NOVEL TECHNIQUES FOR SECURE USE OF PUBLIC CLOUD COMPUTING RESOURCES

DISSERTATION

Kyle E. Stewart, Captain, USAF

AFIT-ENG-DS-15-S-018

DEPARTMENT OF THE AIR FORCE
AIR UNIVERSITY

AIR FORCE INSTITUTE OF TECHNOLOGY

Wright-Patterson Air Force Base, Ohio

DISTRIBUTION STATEMENT A:
APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED

The views expressed in this dissertation are those of the author and do not reflect the official policy or position of the United States Air Force, the Department of Defense, or the United States Government.

This material is declared a work of the U.S. Government and is not subject to copyright protection in the United States.

AFIT-ENG-DS-15-S-018

NOVEL TECHNIQUES FOR SECURE USE OF PUBLIC CLOUD COMPUTING RESOURCES

DISSERTATION

Presented to the Faculty

Graduate School of Engineering and Management

Air Force Institute of Technology

Air University

Air Education and Training Command

in Partial Fulfillment of the Requirements for the

Degree of Doctor of Philosophy

Kyle E. Stewart, B.S. , M.S.

Captain, USAF

September 2015

DISTRIBUTION STATEMENT A:
APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED

AFIT-ENG-DS-15-S-018

NOVEL TECHNIQUES FOR SECURE USE OF PUBLIC CLOUD COMPUTING RESOURCES

DISSERTATION

Kyle E. Stewart, B.S., M.S.
Captain, USAF

Committee Membership:

Kenneth. M. Hopkinson, PhD
Chairman

Benjamin F. Akers, PhD
Member

Todd R. Andel, PhD
Member

Douglas D. Hodson, PhD
Member

ADEDJI B. BADIRU, PhD
Dean, Graduate School of Engineering and Management

AFIT-ENG-DS-15-S-018

Abstract

The federal government has an expressed interest in moving data and services to third party service providers in order to take advantage of the flexibility, scalability, and potential cost savings. This approach is called cloud computing. In order to leverage the cost advantages of cloud computing, the federal government must utilize public cloud computing resources and techniques to mitigate security concerns. The purpose of this research is to examine methodologies that may be employed in order allow users secure access to public cloud computing resources. The thesis for this research