow \mathbb{G}$ (where $k < 2\ell + 1$) from a $(2\ell + 1)$ -independent function family $\mathcal{H}$ such that for each $H \in \mathcal{H}$, we have $H(x_i) = y_i$ for $i = 1$ to k . If such types of hash function families exist then we could simply constrain them exactly as we programmed our random oracles.

Fortunately, Canetti, Halevi, and Katz [14] describe a method of efficiently constructing such a hash function family which allows to map strings to bilinear map elements (or to map strings to elements in another prime-order algebraic group such as $\mathbb{Z}_q$). Any family satisfying the constraints above will work for our purposes, where H_1 and H_2 map into bilinear group $\mathbb{G}$ and H_3 maps into $\mathbb{Z}_q$. The construction remains as before and the new security proof simply uses concrete functions with constraints mirroring the points (statically) programmed in the oracles.

Lemma 5.4 *Under the LRSW assumption in $\mathbb{G}$, the Π -Sig signature scheme is existentially unforgeable in the plain model when $|\mathcal{M}|$ are polynomial in the security parameter.*

Batch Verification of Π -Sig Signatures. Batch verification of n signatures $\sigma_1, \dots, \sigma_n$ on messages $m_1, \dots, m_n$ for the same period ϕ can be done as follows. (Recall that each signer can issue at most one signature per time period. Thus, these n signatures are all from different signers.) Assume that user i with public key X_i signed message m_i . Set $w_i = H(m_i || \phi)$. First check that all public keys X_i where $i \in [1, n]$ are valid, and that $\sigma_i \in \mathbb{G}$ for all i . If not; output *reject*. Otherwise

pick a vector $\Delta = (\delta_1, \dots, \delta_n)$ with each element being a random ℓ_b -bit number and check that $\mathbf{e}(\prod_{i=1}^n \sigma_i^{\delta_i}, g) = \mathbf{e}(a, \prod_{i=1}^n X_i^{\delta_i}) \cdot \mathbf{e}(b, \prod_{i=1}^n X_i^{w_i \delta_i})$. If this equation holds, output *accept*; otherwise output *reject*.

Theorem 5.5 *The algorithm above is a batch verifier for Π -Sig signatures.*

Proof. First we show that $\text{Verify}(X_1, M_1, S_1) = \dots = \text{Verify}(X_n, M_n, S_n) = 1$ implies that $\text{Batch}((X_1, M_1, S_1), \dots, (X_n, M_n, S_n)) = 1$. This follows from the verification equation for the Π -Sig scheme if we keep in mind that

$$\begin{aligned} \prod_{i=1}^n \mathbf{e}(\sigma_i, g)^{\delta_i} &= \prod_{i=1}^n (\mathbf{e}(a, X_i) \cdot \mathbf{e}(b, X_i)^{w_i})^{\delta_i} = \prod_{i=1}^n \mathbf{e}(a, X_i)^{\delta_i} \cdot \prod_{i=1}^n \mathbf{e}(b, X_i)^{w_i \delta_i} \\ \Leftrightarrow \mathbf{e}(\prod_{i=1}^n \sigma_i^{\delta_i}, g) &= \mathbf{e}(a, \prod_{i=1}^n X_i^{\delta_i}) \cdot \mathbf{e}(b, \prod_{i=1}^n X_i^{w_i \delta_i}) . \end{aligned} \quad (12)$$

We must now show the other direction. This proof is again an application of the technique for proving the small exponents test in [4]. Batch verification accepts so we know that $\sigma_i \in \mathbb{G}$ and hence we can write $\sigma_i = g^{c_i}$ for some $c_i \in \mathbb{Z}_q$. We also know that a and b are in $\mathbb{G}$ so we write them as $a = g^r$ and $b = g^s$. Since equation 12 is just an (inefficient) variant of the batch verification, we know that it holds, and we can rewrite it as:

$$\begin{aligned} \prod_{i=1}^n \mathbf{e}(\sigma_i, g)^{\delta_i} &= \prod_{i=1}^n (\mathbf{e}(a, X_i) \cdot \mathbf{e}(b, X_i)^{w_i})^{\delta_i} = \prod_{i=1}^n \mathbf{e}(g, g)^{\delta_i (rx_i + sx_i w_i)} \\ &\Rightarrow \mathbf{e}(g, g)^{\sum_{i=1}^n \delta_i c_i} = \mathbf{e}(g, g)^{\sum_{i=1}^n \delta_i (rx_i + sx_i w_i)} \\ &\Rightarrow \sum_{i=1}^n \delta_i c_i - \sum_{i=1}^n \delta_i (rx_i + sx_i w_i) \equiv 0 \pmod{q} . \end{aligned}$$

Setting $\beta_i = c_i - (rx_i + sx_i w_i)$ this is equivalent to:

$$\sum_{i=1}^n \delta_i \beta_i \equiv 0 \pmod{q} .$$

The rest of the proof follows from the last part of the proof of Theorem 4.1. □

Π -Sig Without Batch Verification. So far we have described Π -Sig only as an efficient signature scheme to batch verify, but for completeness we note that if we are not interested in batch verification, Π -Sig is still a fairly efficient regular signature scheme without any restrictions.

KeyGen: Choose a random $x \in \mathbb{Z}_q$ and set $X = g^x$. Set $sk = x$ and $pk = X$.

Sign: Generate a value $\phi \in \Phi$ that has never been used by the signer before. Then on input message $m \in \mathcal{M}$, set $w = H_3(m||\phi)$, $a = H_1(\phi)$, $b = H_2(\phi)$, and $\sigma = a^x b^{xw}$ and output the signature (σ, ϕ) .

Verify: On input message m and purported signature (σ, ϕ) , compute $w = H_3(m||\phi)$, $a = H_1(\phi)$ and $b = H_2(\phi)$, and check that $\mathbf{e}(\sigma, g) = \mathbf{e}(ab^w, X)$. If true, output *accept*; otherwise output *reject*.

This is very similar to the original scheme. Note that the only change is that ϕ is now generated independently from all other signers and included as part of the signature, which makes the scheme unsuitable for batch verification (since the probability that many signers will share the same value of ϕ is small). However, now that we are only interested in individual verification, we can rewrite the original verification equation $\mathbf{e}(\sigma, g) = \mathbf{e}(a, X) \cdot \mathbf{e}(b, X)^w$ as $\mathbf{e}(\sigma, g) = \mathbf{e}(ab^w, X)$ which requires only two pairings to verify. Finally note that this variant of the verification equation does not depend on how ϕ was generated, and can always be used for individual verification if needed.

Efficiency Note. First, we observe that the Π -Sig signatures are *very* short, requiring only one element in $\mathbb{G}$. Since the BLS signatures also require only one element in $\mathbb{G}$, and since a public key for the Π -Sig scheme is also only one group element, the entire signature plus certificate could be transmitted in three $\mathbb{G}$ elements. In order to get the shortest representation for these elements, we need to use asymmetric bilinear maps $\mathbf{e} : \mathbb{G}_1 \times \mathbb{G}_2 \rightarrow \mathbb{G}_T$, where $\mathbb{G}_1 \neq \mathbb{G}_2$, which will allow elements in $\mathbb{G}_1$ to be 160 bits and elements of $\mathbb{G}_2$ to be 512 bits for a security level comparable to RSA-1024. For Π -Sig signatures we need to hash into $\mathbb{G}_1$ which according to Galbraith, Paterson and Smart can be done efficiently [27]. To summarize; using BLS and Π -Sig we can represent the signature plus certificate using approximately 832 bits with security comparable to RSA-1024, compared to around 3072 bits for actually using RSA-1024.

Second, suppose one uses the universal one-way hash functions described by Canetti, Halevi, and Katz [14] to remove the random oracles from Π -Sig. These hash functions require one exponentiation per constraint. In our case, we may require as many as $|\Phi| \cdot |\mathcal{M}|$ constraints. Thus, the cost to compute the hashes may dampen the efficiency gains of batch verification. However, our scheme will benefit from improvements in the construction of universal one-way hash functions with constraints.

If Π -Sig is used as a signatures scheme without an efficient batch verifier, the signature require one group element in $\mathbb{G}$ and one element in Φ , where the size of Φ only needs to be large enough to represent the number of times a user might want to sign with the same private key. Verification of a single Π -Sig signature requires two pairings.

6 Conclusions and Open Problems

In this paper we focused on batch verification of signatures. We overviewed the large body of existing work, almost exclusively dealing with single signers (Boneh, Lynn and Shacham [9] provide a batch verification scheme for multiple signers on the *same* message). We extended the general batch verification definition of Bellare, Garay and Rabin [4] to the case of multiple signers. We then presented, to our knowledge, the first efficient and practical batch verification scheme for signatures without random oracles. We focused on solutions that comprehended the time to verify the signature *and* the corresponding certificate for the verification key. First, we presented a batch verifier for the Π -IBS that can verify n signatures using only $z + 3$ pairings (the dominant operation), where identities are k bits divided into z elements, each of k/z bits. This is a significant improvement over the $3n$ pairings required by individual verification. Second, we presented a solution in the random oracle model that batch verifies n BLS certificates and n Π -Sig signatures

using only 5 pairings. Here, Π -Sig is a variant of the Camenisch-Lysyanskaya signatures that is much shorter, allows for efficient batch verification from many signers, but where only one signature can be safely issued per period.

It is an open problem to find a fast batch verification scheme for short signatures without the period restrictions from Section 5. Another exciting open problem is to develop fast batch verifiers for various forms of anonymous authentication such as group signatures, e-cash, and anonymous credentials.

Acknowledgments

We thank Ivan Damgård, Anna Lisa Ferrara, Jean-Pierre Hubaux, Panos Papadimitratos and the anonymous reviewers for their helpful input. Susan Hohenberger and Michael Østergaard Pedersen performed part of this research while at IBM Research, Zürich Research Laboratory. Also, Michael Østergaard Pedersen performed part of this research while at the University of Aarhus. Susan Hohenberger is sponsored by the Defense Advanced Research Projects Agency (DARPA) and the Air Force Research Laboratory (AFRL) under contract FA8750-11-2-0211, the Office of Naval Research under contract N00014-11-1-0470, NSF CAREER CNS-1053886, a Microsoft Faculty Fellowship and a Google Faculty Research Award. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the Defense Advanced Research Projects Agency or the US government.

References

- [1] Jee Hea An, Yevgeniy Dodis, and Tal Rabin. On the security of joint signature and encryption. In Lars R. Knudsen, editor, *Advances in Cryptology – EUROCRYPT ’02*, volume 2332 of *Lecture Notes in Computer Science*, pages 83–107. Springer, 2002.
- [2] Boaz Barak, Ran Canetti, Jesper Buus Nielsen, and Rafael Pass. Universally composable protocols with relaxed set-up assumptions. In *45th Symposium on Foundations of Computer Science (FOCS)*, pages 186–195. IEEE Computer Society, 2004.
- [3] Kenneth Barr and Krste Asanović. Energy aware lossless data compression. In *MobiSys*. USENIX, 2003.
- [4] Mihir Bellare, Juan A. Garay, and Tal Rabin. Fast batch verification for modular exponentiation and digital signatures. In Kaisa Nyberg, editor, *Advances in Cryptology – EUROCRYPT ’98*, volume 1403 of *Lecture Notes in Computer Science*, pages 236–250. Springer, 1998.
- [5] Dan Boneh and Xavier Boyen. Efficient selective-ID secure identity-based encryption without random oracles. In Christian Cachin and Jan Camenisch, editors, *Advances in Cryptology – EUROCRYPT ’04*, volume 3027 of *Lecture Notes in Computer Science*, pages 223–238. Springer, 2004.
- [6] Dan Boneh and Xavier Boyen. Short signatures without random oracles. In Christian Cachin and Jan Camenisch, editors, *Advances in Cryptology – EUROCRYPT ’04*, volume 3027 of *Lecture Notes in Computer Science*, pages 382–400. Springer, 2004.

- [7] Dan Boneh and Matthew K. Franklin. Identity-based encryption from the Weil pairing. In Joe Kilian, editor, *Advances in Cryptology – CRYPTO '01*, volume 2139 of *Lecture Notes in Computer Science*, pages 213–229. Springer, 2001.
- [8] Dan Boneh, Craig Gentry, Ben Lynn, and Hovav Shacham. Aggregate and verifiably encrypted signatures from bilinear maps. In Eli Biham, editor, *Advances in Cryptology – EUROCRYPT '03*, volume 2656 of *Lecture Notes in Computer Science*, pages 416–432. Springer, 2003.
- [9] Dan Boneh, Ben Lynn, and Hovav Shacham. Short signatures from the Weil pairing. *Journal of Cryptology*, 17(4):297–319, 2004.
- [10] Colin Boyd and Chris Pavlovski. Attacking and repairing batch verification schemes. In Tatsuaki Okamoto, editor, *Advances in Cryptology – ASIACRYPT '00*, volume 1976 of *Lecture Notes in Computer Science*, pages 58–71. Springer, 2000.
- [11] Xavier Boyen and Brent Waters. Compact group signatures without random oracles. In Serge Vaudenay, editor, *Advances in Cryptology – EUROCRYPT '06*, volume 4004 of *Lecture Notes in Computer Science*, pages 427–444. Springer, 2006.
- [12] Jan Camenisch, Susan Hohenberger, and Michael Østergaard Pedersen. Batch verification of short signatures. In Moni Naor, editor, *Advances in Cryptology – EUROCRYPT '07*, volume 4515 of *Lecture Notes in Computer Science*, pages 246–263. Springer, 2007.
- [13] Jan Camenisch and Anna Lysyanskaya. Signature schemes and anonymous credentials from bilinear maps. In Matthew K. Franklin, editor, *Advances in Cryptology – CRYPTO '04*, volume 3152 of *Lecture Notes in Computer Science*, pages 56–72. Springer, 2004.
- [14] Ran Canetti, Shai Halevi, and Jonathan Katz. A forward-secure public-key encryption scheme. In Eli Biham, editor, *Advances in Cryptology – EUROCRYPT '03*, volume 2656 of *Lecture Notes in Computer Science*, pages 255–271. Springer, 2003.
- [15] Tianjie Cao, Dongdai Lin, and Rui Xue. Security analysis of some batch verifying signatures from pairings. *International Journal of Network Security*, 3(2):138–143, 2006.
- [16] Car 2 Car. Communication consortium. <http://car-to-car.org>.
- [17] Jae Choon Cha and Jung Hee Cheon. An identity-based signature from gap Diffie-Hellman groups. In Yvo Desmedt, editor, *6th Public Key Cryptography (PKC)*, volume 2567 of *Lecture Notes in Computer Science*, pages 18–30. Springer, 2003.
- [18] Sanjit Chatterjee and Palash Sarkar. Trading time for space: Towards an efficient IBE scheme with short(er) public parameters in the standard model. In Dongho Won and Seungjoo Kim, editors, *8th Information Security and Cryptology (ICISC)*, volume 3935 of *Lecture Notes in Computer Science*, pages 424–440. Springer, 2005.
- [19] Sanjit Chatterjee and Palash Sarkar. HIBE with short public parameters without random oracle. In Xuejia Lai, editor, *Advances in Cryptology – ASIACRYPT '06*, volume 4284 of *Lecture Notes in Computer Science*, pages 145–160. Springer, 2006.
- [20] L. Chen, Z. Cheng, and N.P. Smart. Identity-based key agreement protocols from pairings, 2006. Cryptology ePrint Archive: Report 2006/199.

- [21] Jung Hee Cheon, Yongdae Kim, and Hyo Jin Yoon. A new ID-based signature with batch verification, 2004. Cryptology ePrint Archive: Report 2004/131.
- [22] Jung Hee Cheon and Dong Hoon Lee. Use of sparse and/or complex exponents in batch verification of exponentiations. *IEEE Transactions on Computers*, 55(12):1536–1542, January 2006.
- [23] Shi Cui, Pu Duan, and Choong Wah Chan. An efficient identity-based signature scheme with batch verifications. In Abdur Chowdhury, Francis Lau, and Frank Zhigang Wang, editors, *1st International Conference on Scalable Information Systems (InfoScale)*. ACM Press, 2006.
- [24] Whitfield Diffie and Martin Hellman. New directions in cryptography. *IEEE Transactions on Information Theory*, 22:644–654, 1976.
- [25] Anna Lisa Ferrara, Matthew Green, Susan Hohenberger, and Michael Østergaard Pedersen. Practical short signature batch verification, 2008. Cryptology ePrint Archive: Report 2008/015.
- [26] Amos Fiat. Batch RSA. In Gilles Brassard, editor, *Advances in Cryptology – CRYPTO ’89*, volume 435 of *Lecture Notes in Computer Science*, pages 175–185. Springer, 1989.
- [27] Steven D. Galbraith, Kenneth G. Paterson, and Nigel P. Smart. Pairings for cryptographers, 2006. Cryptology ePrint Archive: Report 2006/165.
- [28] Craig Gentry. How to compress Rabin ciphertexts and signatures (and more). In Matthew K. Franklin, editor, *Advances in Cryptology – CRYPTO ’04*, volume 3152 of *Lecture Notes in Computer Science*, pages 179–200. Springer, 2004.
- [29] Craig Gentry and Zulfikar Ramzan. Identity-based aggregate signatures. In Moti Yung, editor, *9th Public Key Cryptography (PKC)*, volume 3958 of *Lecture Notes in Computer Science*, pages 257–273. Springer, 2006.
- [30] Shafi Goldwasser, Silvio Micali, and Ronald L. Rivest. A digital signature scheme secure against adaptive chosen-message attacks. *SIAM J. Computing*, 17(2), 1988.
- [31] R. Granger and N.P. Smart. On computing products of pairings, 2006. Cryptology ePrint Archive: Report 2006/172.
- [32] Lein Harn. Batch verifying multiple DSA digital signatures. *Electronics Letters*, 34(9):870–871, 1998.
- [33] Lein Harn. Batch verifying multiple RSA digital signatures. *Electronics Letters*, 34(12):1219–1220, 1998.
- [34] Fumitaka Hoshino, Masayuki Abe, and Tetsutaro Kobayashi. Lenient/strict batch verification in several groups. In George I. Davida and Yair Frankel, editors, *4th Information Security*, volume 2200 of *Lecture Notes in Computer Science*, pages 81–94. Springer, 2001.
- [35] Min-Shiang Hwang, Cheng-Chi Lee, and Yuan-Liang Tang. Two simple batch verifying multiple digital signatures. In Sihan Qing, Tatsuaki Okamoto, and Jianying Zhou, editors, *3rd Information and Communications Security (ICICS)*, volume 2229 of *Lecture Notes in Computer Science*, pages 233–237. Springer, 2001.

- [36] Min-Shiang Hwang, Iuon-Chang Lin, and Kuo-Feng Hwang. Cryptanalysis of the batch verifying multiple RSA digital signatures. *Informatica, Lithuanian Academy of Sciences*, 11(1):15–19, 2000.
- [37] IEEE. 5.9 GHz Dedicated Short Range Communications. <http://grouper.ieee.org/groups/scc32/dsrc>.
- [38] Neal Koblitz and Alfred Menezes. Pairing-based cryptography at high security levels, 2005. Cryptology ePrint Archive: Report 2005/076.
- [39] Chi-Sung Lai and Sung-Ming Yen. Improved digital signature suitable for batch verification. *IEEE Transactions on Computers*, 44(7):957–959, 1995.
- [40] Olaf Landsiedel, Klaus Wehrle, and Stefan Götz. Accurate prediction of power consumption in sensor networks. In *IEEE Workshop on Embedded Networked Sensors (EmNetS-II)*, 2005.
- [41] Laurie Law and Brian J. Matt. Finding invalid signatures in pairing-based batches. In Steven D. Galbraith, editor, *Cryptography and Coding, 11th IMA International Conference*, volume 4887 of *Lecture Notes in Computer Science*, pages 34–53. Springer, 2007.
- [42] Seungwon Lee, Seongje Cho, Jongmoo Choi, and Yookun Cho. Efficient identification of bad signatures in RSA-type batch signature. *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences*, E89-A(1):74–80, 2006.
- [43] C. Lim and P. Lee. Security of interactive DSA batch verification. In *Electronics Letters*, volume 30(19), pages 1592–1593, 1994.
- [44] Chae Hoon Lim. Efficient multi-exponentiation and application to batch verification of digital signatures, 2000. http://dasan.sejong.ac.kr/~chlim/english_pub.html.
- [45] Anna Lysyanskaya, Ronald L. Rivest, Amit Sahai, and Stefan Wolf. Pseudonym systems. In Carlisle Adams and Howard Heys, editors, *6th Selected Areas in Cryptography (SAC)*, volume 1758 of *Lecture Notes in Computer Science*, pages 184–199. Springer, 1999.
- [46] Alfred Menezes, Scott Vanstone, and Tatsuaki Okamoto. Reducing elliptic curve logarithms to logarithms in a finite field. In *23rd ACM Symposium on Theory of Computing (STOC)*, pages 80–89, 1991.
- [47] D. Naccache. Secure and practical identity-based encryption, 2005. Cryptology ePrint Archive: Report 2005/369.
- [48] David Naccache, David M’Raïhi, Serge Vaudenay, and Dan Rappaport. Can DSA be improved? complexity trade-offs with the digital signature standard. In Alfredo De Santis, editor, *Advances in Cryptology – EUROCRYPT ’94*, volume 950 of *Lecture Notes in Computer Science*, pages 77–85. Springer, 1994.
- [49] Maxim Raya and Jean-Pierre Hubaux. Securing vehicular ad hoc networks. *Journal of Computer Security*, 15:39–68, 2007.
- [50] SeVeCom. Security on the road. <http://www.sevecom.org>.

- [51] Hovav Shacham and Dan Boneh. Improving SSL handshake performance via batching. In David Naccache, editor, *Cryptographer's Track at RSA Conference '01*, volume 2020 of *Lecture Notes in Computer Science*, pages 28–43. Springer, 2001.
- [52] Martin Stanek. Attacking LCCC batch verification of RSA signatures, 2006. Cryptology ePrint Archive: Report 2006/111.
- [53] Brent Waters. Efficient identity-based encryption without random oracles. In Ronald Cramer, editor, *Advances in Cryptology – EUROCRYPT '05*, volume 3494 of *Lecture Notes in Computer Science*, pages 320–329. Springer, 2005.
- [54] HyoJin Yoon, Jung Hee Cheon, and Yongdae Kim. Batch verifications with ID-based signatures. In Choonsik Park and Seongtaek Chee, editors, *7th Information Security and Cryptology (ICISC)*, volume 3506 of *Lecture Notes in Computer Science*, pages 233–248. Springer, 2004.
- [55] Fangguo Zhang and Kwangjo Kim. Efficient ID-based blind signature and proxy signature from bilinear pairings. In Reihaneh Safavi-Naini and Jennifer Seberry, editors, *8th Information Security and Privacy, Australasian Conference (ACISP)*, volume 2727 of *Lecture Notes in Computer Science*, pages 312–323. Springer, 2003.
- [56] Fangguo Zhang, Reihaneh Safavi-Naini, and Willy Susilo. Efficient verifiably encrypted signature and partially blind signature from bilinear pairings. In Thomas Johansson and Subhamoy Maitra, editors, *Progress in Cryptology – INDOCRYPT '03*, volume 2904 of *Lecture Notes in Computer Science*, pages 191–204. Springer, 2003.

Outsourcing the Decryption of ABE Ciphertexts

Matthew Green
Johns Hopkins University

Susan Hohenberger*
Johns Hopkins University

Brent Waters†
University of Texas at Austin

Abstract

Attribute-based encryption (ABE) is a new vision for public key encryption that allows users to encrypt and decrypt messages based on user attributes. For example, a user can create a ciphertext that can be decrypted only by other users with attributes satisfying (“Faculty” OR (“PhD Student” AND “Quads Completed”)). Given its expressiveness, ABE is currently being considered for many cloud storage and computing applications. However, one of the main efficiency drawbacks of ABE is that the size of the ciphertext and the time required to decrypt it grows with the complexity of the access formula.

In this work, we propose a new paradigm for ABE that largely eliminates this overhead for users. Suppose that ABE ciphertexts are stored in the cloud. We show how a user can provide the cloud with a *single* transformation key that allows the cloud to translate *any* ABE ciphertext satisfied by that user’s attributes into a (constant-size) El Gamal-style ciphertext, without the cloud being able to read any part of the user’s messages.

To precisely define and demonstrate the advantages of this approach, we provide new security definitions for both CPA and replayable CCA security with outsourcing, several new constructions, an implementation of our algorithms and detailed performance measurements. In a typical configuration, the user saves significantly on both bandwidth and decryption time, without increasing the number of transmissions.

*Supported by NSF CAREER CNS-1053886, DARPA PROCEED, Air Force Research Laboratory, Office of Naval Research N00014-11-1-0470, a Microsoft Faculty Fellowship and a Google Faculty Research Award. The views expressed are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.

†Supported by NSF CNS-0915361 and CNS-0952692, AFOSR Grant No: FA9550-08-1-0352, DARPA PROCEED, DARPA N11AP20006, Google Faculty Research Award, the Alfred P. Sloan Fellowship, and Microsoft Faculty Fellowship. The views expressed are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.

1 Introduction

Traditionally, we have viewed encryption as a method for one user to encrypt data to another specific targeted party, such that only the target recipient can decrypt and read the message. However, in many applications a user might often wish to encrypt data according to some *policy* as opposed to specified set of users. Trying to realize such applications on top of a traditional public key mechanism poses a number of difficulties. For instance, a user encrypting data will need to have a mechanism which allows him to look up all parties that have access credentials or attributes that match his policy. These difficulties are compounded if a party’s credentials themselves might be sensitive (e.g., the set of users with a TOP SECRET clearance) or if a party gains credentials well after data is encrypted and stored.

To address these issues, a new vision of encryption was put forth by Sahai and Waters [38] called Attribute-Based Encryption (ABE). In an ABE system, a user will associate an encryption of a message M with an function $f(\cdot)$, representing an access policy associated with the decryption. A user with a secret key that represents their set of attributes (e.g., credentials) S and will be able to decrypt a ciphertext associated with function $f(\cdot)$ if and only if $f(S) = 1$. Since the introduction of ABE there have been several other works proposing different variants [24, 7, 14, 36, 23, 42, 15, 28, 35] extending both functionality and refining security proof techniques.¹

One property that all of these ABE systems have is that both the ciphertext size and time for decryption grow with the size of the access formula f . Roughly, current efficient ABE realizations are set in pairing-based groups where the ciphertexts require two group elements for every node in the formula and decryption will require

¹A more general concept of functional encryption [11] allows for more general functions to be computed on the encrypted data and encompasses work such as searching on encrypted data and predicate encryption [10, 2, 12, 39, 27].

Scheme	ABE Type	Security Level	Model	Full CT Size	Full Decrypt Ops	Out CT Size	Out Dec Ops
Waters [42]	CP	CPA	-	$ \mathbb{G}_T + (1 + 2\ell) \mathbb{G} $	$\leq (2 + \ell)P + 2\ell E_G$	-	-
§3.1	CP	CPA	-	$ \mathbb{G}_T + (1 + 2\ell) \mathbb{G} $	$\leq (2 + \ell)P + 2\ell E_G$	$2 \mathbb{G}_T $	E_T
§3.2	CP	RCCA	RO	$ \mathbb{G}_T + (1 + 2\ell) \mathbb{G} + k$	$\leq (2 + \ell)P + 2\ell E_G + 2E_T$	$2 \mathbb{G}_T + k$	$3E_T$
GPSW [24]	KP	CPA	-	$ \mathbb{G}_T + (1 + s) \mathbb{G} $	$\leq (1 + \ell)P + 2\ell E_G$	-	-
§4.1	KP	CPA	-	$ \mathbb{G}_T + (1 + s) \mathbb{G} $	$\leq (1 + \ell)P + 2\ell E_G$	$2 \mathbb{G}_T $	E_T
§4.2	KP	RCCA	RO	$ \mathbb{G}_T + (1 + s) \mathbb{G} + k$	$\leq (1 + \ell)P + 2\ell E_G + 2E_T$	$2 \mathbb{G}_T + k$	$3E_T$

Figure 1: Summary of ABE outsourcing results. Above s denotes the size of an attribute set, ℓ refers to an LSSS access structure with an $\ell \times n$ matrix, k is the message bit length in RCCA schemes, and P, E_G, E_T stand for the maximum time to compute a pairing, exponentiation in $\mathbb{G}$ and exponentiation in $\mathbb{G}_T$ respectively. We ignore non-dominant operations. All schemes are in the selective security setting. We discuss methods for moving to adaptive security in Section 5.1.

a pairing for each node in the satisfied formula. While conventional desktop computers should be able to handle such a task for typical formula sizes, this presents a significant challenge for users that manage and view private data on mobile devices where processors are often one to two orders of magnitude slower than their desktop counterparts and battery life is a persistent problem. Interestingly, in tandem there has emerged the ability for users to buy on-demand computing from cloud-based services such as Amazon’s EC2 and Microsoft’s Windows Azure.

Can cloud services be *securely* used to outsource decryption in Attribute-Based Encryption systems? A naive first approach would be for a user to simply hand over their secret key, SK, to the outsourcing service. The service could then simply decrypt all ciphertexts requested by the user and then transmit the decrypted data. However, this requires complete trust of the outsourcing service; using the secret key the outsourcing service could read any encrypted message intended for the user.

A second approach might be to leverage recent outsourcing techniques [20, 17] based on Gentry’s [21] fully homomorphic encryption system. These give outsourcing for general computations and importantly preserve the privacy of the inputs so that the decryption keys and messages can remain hidden. Unfortunately, the overhead for these systems is currently impractical. Gentry and Halevi [22] showed that even for weak security parameters one “bootstrapping” operation of the homomorphic operation would take at least 30 seconds on a high performance machine (and 30 minutes for the high security parameter). Since one such operation would only count for a small constant number of gates in the overall computation, this would need to be repeated many times to evaluate an ABE decryption using the methods above.

Closer to practice, we might leverage recent techniques on secure outsourcing of pairings [16]. These techniques allow a client to outsource a pairing operation to a server. However, the solutions presented in [16] still require the client to compute multiple exponentiations in the target group for every pairing it outsources. These ex-

ponentiations can be quite expensive and the work of the client will still be proportional to the size of the policy f . Moreover, every pairing operation in the original protocol will trigger four pairings to be done by the proxy. Thus, the total workload is increased by a factor of at least four from the original decryption algorithm, and the client’s bandwidth requirements may actually *increase*. Given these drawbacks, we aim for an ABE outsourcing system that is secure and imposes minimal overhead.

Our Contributions. We give new methods for efficiently and securely outsourcing decryption of ABE ciphertexts. The core change to outsourceable ABE systems is a modified Key Generation algorithm that produces two keys. The first key is a short El Gamal [19] type secret key that must be kept private by the user. The second is what we call a “transformation key”, TK, that is shared with a proxy (and can be publicly distributed). If the proxy then receives a ciphertext CT for a function f for which the user’s credentials satisfy, it is then able to use the key TK to transform CT into a simple and short El Gamal ciphertext CT’ of the same message encrypted under the user’s key SK. The user is then able to decrypt with one simple exponentiation. Our system is secure against any malicious proxy. Moreover, the computational effort of the proxy is no more than that used to decrypt a ciphertext in a standard ABE system.

To achieve our results, we create what we call a new key blinding technique. At a high level, the new outsourced key generation algorithm will first run a key generation algorithm from an existing bilinear map based ABE scheme such as [24, 42]. Then it will choose a blinding factor exponent $z \in \mathbb{Z}_p$ (for groups of prime order p) and raise all elements to $z^{-1} \pmod{p}$. This will produce the transformation key TK, while the blinding factor z can serve as the secret key.

We show that we are able to adapt our outsourcing techniques to both the “Ciphertext-Policy” (CP-ABE) and “Key-Policy” (KP-ABE) types of ABE systems.² To

²CP-ABE systems behave as we outlined above where a ciphertext



Figure 2: Illustration of how ABE ciphertexts are fetched today.

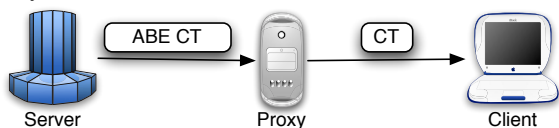


Figure 3: Outsourcing the Decryption: Illustration of how ABE ciphertexts could be transformed by a proxy into much shorter El Gamal-style ciphertexts.

achieve our KP-ABE and CP-ABE outsourcing systems we respectively apply our methodology to the constructions of Goyal et al. [24] and Waters [42]. To prove security of the systems we must show that they remain secure even in the presence of an attacker that acts as a user’s proxy. Our first systems and proofs model semantic security for an attacker that tries to eavesdrop on the user. We then extend our systems and proofs to chosen ciphertext attacks where the attack might query the user’s decryption routine on maliciously formed ciphertexts to compromise privacy. Our solutions in this setting apply the random oracle heuristic to achieve efficiency near the chosen plaintext versions.

Typical Usage Scenarios. We envision a typical usage scenario in Figures 2 and 3. Here a client sends a single transformation key *once* to the proxy, who can then retrieve (potentially large) ABE ciphertexts that the user is interested in and forward to her (small, constant-size) El Gamal-type ciphertexts. The proxy could be the client’s mail server, or the ciphertext server and the proxy could be the same entity, as in a cloud environment.

The savings in bandwidth and local computation time for the client are immediate: a transformed ciphertext is always smaller and faster to decrypt than an ABE ciphertext of [24, 42] (for any policy size). *We emphasize in this usage scenario that the number of transmissions will be the same as in the prior (non-outsourced) solutions.* Thus, the power consumption can only improve with faster computations and smaller transmissions.

Implementation and Evaluation. To evaluate our outsourcing systems, we implemented the CP-ABE version

is associated with a boolean access formula f and a user’s key is a set of attributes x , where a user can decrypt if $f(x) = 1$. KP-ABE is useful in applications where we want to have the mirror image semantics where the attributes x are associated with a ciphertext and an access formula f with the key.

and tested it in an outsourcing environment. Our implementation modified part of the libfenc [25] library, which includes a current CP-ABE implementation. We conducted our experiments on both an ARM-based mobile device and an Intel server to model the user device and proxy respectively.

Outsourcing decryption resulted in significant practical benefits. Decrypting on an ABE ciphertext containing 100 attributes, we found that without the use of a proxy the mobile device would require about 30 seconds of computation time and drain a significant amount of the device’s battery. When we applied our outsourcing technique, decrypting the ciphertext took 2 seconds on our Intel server and approximately 60 milliseconds on the mobile device itself.

To demonstrate compatibility with existing infrastructure, we constructed a re-usable platform for outsourcing decryption using the Amazon EC2 service. Our proxy is deployed as a public Amazon Machine Image that can be programmatically instantiated by any application requiring acceleration.

In addition to the core benefits of outsourcing, we discovered other collateral advantages. In existing ABE implementations [6, 25] much of the decryption code is dedicated to determining how a policy is satisfied by a key and executing the corresponding pairing computations of decryption. In our outsourcing solution, most of this code is pushed into the untrusted transformation algorithm, leaving only a much smaller portion on the user’s device. This has two advantages. First, the amount of decryption code that needs to reside on a resource constrained user device will be smaller. Actually, all bilinear map operations can be pushed outside. Second, this partitioning will dramatically decrease the size of the trusted code base, removing thousands of lines of complex parsing code. Even without using outsourcing, this partitioning of code is useful.

Related Work: Proxy Re-Encryption. In this work, we show how to delegate (in a true offline sense) the ability to transform an ABE ciphertext on message m into an El Gamal-style ciphertext on the same m , without learning anything about m . This is similar to the concept of *proxy re-encryption* [8, 4] where an untrusted proxy is given a re-encryption key that allows it to transform an encryption under Alice’s key of m into an encryption under Bob’s key of the same m , without allowing the proxy to learn anything about m .

2 Background

We first give the security definitions for ABE with outsourcing. We then give background information on bilinear maps. Finally, we provide formal definitions for

access structures and relevant background on Linear Secret Sharing Schemes (LSSS), as taken from [42].

Types of ABE. We consider two distinct varieties of Attribute-Based Encryption: Ciphertext-Policy (CP-ABE) and Key-Policy (KP-ABE). In CP-ABE an *access structure* (policy) is embedded into the ciphertext during encryption, and each decryption key is based on some attribute set S . KP-ABE inverts this relationship, embedding S into the ciphertext and a policy into the key.³ We capture both paradigms in a generalized ABE definition.

2.1 Access Structures

Definition 1 (Access Structure [5]) Let $\{P_1, P_2, \dots, P_n\}$ be a set of parties. A collection $\mathbb{A} \subseteq 2^{\{P_1, P_2, \dots, P_n\}}$ is *monotone* if $\forall B, C : \text{if } B \in \mathbb{A} \text{ and } B \subseteq C \text{ then } C \in \mathbb{A}$. An access structure (respectively, *monotone access structure*) is a collection (resp., *monotone collection*) $\mathbb{A}$ of non-empty subsets of $\{P_1, P_2, \dots, P_n\}$, i.e., $\mathbb{A} \subseteq 2^{\{P_1, P_2, \dots, P_n\}} \setminus \{\emptyset\}$. The sets in $\mathbb{A}$ are called the *authorized sets*, and the sets not in $\mathbb{A}$ are called the *unauthorized sets*.

In our context, the role of the parties is taken by the attributes. Thus, the access structure $\mathbb{A}$ will contain the authorized sets of attributes. We restrict our attention to monotone access structures. However, it is also possible to (inefficiently) realize general access structures using our techniques by defining the “not” of an attribute as a separate attribute altogether. Thus, the number of attributes in the system will be doubled. From now on, unless stated otherwise, by an access structure we mean a monotone access structure.

2.2 ABE with Outsourcing

Let S represent a set of attributes, and $\mathbb{A}$ an access structure. For generality, we will define (I_{enc}, I_{key}) as the inputs to the encryption and key generation function respectively. In a CP-ABE scheme $(I_{enc}, I_{key}) = (\mathbb{A}, S)$, while in a KP-ABE scheme we will have $(I_{enc}, I_{key}) = (S, \mathbb{A})$. A CP-ABE (resp. KP-ABE) scheme with outsourcing functionality consists of five algorithms:

Setup (λ, U) . The setup algorithm takes security parameter and attribute universe description as input. It outputs the public parameters PK and a master key MK.

Encrypt (PK, M, I_{enc}) . The encryption algorithm takes as input the public parameters PK, a message M , and an

access structure (resp. attribute set) I_{enc} . It outputs the ciphertext CT.

KeyGen $_{out}(MK, I_{key})$. The key generation algorithm takes as input the master key MK and an attribute set (resp. access structure) I_{key} and outputs a private key SK and a transformation key TK.

Transform (TK, CT) . The ciphertext transformation algorithm takes as input a transformation key TK for I_{key} and a ciphertext CT that was encrypted under I_{enc} . It outputs the partially decrypted ciphertext CT' if $S \in \mathbb{A}$ and the error symbol $\perp$ otherwise.

Decrypt $_{out}(SK, CT')$. The decryption algorithm takes as input a private key SK for I_{key} and a partially decrypted ciphertext CT' that was originally encrypted under I_{enc} . It outputs the message M if $S \in \mathbb{A}$ and the error symbol $\perp$ otherwise.⁴

Why RCCA security? We describe a security model for ABE that support outsourcing. We want a very strong notion of security. The traditional notion of security against adaptive chosen-ciphertext attacks (CCA) is a bit too strong since it does not allow any bit of the ciphertext to be altered, and the purpose of our outsourcing is to compress the size of the ciphertext. We thus adopt a relaxation due to Canetti, Krawczyk and Nielsen [13] called *replayable* CCA security, which allows modifications to the ciphertext provided they cannot change the underlying message in a meaningful way.

RCCA Security Model for ABE with Outsourcing. Figure 4 describes a generalized RCCA security game for both KP-ABE and CP-ABE schemes with outsourcing. We define the *advantage* of an adversary $\mathcal{A}$ in this game as $\Pr[b' = b] - \frac{1}{2}$.

Definition 2 (RCCA-Secure ABE with Outsourcing)

A CP-ABE or KP-ABE scheme with outsourcing is *RCCA-secure* (or *secure against replayable chosen-ciphertext attacks*) if all polynomial time adversaries have at most a negligible advantage in the RCCA game defined above.

CPA Security. We say that a system is *CPA-secure* (or *secure against chosen-plaintext attacks*) if we remove the Decrypt oracle in both Phase 1 and 2.

Selective Security. We say that a CP-ABE (resp. KP-ABE) system is *selectively* secure if we add an Init stage before Setup where the adversary commits to the challenge value I_{enc}^* .

³More intuitively, CP-ABE is often suggested as a means to implement role-based access control, where the user's key attributes correspond the long-term roles and ciphertexts carry an access policy. Key-Policy ABE is more appropriate in applications where ciphertexts may be tagged with attributes (e.g., relating to message content), and each user's access to these ciphertexts determined by a policy in their decryption key. For more on applications, see e.g., [37].

⁴Note that we can implement the standard (non-outsourced) ABE Decrypt algorithm by combining Transform and Decrypt $_{out}$.

Setup. The challenger runs the Setup algorithm and gives the public parameters, PK to the adversary.

Phase 1. The challenger initializes an empty table T , an empty set D and an integer $j = 0$. Proceeding adaptively, the adversary can repeatedly make any of the following queries:

- **Create(I_{key}):** The challenger sets $j := j + 1$. It runs the outsourced key generation algorithm on I_{key} to obtain the pair (SK, TK) and stores in table T the entry $(j, I_{key}, \text{SK}, \text{TK})$. It then returns to the adversary the transformation key TK.
Note: Create can be repeatedly queried with the same input.
- **Corrupt(i):** If there exists an i^{th} entry in table T , then the challenger obtains the entry $(i, I_{key}, \text{SK}, \text{TK})$ and sets $D := D \cup \{I_{key}\}$. It then returns to the adversary the private key SK. If no such entry exists, then it returns $\perp$.
- **Decrypt(i, CT):** If there exists an i^{th} entry in table T , then the challenger obtains the entry $(i, I_{key}, \text{SK}, \text{TK})$ and returns to the adversary the output of the decryption algorithm on input (SK, CT). If no such entry exists, then it returns $\perp$.

Challenge. The adversary submits two equal length messages M_0 and M_1 . In addition the adversary gives a value I_{enc}^* such that for all $I_{key} \in D$, $f(I_{key}, I_{enc}^*) \neq 1$. The challenger flips a random coin b , and encrypts M_b under I_{enc}^* . The resulting ciphertext CT^* is given to the adversary.

Phase 2. Phase 1 is repeated with the restrictions that the adversary cannot

- trivially obtain a private key for the challenge ciphertext. That is, it cannot issue a Corrupt query that would result in a value I_{key} which satisfies $f(I_{key}, I_{enc}^*) = 1$ being added to D .
- issue a trivial decryption query. That is, Decrypt queries will be answered as in Phase 1, except that if the response would be either M_0 or M_1 , then the challenger responds with the special message **test** instead.

Guess. The adversary outputs a guess b' of b .

Figure 4: Generalized RCCA Security game for CP- and KP-ABE with outsourcing functionality. For CP-ABE we define the function $f(I_{key}, I_{enc})$ as $f(S, \mathbb{A})$ and for KP-ABE it is defined as $f(\mathbb{A}, S)$. In either case the function f evaluates to 1 iff $S \in \mathbb{A}$.

2.3 Bilinear Maps

Let $\mathbb{G}$ and $\mathbb{G}_T$ be two multiplicative cyclic groups of prime order p . Let g be a generator of $\mathbb{G}$ and $e : \mathbb{G} \times \mathbb{G} \rightarrow \mathbb{G}_T$ be a bilinear map with the properties:

1. Bilinearity: for all $u, v \in \mathbb{G}$ and $a, b \in \mathbb{Z}_p$, we have $e(u^a, v^b) = e(u, v)^{ab}$.
2. Non-degeneracy: $e(g, g) \neq 1$.

We say that $\mathbb{G}$ is a bilinear group if the group operation in $\mathbb{G}$ and the bilinear map $e : \mathbb{G} \times \mathbb{G} \rightarrow \mathbb{G}_T$ are both efficiently computable.

The schemes we present in this work are provably secure under the Decisional Parallel BDHE Assumption [42] and the Decisional Bilinear Diffie-Hellman assumption (DBDH) [9] in bilinear groups. For reasons of space we will omit a definition of these assumptions here, and refer the reader to the cited works.

2.4 Linear Secret Sharing Schemes

We will make essential use of linear secret-sharing schemes. We adapt our definitions from those in [5]:

Definition 3 (Linear Secret-Sharing Schemes (LSSS))
A secret-sharing scheme Π over a set of parties $\mathcal{P}$ is called linear (over $\mathbb{Z}_p$) if

1. The shares of the parties form a vector over $\mathbb{Z}_p$.
2. There exists a matrix M with ℓ rows and n columns called the share-generating matrix for Π . There exists a function ρ which maps each row of the matrix to an associated party. That is for $i = 1, \dots, \ell$, the value $\rho(i)$ is the party associated with row i . When we consider the column vector $v = (s, r_2, \dots, r_n)$, where $s \in \mathbb{Z}_p$ is the secret to be shared, and $r_2, \dots, r_n \in \mathbb{Z}_p$ are randomly chosen, then Mv is the vector of ℓ shares of the secret s according to Π . The share $(Mv)_i$ belongs to party $\rho(i)$.

It is shown in [5] that every linear secret sharing-scheme according to the above definition also enjoys the

linear reconstruction property, defined as follows: Suppose that Π is an LSSS for the access structure $\mathbb{A}$. Let $S \in \mathbb{A}$ be any authorized set, and let $I \subset \{1, 2, \dots, \ell\}$ be defined as $I = \{i : \rho(i) \in S\}$. Then, there exist constants $\{\omega_i \in \mathbb{Z}_p\}_{i \in I}$ such that, if $\{\lambda_i\}$ are valid shares of any secret s according to Π , then $\sum_{i \in I} \omega_i \lambda_i = s$. It is shown in [5] that these constants $\{\omega_i\}$ can be found in time polynomial in the size of the share-generating matrix M .

Like any secret sharing scheme, it has the property that for any unauthorized set $S \notin \mathbb{A}$, the secret s should be information theoretically hidden from the parties in S .

Note on Convention. We use the convention that vector $(1, 0, 0, \dots, 0)$ is the “target” vector for any linear secret sharing scheme. For any satisfying set of rows I in M , we will have that the target vector is in the span of I .

For any unauthorized set of rows I the target vector is not in the span of the rows of the set I . Moreover, there will exist a vector w such that $w \cdot (1, 0, 0, \dots, 0) = -1$ and $w \cdot M_i = 0$ for all $i \in I$.

Using Access Trees. Some prior ABE works (e.g., [24]) described access formulas in terms of binary trees. Using standard techniques [5] one can convert any monotonic boolean formula into an LSSS representation. An access tree of ℓ nodes will result in an LSSS matrix of ℓ rows.

3 Outsourcing Decryption for Ciphertext-Policy ABE

3.1 A CPA-secure Construction

Our CP-ABE construction is based on the “large universe” construction of Waters [42], which was proven to be selectively CPA-secure under the Decisional q -parallel BDHE assumption for a challenge matrix of size $\ell^* \times n^*$, where $\ell^*, n^* \leq q$.⁵ The Setup, Encrypt and (non-outsourced) Decrypt algorithms are identical to [42]. To enable outsourcing we modify the KeyGen algorithm to output a transformation key. We also define a new Transform algorithm, and modify the decryption algorithm to handle outputs of Encrypt as well as Transform. We present the full construction in Figure 5.

Discussion. For generality, we defined the transformation key TK as being created by the master authority. However, we observe that our outsourcing approach above is actually backwards compatible with existing deployments of the Waters system. In particular, one can see that any existing user with her own Waters SK can create a corresponding outsourcing pair (SK', TK') by rerandomizing with a random value z .

⁵By “large universe”, we mean a system that allows for a super-polynomial number of attributes.

Theorem 3.1 *Suppose the large universe construction of Waters [42, Appendix C] is a selectively CPA-secure CP-ABE scheme. Then the CP-ABE scheme of Figure 5 is a selectively CPA-secure outsourcing scheme.*

Note that the Waters scheme of [42] was proven secure under the Decisional q -parallel BDHE assumption. Due to space constraints, we omit a proof of Theorem 3.1. However, we observe that the proof techniques are quite similar to those used for the RCCA-secure variant we present in the next section.

3.2 An RCCA-secure Construction

We now extend our CPA-secure system to achieve the stronger RCCA-security guarantee. To do so, we borrow some techniques from Fujisaki and Okamoto [18], who (roughly) showed how to transform a CPA-secure encryption scheme into a CCA-secure encryption scheme in the random oracle model. Here we relax to RCCA-security and have the additional challenge of preserving the decryption outsourcing capability.

The Setup and KeyGen algorithms operate exactly as in the CPA-secure scheme, except the public key additionally includes the description of hash functions $H_1 : \{0, 1\}^* \rightarrow \mathbb{Z}_p$ and $H_2 : \{0, 1\}^* \rightarrow \{0, 1\}^k$. We now describe the remaining algorithms.

Encrypt_{rcca}(PK, $\mathcal{M} \in \{0, 1\}^k, (M, \rho)$) The encryption algorithm selects a random $R \in \mathbb{G}_T$ and then computes $s = H_1(R, \mathcal{M})$ and $r = H_2(R)$. It then computes $(C_1, D_1), \dots, (C_\ell, D_\ell)$ as in the CPA-secure construction of Figure 5 (except that s is no longer chosen randomly as part of $\bar{v}$). The ciphertext is published as CT =

$$C = R \cdot e(g, g)^{\alpha s}, C' = g^s, C'' = \mathcal{M} \oplus r, \\ (C_1, D_1), \dots, (C_\ell, D_\ell)$$

along with a description of access structure (M, ρ) .

Transform_{rcca}(TK, CT). The transformation algorithm recovers the value $e(g, g)^{s\alpha/z}$ as before. It outputs the partially decrypted ciphertext CT' as $(C, C'', e(g, g)^{s\alpha/z})$.

Decrypt_{rcca}(SK, CT). The decryption algorithm takes as input a private key $SK = (z, TK)$ and a ciphertext CT. If the ciphertext is not partially decrypted, then the algorithm first executes Transform_{out}(TK, CT). If the output is $\perp$, then this algorithm outputs $\perp$ as well. Otherwise, it takes the ciphertext (T_0, T_1, T_2) and computes $R = T_0/T_2^z$, $\mathcal{M} = T_1 \oplus H_2(R)$, and $s = H_1(R, \mathcal{M})$. If $T_0 = R \cdot e(g, g)^{\alpha s}$ and $T_2 = e(g, g)^{\alpha s/z}$, it outputs $\mathcal{M}$; otherwise, it outputs the error symbol $\perp$.

Setup(λ, U). The setup algorithm takes as input a security parameter and a universe description U . To cover the most general case, we let $U = \{0, 1\}^*$. It then chooses a group $\mathbb{G}$ of prime order p , a generator g and a hash function F that maps $\{0, 1\}^*$ to $\mathbb{G}$.^a In addition, it chooses random exponents $\alpha, a \in \mathbb{Z}_p$. The authority sets $\text{MSK} = (g^\alpha, \text{PK})$ as the master secret key. It publishes the public parameters as:

$$\text{PK} = g, e(g, g)^\alpha, g^a, F$$

Encrypt($\text{PK}, \mathcal{M}, (M, \rho)$) The encryption algorithm takes as input the public parameters PK and a message $\mathcal{M}$ to encrypt. In addition, it takes as input an LSSS access structure (M, ρ) . The function ρ associates rows of M to attributes. Let M be an $\ell \times n$ matrix. The algorithm first chooses a random vector $\vec{v} = (s, y_2, \dots, y_n) \in \mathbb{Z}_p^n$. These values will be used to share the encryption exponent s . For $i = 1$ to ℓ , it calculates $\lambda_i = \vec{v} \cdot M_i$, where M_i is the vector corresponding to the i th row of M . In addition, the algorithm chooses random $r_1, \dots, r_\ell \in \mathbb{Z}_p$. The ciphertext is published as $\text{CT} =$

$$C = \mathcal{M} \cdot e(g, g)^{\alpha s}, C' = g^s, \\ (C_1 = g^{a\lambda_1} \cdot F(\rho(1))^{-r_1}, D_1 = g^{r_1}), \dots, (C_\ell = g^{a\lambda_\ell} \cdot F(\rho(\ell))^{-r_\ell}, D_\ell = g^{r_\ell})$$

along with a description of (M, ρ) .

KeyGen_{out}(MSK, S) The key generation algorithm runs $\text{KeyGen}(\text{MSK}, S)$ to obtain $\text{SK}' = (\text{PK}, K' = g^\alpha g^{at'}, L' = g^{t'}, \{K'_x = F(x)^{t'}\}_{x \in S})$. It chooses a random value $z \in \mathbb{Z}_p^*$. It sets the transformation key TK as

$$\text{PK}, \quad K = K'^{1/z} = g^{(\alpha/z)} g^{a(t'/z)} = g^{(\alpha/z)} g^{at}, \quad L = L'^{1/z} = g^{(t'/z)} = g^t, \quad \{K_x\}_{x \in S} = \{K'_x\}_{x \in S}^{1/z}$$

and the private key SK as (z, TK) .

Transform_{out}(TK, CT) The transformation algorithm takes as input a transformation key $\text{TK} = (\text{PK}, K, L, \{K_x\}_{x \in S})$ for a set S and a ciphertext $\text{CT} = (C, C', C_1, \dots, C_\ell)$ for access structure (M, ρ) . If S does not satisfy the access structure, it outputs $\perp$. Suppose that S satisfies the access structure and let $I \subset \{1, 2, \dots, \ell\}$ be defined as $I = \{i : \rho(i) \in S\}$. Then, let $\{\omega_i \in \mathbb{Z}_p\}_{i \in I}$ be a set of constants such that if $\{\lambda_i\}$ are valid shares of any secret s according to M , then $\sum_{i \in I} \omega_i \lambda_i = s$. The transformation algorithm computes

$$e(C', K) / \left(e(\prod_{i \in I} C_i^{\omega_i}, L) \cdot \prod_{i \in I} e(D_i^{\omega_i}, K_{\rho(i)}) \right) = \\ e(g, g)^{s\alpha/z} e(g, g)^{ast} / \left(\prod_{i \in I} e(g, g)^{ta\lambda_i \omega_i} \right) = e(g, g)^{s\alpha/z}$$

It outputs the partially decrypted ciphertext CT' as $(C, e(g, g)^{s\alpha/z})$, which can be viewed as the El Gamal ciphertext $(\mathcal{M} \cdot G^{zd}, G^d)$ where $G = e(g, g)^{1/z} \in \mathbb{G}_T$ and $d = s\alpha \in \mathbb{Z}_p$.

Decrypt_{out}(SK, CT) The decryption algorithm takes as input a private key $\text{SK} = (z, \text{TK})$ and a ciphertext CT . If the ciphertext is not partially decrypted, then the algorithm first executes $\text{Transform}_{\text{out}}(\text{TK}, \text{CT})$. If the output is $\perp$, then this algorithm outputs $\perp$ as well. Otherwise, it takes the ciphertext (T_0, T_1) and computes $T_0/T_1^z = \mathcal{M}$.

Notice that if the ciphertext is already partially decrypted for the user, then she need only compute one exponentiation and no pairings to recover the message.

^aSee Waters [42] for details on how to implement this hash in the standard model. For our purposes, one can think of F as a random oracle.

Figure 5: A CPA-secure CP-ABE outsourcing scheme based on the large-universe construction of Waters [42, Appendix C].

Theorem 3.2 *Suppose the large universe construction of Waters [42, Appendix C] is a selectively CPA-secure CP-ABE scheme. Then the outsourcing scheme above is selectively RCCA-secure in the random oracle model for large message spaces.*⁶

We present a proof of Theorem 3.2 in Appendix A.

4 Outsourcing Decryption for Key-Policy ABE

4.1 A CPA-secure Construction

We now present an outsourcing scheme based on the large universe KP-ABE construction due to Goyal, Pandey, Sahai and Waters [24].⁷ The Setup and Encrypt algorithms are identical to [24]. We modify KeyGen to output a transformation key, introduce a Transform algorithm, and then modify the decryption algorithm to handle outputs of Encrypt as well as Transform. The full construction is presented in Figure 6.

Theorem 4.1 *Suppose the GPSW KP-ABE scheme [24] is selectively CPA-secure. Then the KP-ABE scheme of Figure 6 is a selectively CPA-secure outsourcing scheme.*

Discussion. As in the previous construction, we defined the transformation key TK as being created by the master authority. We again note that our outsourcing approach above is actually backwards compatible with existing deployments of the GPSW system.

Due to restrictions on space, we leave the proof of security to the full version of this work [26].

4.2 An RCCA-secure construction

We now extend our above results, which only hold for CPA-security, to the stronger RCCA-security guarantee. Once again, we accomplish this using the techniques from Fujisaki and Okamoto [18]. The Setup and KeyGen algorithms operate exactly as before, except the public key additionally includes the value $e(g, h)^\alpha$ (which was already computable from existing values) and the description of hash functions $H_1 : \{0, 1\}^* \rightarrow \mathbb{Z}_p$ and $H_2 : \{0, 1\}^* \rightarrow \{0, 1\}^k$.

⁶The security of this scheme follows for large message spaces; e.g., k -bit spaces where $k \geq \lambda$, the security parameter. To obtain a secure scheme for smaller message spaces, replace C'' with any CPA-secure symmetric encryption of $\mathcal{M}$ using key $H_2(R)$ and let the range of H_2 be the key space of this symmetric scheme. Since the focus of this work is on efficiency, we'll typically be assuming large enough message spaces and therefore opting for the quicker XOR operation.

⁷This construction was originally described using access trees; here we generalize it to LSSS access structures.

Encrypt_{rcca}(PK, $\mathcal{M} \in \{0, 1\}^k, S$). The encryption algorithm chooses a random $R \in \mathbb{G}_T$. It then computes $s = H_1(R, \mathcal{M})$ and $r = H_2(R)$. For each $x \in S$ it generates C_x as in the CPA-secure scheme. The ciphertext is published as $CT =$

$$C = R \cdot e(g, h)^{\alpha s}, C' = g^s, C'' = r \oplus \mathcal{M}, \{C_x\}_{x \in S}$$

along with a description of S .

Transform_{rcca}(TK, CT). The transformation algorithm recovers the value $e(g, h)^{s\alpha/z}$ as before. It outputs the partially decrypted ciphertext CT' as $(C, C'', e(g, h)^{s\alpha/z})$.

Decrypt_{rcca}(SK, CT). The decryption algorithm takes as input a private key $SK = (z, TK)$ and a ciphertext CT. If the ciphertext is not partially decrypted, then the algorithm first executes **Transform_{out}**(TK, CT). If the output is $\perp$, then this algorithm outputs $\perp$ as well. Otherwise, it takes the ciphertext (T_0, T_1, T_2) and computes $R = T_0/T_2^z$, $\mathcal{M} = T_1 \oplus H_2(R)$, and $s = H_1(R, \mathcal{M})$. If $T_0 = R \cdot e(g, h)^{\alpha s}$ and $T_2 = e(g, h)^{\alpha s/z}$, it outputs $\mathcal{M}$; otherwise, it outputs the error symbol $\perp$.

Theorem 4.2 *Suppose the construction of GPSW [24] is a selectively CPA-secure KP-ABE scheme. Then the outsourcing scheme above is selectively RCCA-secure in the random oracle model for large message spaces.*

See the footnote on Theorem 3.2 for a definition and discussion of “large message spaces”. We present a proof of Theorem 4.2 in the full version [26] of this work.

5 Discussion

5.1 Achieving Adaptive Security

The systems we presented were proven secure in the selective model of security. We briefly sketch how we can adapt our techniques to achieve ABE systems that are provably secure in the adaptive model.⁸

Recently, the first ABE systems that achieved adaptive security were proposed by Lewko *et al.* [28] using the techniques of Dual System Encryption [41]. Since the underlying structure of the KP-ABE and CP-ABE schemes presented by Lewko *et al.* is almost respectively identical to the underlying Goyal *et al.* [24] and Waters [42] systems we use, it is possible to adapt our construction techniques to these underlying constructions.⁹

⁸We briefly note that it is simple to prove adaptive security of our schemes in the generic group model like Bethencourt, Sahai, and Waters [7]. Here we are interested in proofs under non-interactive assumptions.

⁹The main difference in terms of the constructions is that the systems proposed by Lewko *et al.* are set in composite order groups where the “core scheme” sits in one subgroup. The primary novelty of their work is in developing adaptive proofs of security for ABE systems.

Setup(λ, U). The setup algorithm takes as input a security parameter and a universe description U . To cover the most general case, we let $U = \{0, 1\}^*$. It then chooses a group $\mathbb{G}$ of prime order p , a generator g and a hash function F that maps $\{0, 1\}^*$ to $\mathbb{G}$.^a In addition, it chooses random values $\alpha \in \mathbb{Z}_p$ and $h \in \mathbb{G}$. The authority sets $\text{MSK} = (\alpha, \text{PK})$ as the master secret key. The public key is published as

$$\text{PK} = g, g^\alpha, h, F$$

Encrypt($\text{PK}, \mathcal{M}, S$). The encryption algorithm takes as input the public parameters PK , a message $\mathcal{M}$ to encrypt, and a set of attributes S . It chooses a random $s \in \mathbb{Z}_p$. The ciphertext is published as $\text{CT} = (S, C)$ where

$$C = \mathcal{M} \cdot e(g, h)^{\alpha s}, C' = g^s, \{C_x = F(x)^s\}_{x \in S}.$$

KeyGen_{out}($\text{MSK}, (M, \rho)$). Parse $\text{MSK} = (\alpha, \text{PK})$. The key generation algorithm runs $\text{KeyGen}((\alpha, \text{PK}), (M, \rho))$ to obtain $\text{SK}' = (\text{PK}, (D'_1 = h^{\lambda_1} \cdot F(\rho(1))^{r'_1}, R'_1 = g^{r'_1}), \dots, (D'_\ell, R'_\ell))$. Next, it chooses a random value $z \in \mathbb{Z}_p$, computes the transformation key TK as below, and outputs the private key as (z, TK) . Denoting r'_i/z as r_i , TK is computed as:

$$\text{PK}, (D_1 = D'_1^{1/z} = h^{\lambda_1/z} \cdot F(\rho(1))^{r_1}, R_1 = R'_1^{1/z} = g^{r_1}), \dots, (D_\ell = D'_\ell^{1/z}, R_\ell = R'_\ell^{1/z})$$

Transform_{out}(TK, CT). The transformation algorithm takes as input a transformation key $\text{TK} = (\text{PK}, (D_1, R_1), \dots, (D_\ell, R_\ell))$ for access structure (M, ρ) and a ciphertext $\text{CT} = (C, C', \{C_x\}_{x \in S})$ for set S . If S does not satisfy the access structure, it outputs $\perp$. Suppose that S satisfies the access structure and let $I \subset \{1, 2, \dots, \ell\}$ be defined as $I = \{i : \rho(i) \in S\}$. Then, let $\{\omega_i \in \mathbb{Z}_p\}_{i \in I}$ be a set of constants such that if $\{\lambda_i\}$ are valid shares of any secret s according to M , then $\sum_{i \in I} \omega_i \lambda_i = s$. The transformation algorithm computes

$$\begin{aligned} e(C', \prod_{i \in I} D_i^{\omega_i}) / \left(\prod_{i \in I} e(R_i, C_{\rho(i)}^{\omega_i}) \right) &= e(g^s, \prod_{i \in I} h^{\lambda_i \omega_i / z} \cdot F(\rho(i))^{r_i \omega_i}) / \left(\prod_{i \in I} e(g^{r_i}, F(\rho(i))^{s \omega_i}) \right) \\ &= e(g, h)^{s \alpha / z} \cdot \prod_{i \in I} e(g^s, F(\rho(i))^{r_i \omega_i}) / \left(\prod_{i \in I} e(g^{r_i}, F(\rho(i))^{s \omega_i}) \right) = e(g, h)^{s \alpha / z} \end{aligned}$$

It outputs the partially decrypted ciphertext CT' as $(C, e(g, h)^{s \alpha / z})$, which can be viewed as the El Gamal ciphertext $(\mathcal{M} \cdot G^{zd}, G^d)$ where $G = e(g, h)^{1/z} \in \mathbb{G}_T$ and $d = s \alpha \in \mathbb{Z}_p$.

Decrypt_{out}(SK, CT). The decryption algorithm takes as input a private key $\text{SK} = (z, \text{TK})$ and a ciphertext CT . If the ciphertext is not partially decrypted, then the algorithm first executes $\text{Transform}_{\text{out}}(\text{TK}, \text{CT})$. If the output is $\perp$, then this algorithm outputs $\perp$ as well. Otherwise, it takes the ciphertext (T_0, T_1) and computes $T_0/T_1^z = \mathcal{M}$.

^aGoyal *et al.* [24] give a standard model instantiation for F using an n -wise independent hash function (in the exponents) with the restriction that any ciphertext can contain at most n attributes. For our purposes, one can think of F as a random oracle.

Figure 6: A CPA-secure KP-ABE outsourcing scheme based on the large-universe construction of Goyal, Pandey, Sahai and Waters [24].

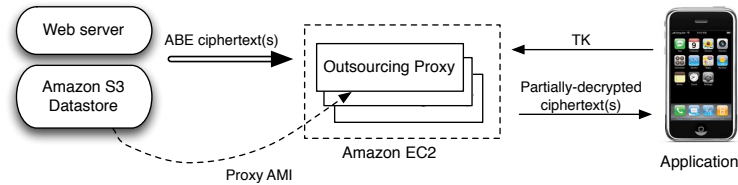


Figure 7: Architecture and data flow for our cloud-based outsourcing proxy. An application programmatically instantiates one or more instances of the outsourcing proxy, which is loaded from a public Amazon Machine Image (AMI) in the S3 storage cloud. Next the application uploads a transform key TK to the proxy, and subsequently instructs the proxy to obtain ciphertexts from remote web servers or from locations within the S3 storage cloud. The proxy transforms the ciphertexts and returns the partially-decrypted result to the application, which completes decryption to obtain a plaintext. We emphasize that the setup step including uploading the transformation key only needs to be done once; subsequently, many decryption steps can follow. In an alternative configuration (not shown) the application can also upload ABE ciphertexts to the proxy from its local storage. We note the first configuration conflates the ciphertext delivery and partial decryption and thus requires no additional transmissions relative to non outsourcing solutions. The alternative will require an round trip for each outsourcing operation.

One might hope that the proof of adaptive security could be a black box reduction to the adaptively secure schemes of Lewko *et al.* Unfortunately, this seems infeasible. Consider any direct black box reduction to the security of the underlying scheme. When the attacker makes a query to some transformation key, the reduction algorithm has two options. First, it could ask the security game for the underlying ABE system for a private key. Yet, it might turn out that the key both is never corrupted and is capable of decryption for the eventual challenge ciphertext. In this case the simulator will have to abort. A second option is for the reduction algorithm not to ask for such a key, but fill in the transformation key itself. However, if that user's key is later corrupted it will be difficult for the reduction to both ask for such a private key *and* match it to the published transformation key.

Accordingly, to prove security one needs to make a direct Dual-System encryption type proof. The proof would go along the lines of Lewko *et al.*, with the exception that in the hybrid stage of the proof *all* private keys and transformational keys will be set (one by one) to be semi-functional including those that could decrypt the eventual challenge ciphertext. In the Lewko *et al.* proof giving a private key that could decrypt the challenge ciphertext would undesirably result in the simulator producing observably incorrect correlations between the challenge ciphertext and keys. However, if we only give out the transformation part of such a key (and keep the whole private key hidden) then this correlation will remain hidden. This part of the argument is somewhat similar to the work of Lewko, Rouselakis, and Waters [29], who show that in their leakage resilient ABE scheme if only part of a private key is leaked such a correlation will be hidden.

5.2 Checking the Transformation

In the description of our systems a proxy will be able to transform any ABE ciphertext into a short ciphertext for the user. While the security definitions show that an attacker will not be able to learn an encrypted message, there is no guarantee on the transformation's correctness. In some applications a user might want to request the transformation of a particular ciphertext and (efficiently) check that the transformation was indeed done correctly (assuming the original ciphertext was valid). It is easy to adapt our RCCA systems to such a setting. Since decryption results in recovery of the ciphertext randomness, one can simply add a tag to the ciphertext as $H'(r)$, where H' is a different hash function modeled as a random oracle and r is the ciphertext randomness. On recovery of r the user can compute $H'(r)$ and make sure it matches the tag.

6 Performance in Practice

To validate our results, we implemented the CPA-secure CP-ABE of Section 3 as an extension to the libfenc Attribute Based Encryption library [25]. We then used this as a building block for a platform for accelerating ABE decryption through cloud-based computing resources.

The core of our solution is a virtualized outsourcing “proxy” that runs in the Amazon Elastic Compute Cloud (EC2). Our proxy exists as a machine image that can be programmatically instantiated by any application that requires assistance with ABE decryption. As we demonstrate below, this proxy is particularly useful for accelerating decryption on constrained devices such as mobile phones. However, the system can be used in any application where significant numbers of ABE decryptions must be performed, *e.g.*, in large-scale search op-

erations.¹⁰ The use of on-demand computing is particularly well-suited to our outsourcing techniques, since we do not require trusted remote servers or long-term storage of secrets.

System Architecture. Figure 7 illustrates the architecture of our outsourcing platform. The proxy is stored in Amazon’s S3 datastore as a public Amazon Machine Image (AMI), which wraps a standard Linux/Apache distribution along with the code needed to execute the Transform algorithm. Applications can remotely instantiate the proxy and upload a TK corresponding to a particular ABE decryption key.¹¹ Depending on the use case, they can either push ciphertexts to the proxy for transformation, or direct the proxy to retrieve ABE ciphertexts from remote locations such as the web or the Amazon S3 storage cloud. The latter technique is helpful when accessing remotely-held records on a mobile device, since the proxy transformation dramatically reduces the mobile device’s bandwidth requirements vs. downloading and decrypting each ABE ciphertext locally. This can significantly enhance device battery life.

6.1 Performance: Microbenchmarks

To evaluate the performance of our CPA-secure CP-ABE outsourcing scheme in isolation (without confounding factors such as network lag, file I/O, etc.) we conducted a series of microbenchmarks using the libfenc implementation. For consistency, we ran these tests on two dedicated hardware platforms: a 3GHz Intel Core Duo platform with 4GB of RAM running 32-bit Linux Kernel version 2.6.32, and a 412MHz ARM-based iPhone 3G with 128MB of RAM running iOS 4.0.¹² We instantiated the ABE schemes using a 224-bit MNT elliptic curve from the Stanford Pairing-Based Crypto library [30].¹³

The existing libfenc implementation implements the Waters scheme using a Key Encapsulation variant. For backwards compatibility, we adopted this approach in our implementation as well. Herein, the ciphertext carries a symmetric session key k that is computed at encryption time as $k = H(e(g, g)^{\alpha s})$. The element $C =$

$\mathcal{M} \cdot e(g, g)^{\alpha s}$ is omitted from the ciphertext, and any data payload must be carried via a separate symmetric encryption under k . The practical impact of this approach is that the ABE ciphertexts (and partially-decrypted ciphertexts) are shortened by one element of $\mathbb{G}_T$.

Experimental setup. Both decryption time and ciphertext size in the CP-ABE scheme depend on the complexity of the ciphertext’s policy. To capture this in our experiments, we first generated a collection of 100 distinct ciphertext policies of the form $(A_1 \text{ AND } A_2 \text{ AND } \dots \text{ AND } A_N)$, where each A_i is an attribute, for values of N increasing from 1 to 100. In each case we constructed a corresponding decryption key that contained the N attributes necessary for decryption. This approach ensures that the decryption procedure depends on all N components of the ciphertext and is a reasonable sample of a complex policy.

To obtain our baseline results, we encapsulated a random 128-bit symmetric key under each of these 100 different policies, then decrypted the resulting ABE ciphertext using the normal (non-outsourced) Decrypt algorithm.¹⁴ To smooth any experimental variability, we repeated each of our experiments 100 times on the Intel device (due to the time consuming nature of the experiments, we repeated the test only 30 times on the ARM device) and averaged to obtain our decryption timings. Figure 8 shows the size of the resulting ciphertexts as a function of N , along with the measured decryption times on our Intel and ARM test platforms.

Next, we evaluated the algorithms by generating a Transform Key (TK) from the appropriate N -attribute ABE decryption key and applying the Transform algorithm to the ABE ciphertext using this key.¹⁵ Finally we decrypted the resulting transformed ciphertext. Figure 8 shows the time required for each of those operations.

Discussion. As expected, the ABE ciphertext size and decryption/transform time were linear in the complexity of the ciphertext’s policy (N). However, our results illustrate the surprisingly high constants. Encrypting under a 100-component ciphertext policy produced an unwieldy 25KB of ABE ciphertext. The relatively fast Intel processor required nearly 2 full seconds to decrypt this value. By comparison, the same machine can perform a 1024-bit RSA decryption in 1.7 *milliseconds*.¹⁶

The results were more dramatic on the mobile device. Decrypting a 100-component ciphertext policy on the

¹⁰Indeed, since cloud computing platforms support the creation of multiple proxy instances, servers can rapidly scale their outsourcing capability up and down to meet demand.

¹¹The proxy requires only one TK to decrypt an unlimited number of ciphertexts. However, a proxy can be shared by multiple users, each with their own TK.

¹²Note that our tests were single-threaded, and thus used resources from only a single core of the Intel processor. In all cases we conducted our timing experiments with accessible background services disabled, and with the mobile device connected to a power source.

¹³Although we define our schemes in the symmetric bilinear group setting, the MNT curve choice required that we implement the scheme in asymmetric groups with a pairing of the form $\mathbb{G}_1 \times \mathbb{G}_2 \rightarrow \mathbb{G}_T$. As a result we assigned various elements of the ciphertext and key to the groups $\mathbb{G}_1$ and $\mathbb{G}_2$ with the aim of minimizing ciphertext size.

¹⁴Note that for this experiment we did not employ any symmetric encryption, hence all times and ciphertext sizes refer to the ABE key encapsulation ciphertext.

¹⁵We used the “backwards-compatible” key generation approach described in Section 3.1 to derive a TK from a standard ABE decryption key, rather than having the PKG generate the TK directly. This allowed us to retain compatibility with the existing CP-ABE implementation.

¹⁶Measured with OpenSSL 1.0 [40].

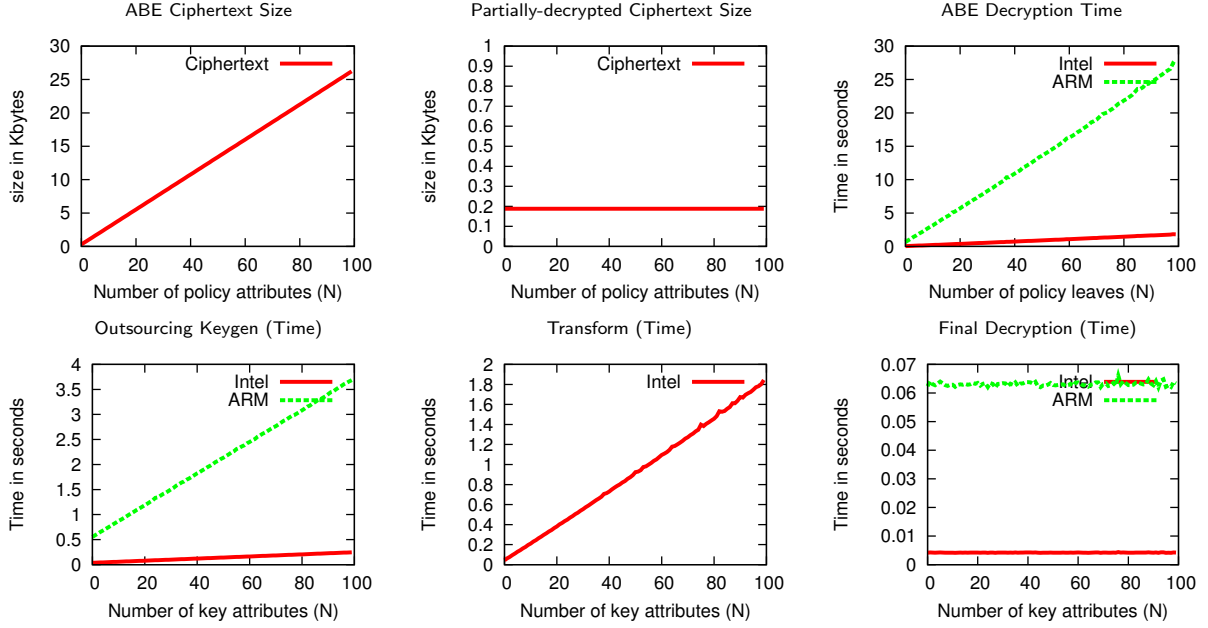


Figure 8: Microbenchmark results for our CP-ABE scheme with outsourcing. Timing results are provided for both Intel and ARM platforms. Key generation times represent the time to convert a standard ABE decryption key into an outsourcing key, using the “backwards-compatible” approach described in Section 3.1. “Final decryption” refers to the decryption of a partially-decrypted ciphertext. Note that we present the Transform timing results for the Intel platform only, since we view this as the more likely outsourcing platform. Intel (resp. ARM) timings represent the average of 100 (resp. 30) test iterations.

ARM processor required nearly *30 seconds* of sustained computation. Even at lower policy complexities, our results seem problematic for implementers looking to deploy unassisted ABE on limited computing devices.

Outsourcing substantially reduced both ciphertext size and the time needed to decrypt the partially-decrypted ciphertext. Each partially-decrypted ciphertext was a fixed 188 bytes in size, regardless of the original ciphertext’s CP-ABE policy. Furthermore, the final decryption process required only 4ms on the Intel processor and a manageable 60ms on ARM.¹⁷ Thus, it appears that outsourcing can provide a noticeable decryption time advantage for ciphertexts with 10 or more attributes.

Other Implementation Remarks. There are several optimizations and tradeoffs one might explore that could impact both the performance of the existing ABE scheme and our outsourced scheme. We chose to use the PBC library due to its use in the libfenc system and its simple API. However, PBC does not include all of the latest optimizations discussed in the research literature. Other future optimizations could include the use of multi-pairings for decryption. We emphasize that while using such op-

timizations to the existing ABE systems could give some performance improvements, they will not improve the size of ABE ciphertexts. Furthermore, decryption time will still be linear in the size of the satisfied formula, whereas our outsourcing technique transforms the final decryption step to a short El-Gamal-type ciphertext.

A note on policy complexity. The reader might assume that 50- or 100-component policies are rare in practice. In fact, we observed that it is relatively easy to arrive at highly complex policies in typical use cases. This is particularly true when using policies that contain integer comparison operators, *e.g.*, “AGE < 30”. The libfenc library implements integer comparison operators using the technique of Bethencourt et al. [7]: prior to encryption, each comparison operator is converted into a boolean policy circuit composed of OR and AND gates, and the resulting policy is applied to the ciphertext. Comparing an attribute to a fixed n -bit integer adds approximately n components to the policy. For example, without special optimizations, a restriction window involving a Unix time value ($x < \text{KEY\_CREATION\_TIME} < y$) increases the policy size by approximately 64 components.

¹⁷We conducted our experiments on the CPA-secure version of our scheme. The primary performance differences in the RCCA version are an extra exponentiation in $\mathbb{G}_T$ and some additional bytes.

Operation	local-only (sec)	local+web (sec/kb)	proxy (sec/kb)	proxy+web (sec/kb)
New proxy instantiation	.	.	93.4 sec	93.4 sec
Restart existing proxy instance	.	.	45 sec	45 sec
Generate & set 70-element transform key	.	.	2.9 sec	2.9 sec
Decryption:				
((DOCTOR OR NURSE) AND INSTITUTION)	1.1s	1.2s/1.1k	.2s/1.4k	.2s/0.4k
(DOCTOR AND TIME > 1262325600 AND TIME < 1267423200)	17.3s	17.3s/22.8k	1.2s/23.2k	1.2s/0.4k

Figure 9: Some average performance results for the proxy-enhanced iHealthEHR application running on our iPhone 3G. From left to right, “local-only” indicates device-local decryption and storage of ciphertexts, “local+web” indicates that ciphertexts were downloaded from a web server and decrypted at the device. “proxy” indicates local ciphertext storage with proxy outsourcing. “proxy+web” indicates that ciphertexts were obtained from the web *via* the proxy. Where relevant we provide both timings *and* total bandwidth transferred (up+down) from the device. Note that proxy launch times exhibit some variability depending on factors outside of our control.

6.2 Performance: Mobile Example

To validate our ideas in a real application, we incorporated outsourcing into the iPhone viewer component of iHealthEHR [3], an experimental system for distributing Electronic Health Records (EHRs). Since EHRs can contain highly sensitive data, iHealthEHR uses CP-ABE to perform end-to-end encryption of records from the origination point to the viewing device. Distinct ciphertext policies may be applied to each node in an individual’s health record (e.g., to admit special permissions for psychiatric records). iHealthEHR supports both local and cloud-based storage of records.

We modified the iPhone application to remotely instantiate our outsourcing proxy on startup, using a “small” server instance within Amazon’s storage cloud.¹⁸ In our experiments we found that the first EC2 instantiation required anywhere from 1-3 minutes, presumably depending on the system’s load. However, once the proxy was launched, it could be left running indefinitely and shared by many different users with different TKs, or — when not in use — paused and brought back to full operation in as little as 30 seconds (with an average closer to 45 seconds). During this startup interval we set the application to locally process all decryption operations. Once the proxy signaled its availability, the application pushed a TK to it via HTTP, and outsourced all further decryption operations.

To evaluate the performance implications, we conducted experiments on the system with outsourcing enabled and disabled, considering four likely usage scenarios. In the first scenario (local-only), we conducted device-local decryption on ciphertexts stored locally in the device’s Flash memory. In the second scenario (local+web) we downloaded ciphertexts from a web server,

then decrypted them locally at the device. In the third scenario (proxy), we stored ciphertexts locally and then *uploaded* them to the proxy for transformation. In the final scenario (proxy+web) ciphertexts were retrieved from a web server by the proxy, then Transformed before being sent to the device. In each case we measured the time required to decrypt, along with the total bandwidth transmitted and received by the device (excepting the local-only case, which did not employ the network connection). The results are summarized in Figure 9.

7 Hardening ABE Implementations

Thus far we described outsourcing solely as a means to improve decryption *performance*. In certain cases outsourcing can also be used to enhance security. By way of motivation, we observe that ABE implementations tend to be relatively complex compared to implementations of other public-key encryption schemes. For example, libfenc’s policy handling components alone comprise nearly 3,000 lines of C code, excluding library dependencies. It has been observed that the number of vulnerabilities in a software product tends to increase in proportion to the code’s complexity [34].

It is common for designers to mitigate software issues by sandboxing vulnerable processes *e.g.*, [33], or through techniques that isolate security-sensitive functions within a process [32]. McCune *et al.* recently proposed TrustVisor [31], a specialized hypervisor designed to protect and isolate security-sensitive “Pieces of Application Logic” (PALs) from less sensitive code.

We propose outsourcing as a tool to harden ABE implementations in platforms with code isolation. For example, in a system equipped with TrustVisor, implementers can embed the relatively simple key generation and Decrypt_{out} routines in security-sensitive code (*e.g.*, a TrustVisor PAL) and use outsourcing to push the remaining calculations into non-sensitive code. This not

¹⁸According to Amazon’s documentation, a small EC2 instance provides “the equivalent CPU capacity of a 1.0-1.2 GHz 2007 Opteron or 2007 Xeon processor” and 1.7GB of RAM, at a cost of USD \$0.085/hr. [1].

only reduces the size of the sensitive code base, it also simplifies parameter validation for the PAL (since the partially-decrypted ABE ciphertext is substantially less complex than the original). We refer to this technique as “self-outsourcing” and note that it can also be used in systems containing hardware security modules (*e.g.*, cryptographic smart cards). Moreover, based on our experiments of Section 6, we estimate that this approach will have a minimal impact on performance.

Acknowledgments

We thank the anonymous reviewers for their helpful comments.

References

- [1] Amazon EC2 FAQs. <http://aws.amazon.com/ec2/faqs/>, November 2010.
- [2] Michel Abdalla, Mihir Bellare, Dario Catalano, Eike Kiltz, Tadayoshi Kohno, Tanja Lange, John Malone-Lee, Gregory Neven, Pascal Paillier, and Haixia Shi. Searchable encryption revisited: Consistency properties, relation to anonymous ibe, and extensions. In *CRYPTO*, pages 205–222, 2005.
- [3] Joseph A. Akinyele, Christoph U. Lehmann, Matthew Green, Matthew W. Pagano, Zachary N. J. Peterson, and Aviel D. Rubin. Self-protecting electronic medical records using Attribute-Based Encryption. Cryptology ePrint Archive, Report 2010/565, 2010. Available from <http://eprint.iacr.org/>.
- [4] Giuseppe Ateniese, Kevin Fu, Matthew Green, and Susan Hohenberger. Improved proxy re-encryption schemes with applications to secure distributed storage. In *NDSS*, pages 29–43, 2005.
- [5] Amos Beimel. *Secure Schemes for Secret Sharing and Key Distribution*. PhD thesis, Israel Institute of Technology, Technion, Haifa, Israel, 1996.
- [6] John Bethencourt. Ciphertext-policy attribute-based encryption library. Available from <http://acsc.cs.utexas.edu/cpabe>, May 2010.
- [7] John Bethencourt, Amit Sahai, and Brent Waters. Ciphertext-policy attribute-based encryption. In *IEEE Symposium on Security and Privacy*, pages 321–334, 2007.
- [8] Matt Blaze, Gerrit Bleumer, and Martin Strauss. Divertible protocols and atomic proxy cryptography. In *EUROCRYPT*, pages 127–144, 1998.
- [9] Dan Boneh and Xavier Boyen. Efficient selective-id secure identity-based encryption without random oracles. In *EUROCRYPT*, pages 223–238, 2004.
- [10] Dan Boneh, Giovanni Di Crescenzo, Rafail Ostrovsky, and Giuseppe Persiano. Public key encryption with keyword search. In *EUROCRYPT*, pages 506–522, 2004.
- [11] Dan Boneh, Amit Sahai, and Brent Waters. Functional encryption: Definitions and challenges. In *TCC*, pages 253–273, 2011.
- [12] Dan Boneh and Brent Waters. Conjunctive, subset, and range queries on encrypted data. In *TCC*, pages 535–554, 2007.
- [13] Ran Canetti, Hugo Krawczyk, and Jesper Buus Nielsen. Relaxing chosen-ciphertext security. In *CRYPTO*, pages 565–582, 2003.
- [14] Melissa Chase. Multi-authority attribute based encryption. In *TCC*, pages 515–534, 2007.
- [15] Melissa Chase and Sherman S. M. Chow. Improving privacy and security in multi-authority attribute-based encryption. In *ACM Conference on Computer and Communications Security*, pages 121–130, 2009.
- [16] Benoît Chevallier-Mames, Jean-Sébastien Coron, Noel McCullagh, David Naccache, and Michael Scott. Secure delegation of elliptic-curve pairing. In *CARDIS*, pages 24–35, 2010.
- [17] Kai-Min Chung, Yael Kalai, and Salil P. Vadhan. Improved delegation of computation using fully homomorphic encryption. In *CRYPTO*, pages 483–501, 2010.
- [18] Eiichiro Fujisaki and Tatsuaki Okamoto. Secure integration of asymmetric and symmetric encryption schemes. In *CRYPTO '99*, volume 1666, pages 537–554, 1999.
- [19] Taher El Gamal. A public key cryptosystem and a signature scheme based on discrete logarithms. In *CRYPTO*, pages 10–18, 1984.
- [20] Rosario Gennaro, Craig Gentry, and Bryan Parno. Non-interactive verifiable computing: Outsourcing computation to untrusted workers. In *CRYPTO*, pages 465–482, 2010.
- [21] Craig Gentry. Fully homomorphic encryption using ideal lattices. In *STOC*, pages 169–178, 2009.

- [22] Craig Gentry and Shai Halevi. Implementing Gentry’s fully-homomorphic encryption scheme. In *EUROCRYPT*, pages 129–148, 2011.
- [23] Vipul Goyal, Abishek Jain, Omkant Pandey, and Amit Sahai. Bounded ciphertext policy attribute-based encryption. In *ICALP*, pages 579–591, 2008.
- [24] Vipul Goyal, Omkant Pandey, Amit Sahai, and Brent Waters. Attribute-based encryption for fine-grained access control of encrypted data. In *ACM Conference on Computer and Communications Security*, pages 89–98, 2006.
- [25] Matthew Green, Ayo Akinyele, and Michael Rushanan. libfenc: The Functional Encryption Library. Available from <http://code.google.com/p/libfenc>.
- [26] Matthew Green, Susan Hohenberger, and Brent Waters. Outsourcing the decryption of ABE ciphertexts, 2011. The full version of this paper is available from the Cryptology ePrint Archive.
- [27] Jonathan Katz, Amit Sahai, and Brent Waters. Predicate encryption supporting disjunctions, polynomial equations, and inner products. In *EUROCRYPT*, pages 146–162, 2008.
- [28] Allison Lewko, Tatsuaki Okamoto, Amit Sahai, Katsuyuki Takashima, and Brent Waters. Fully secure functional encryption: Attribute-based encryption and (hierarchical) inner product encryption. In *EUROCRYPT*, pages 62–91, 2010.
- [29] Allison Lewko, Yannis Rouselakis, and Brent Waters. Achieving leakage resilience through dual system encryption. In *TCC*, pages 70–88, 2011.
- [30] Ben Lynn. The Stanford Pairing Based Crypto Library. Available from <http://crypto.stanford.edu/abc>.
- [31] Jonathan M. McCune, Yanlin Li, Ning Qu, Zongwei Zhou, Anupam Datta, Virgil D. Gligor, and Adrian Perrig. TrustVisor: Efficient TCB Reduction and Attestation. In *IEEE Symposium on Security and Privacy*, pages 143–158, May 2010.
- [32] Jonathan M. McCune, Bryan Parno, Adrian Perrig, Michael K. Reiter, and Arvind Seshadri. Minimal tcb code execution (extended abstract). In *IEEE Symposium on Security and Privacy*, pages 267–272, 2007.
- [33] Elinor Mills. Chrome OS security: ‘Sandboxing’ and auto updates. eWeek., 2009.
- [34] Subhas C. Misra and Virendra C. Bhavsar. Relationships between selected software measures and latent bug-density: guidelines for improving quality. In *ICCSA’03*, pages 724–732, 2003.
- [35] Tatsuaki Okamoto and Katsuyuki Takashima. Fully secure functional encryption with general relations from the decisional linear assumption. In *CRYPTO*, pages 191–208, 2010.
- [36] Rafail Ostrovsky, Amit Sahai, and Brent Waters. Attribute-based encryption with non-monotonic access structures. In *ACM Conference on Computer and Communications Security*, pages 195–203, 2007.
- [37] Matthew Pirretti, Patrick Traynor, Patrick McDaniel, and Brent Waters. Secure attribute-based systems. In *ACM Conference on Computer and Communications Security*, pages 99–112, 2006.
- [38] Amit Sahai and Brent Waters. Fuzzy identity-based encryption. In *EUROCRYPT*, pages 457–473, 2005.
- [39] Elaine Shi, John Bethencourt, Hubert T.-H. Chan, Dawn Xiaodong Song, and Adrian Perrig. Multi-dimensional range query over encrypted data. In *IEEE Symposium on Security and Privacy*, pages 350–364, 2007.
- [40] The OpenSSL Project v1.0. OpenSSL: The open source toolkit for SSL/TLS. www.openssl.org, April 2010.
- [41] Brent Waters. Dual system encryption: Realizing fully secure IBE and HIBE under simple assumptions. In *CRYPTO*, pages 619–636, 2009.
- [42] Brent Waters. Ciphertext-policy attribute-based encryption: An expressive, efficient, and provably secure realization. In *PKC*, pages 53–70, 2011.

A Proof of Theorem 3.2

Proof. Suppose there exists a polynomial-time adversary $\mathcal{A}$ that can attack our scheme in the selective RCCA-security model for outsourcing with advantage ϵ . We build a simulator $\mathcal{B}$ that can attack the Waters scheme of [42, Appendix C] in the selective CPA-security model with advantage ϵ minus a negligible amount. In [42] the Waters scheme is proven secure under the decisional q -parallel BDHE assumption.

Init. The simulator $\mathcal{B}$ runs $\mathcal{A}$. $\mathcal{A}$ chooses the challenge access structure (M^*, ρ^*) , which $\mathcal{B}$ passes on to the Waters challenger as the structure on which it wishes to be challenged.

Setup. The simulator $\mathcal{B}$ obtains the Waters public parameters $\text{PK} = g, e(g, g)^\alpha, g^a$ and a description of the hash function F . It sends these to $\mathcal{A}$ as the public parameters.

Phase 1. The simulator $\mathcal{B}$ initializes empty tables T, T_1, T_2 , an empty set D and an integer $j = 0$. It answers the adversary's queries as follows:

- Random Oracle Hash $H_1(R, \mathcal{M})$: If there is an entry $(R, \mathcal{M}, s)$ in T_1 , return s . Otherwise, choose a random $s \in \mathbb{Z}_p$, record $(R, \mathcal{M}, s)$ in T_1 and return s .
- Random Oracle Hash $H_2(R)$: If there is an entry (R, r) in T_2 , return r . Otherwise, choose a random $r \in \{0, 1\}^k$, record (R, r) in T_2 and return r .
- Create((S)): $\mathcal{B}$ sets $j := j + 1$. It now proceeds one of two ways.
 - If S satisfies (M^*, ρ^*) , then it chooses a “fake” transformation key as follows: choose a random $d \in \mathbb{Z}_p$ and run $\text{KeyGen}((d, \text{PK}), S)$ to obtain SK' . Set $\text{TK} = \text{SK}'$ and set $\text{SK} = (d, \text{TK})$. Note that the pair (d, TK) is not well-formed, but that TK is properly distributed if d was replaced by the unknown value $z = \alpha/d$.
 - Otherwise, it calls the Waters key generation oracle on S to obtain the key $\text{SK}' = (\text{PK}, K', L', \{K'_x\}_{x \in S})$. (Recall that in the non-outsourcing CP-ABE game, the Create and Corrupt functionalities are combined in one oracle.) The algorithm chooses a random value $z \in \mathbb{Z}_p$ and sets the transformation key TK as $(\text{PK}, K = K'^{1/z}, L = L'^{1/z}, \{K_x\}_{x \in S} = \{K'_x\}_{x \in S})$ and the private key as (z, TK) .

Finally, store $(j, S, \text{SK}, \text{TK})$ in table T and return TK to $\mathcal{A}$.

- Corrupt(i): $\mathcal{A}$ cannot ask to corrupt any key corresponding to the challenge structure (M^*, ρ^*) . If there exists an i th entry in table T , then $\mathcal{B}$ obtains the entry $(i, S, \text{SK}, \text{TK})$ and sets $D := D \cup \{S\}$. It then returns SK to $\mathcal{A}$, or $\perp$ if no such entry exists.
- Decrypt(i, CT): Without loss of generality, we assume that all ciphertexts input to this oracle are already partially decrypted. Recall that both $\mathcal{B}$ and $\mathcal{A}$ have access to the TK values for all keys created, so either can execute the transformation operation. Let $\text{CT} = (C_0, C_1, C_2)$ be associated with structure (M, ρ) . Obtain the record $(i, S, \text{SK}, \text{TK})$ from table T . If it is not there or $S \notin (M, \rho)$, return $\perp$ to $\mathcal{A}$. If key i does not satisfy the challenge structure (M^*, ρ^*) , proceed as follows:

1. Parse $\text{SK} = (z, \text{TK})$. Compute $R = C_0/C_2^z$.
2. Obtain the records $(R, \mathcal{M}_i, s_i)$ from table T_1 . If none exist, return $\perp$ to $\mathcal{A}$.

3. If in this set, there exists indices $y \neq x$ such that $(R, \mathcal{M}_y, s_y)$ and $(R, \mathcal{M}_x, s_x)$ are in table T_1 , $\mathcal{M}_y \neq \mathcal{M}_x$ and $s_y = s_x$, then $\mathcal{B}$ aborts the simulation.
4. Otherwise, obtain the record (R, r) from table T_2 . If it does not exist, $\mathcal{B}$ outputs $\perp$.
5. For each i , test if $C_0 = R \cdot e(g, g)^{\alpha s_i}$, $C_1 = \mathcal{M}_i \oplus r$ and $C_2 = e(g, g)^{\alpha s_i/z}$.
6. If there is an i that passes the above test, output the message $\mathcal{M}_i$; otherwise, output $\perp$. (Note: at most one value of s_i , and thereby one index i , can satisfy the third check of the above test.)

If key i does satisfy the challenge structure (M^*, ρ^*) , proceed as follows:

1. Parse $\text{SK} = (d, \text{TK})$. Compute $\beta = C_2^{1/d}$.
2. For each record $(R_i, \mathcal{M}_i, s_i)$ in table T_1 , test if $\beta = e(g, g)^{s_i}$.
3. If zero matches are found, $\mathcal{B}$ outputs $\perp$ to $\mathcal{A}$.
4. If more than one matches are found, $\mathcal{B}$ aborts the simulation.
5. Otherwise, let $(R, \mathcal{M}, s)$ be the sole match. Obtain the record (R, r) from table T_2 . If it does not exist, $\mathcal{B}$ outputs $\perp$.
6. Test if $C_0 = R \cdot e(g, g)^{\alpha s}$, $C_1 = \mathcal{M} \oplus r$ and $C_2 = e(g, g)^{ds}$.
7. If all tests pass, output $\mathcal{M}$; else, output $\perp$.

Challenge. Eventually, $\mathcal{A}$ submits a message pair $(\mathcal{M}_0^*, \mathcal{M}_1^*) \in \{0, 1\}^{2 \times k}$. $\mathcal{B}$ acts as follows:

1. $\mathcal{B}$ chooses random “messages” $(\mathcal{R}_0, \mathcal{R}_1) \in \mathbb{G}_T^2$ and passes them on to the Waters challenger to obtain a ciphertext $\text{CT} = (C, C', \{C_i\}_{i \in [1, \ell]})$ under (M^*, ρ^*) .
2. $\mathcal{B}$ chooses a random value $C'' \in \{0, 1\}^k$.
3. $\mathcal{B}$ sends to $\mathcal{A}$ the challenge ciphertext $\text{CT}^* = (C, C', C'', \{C_i\}_{i \in [1, \ell]})$.

Phase 2. The simulator $\mathcal{B}$ continues to answer queries as in Phase 1, except that if the response to a Decrypt query would be either $\mathcal{M}_0^*$ or $\mathcal{M}_1^*$, then $\mathcal{B}$ responds with the message **test** instead.

Guess. Eventually, $\mathcal{A}$ must either output a bit or abort, either way $\mathcal{B}$ ignores it. Next, $\mathcal{B}$ searches through tables T_1 and T_2 to see if the values $\mathcal{R}_0$ or $\mathcal{R}_1$ appear as the first element of any entry (i.e., that $\mathcal{A}$ issued a query of the form $H_1(\mathcal{R}_i, \cdot)$ or $H_2(\mathcal{R}_i)$). If neither or both values appear, $\mathcal{B}$ outputs a random bit as its guess. If only value $\mathcal{R}_b$ appears, then $\mathcal{B}$ outputs b as its guess.

This ends the description of the simulation. Due to space limitations, our analysis of this simulation appears in the full version of this work [26]. $\square$

Glossary of Terms

ABE	Attribute Based Encryption
AES	Advanced Encryption Standard
AES-NI	AES Encryption Instruction Set from Intel
BLS	Boneh-Lynn-Shachem Signature
CCA	Chosen Ciphertext Attack
CHARM	A Toolkit for Prototyping Cryptosystems
CLP ORAM	Circuit Oblivious Random Access Memory
cNM-CCA1	Chosen Non-Malleable Chosen Ciphertext Attack
CPA	Chosen Plaintext Attack
CPU	Central Processing Unit
CS-Lite	Cramer-Shoup Lite Scheme
DAP Schemes	Decentralized Anonymous Payment Scheme
DARPA	Defense Advanced Research Projects Agency
DCCA	Detectable Chosen Ciphertext Security
DDH	Decisional Diffe-Hellman Assumption
DHK	Diffe-Hellman Knowledge
DRG	Damgard Elgamal Scheme
EC2	Amazon Elastic Compute Cloud Infrastructure
ECC	Error Correcting Code
ECDSA	Elliptic Curve Signature Algorithm
EDT	Encryption Data Table
FHE	Fully Homomorphic Encryption
GPGPU	General Purpose Graphics Processing Unit
GPU	Graphical Processing Unit
HBC	Honest But Curioous
IBE	Identity Based Encryption Scheme; Boneh - Franklin
LWE	Learning With Errors
MIMD	Multiple Instruction, Multiple Data; a class of Parallel Computers
MIRACL	A Pairing Based Encryption Library
NAND	Not AND Logical Gate
NIZK	Non-Interactive Zero Knowledge Proof
NM-CCA1	Non-Maleable Chosen Ciphertext Attack
ORAM	Oblivious Randon Access Memory
OT	Oblivious Transfer
PCF	Portable Circuit Format
PCP Theorem	Probabalistically Checkable Proof Characterization Theorem
PI	Principle Investigator
POK	Proof of Knowledge
p-Tampering	Attack where each bit may be tampered with probabality p
RAM	Random Access Memory
RELIC	A Pairing Based Encryption Library
RLWE	Ring Learning With Errors
RSA	Rivest, Shamir, Adleman Encryption Scheme
SC	Secure Computation
SCORAM	Heuristic Compact Oblivious Random Access Memory
SFE	Secure Function Evaluation
SIMD	Single Instruction, Multiple Data; a class of Parallel Computers

Approved for Public Release; Distribution Unlimited.

SMC	Secure Multi-Party Encryption
SMT Solvers	Satisfiability Modulo Theories Solvers
SOA	Selective Opening Attack
SSE2	Extension of SIMD Instruction Set
TFHE	Threshold Fully Homomorphic Encryption
UC	Universal Composition
VSS	Verifiable Secret Sharing Scheme
XEN Hypervisor	Hypervisor Utilized in Amazon EC2 Cloud
XOR	Exclusive OR Logical Gate
Yau	Yau's Garbled Circuit Protocol
ZK Proof	Zero Knowledge Proof
zk-SNARKs	Zero Knowledge Succinct Non-Interactive Arguments of Knowledge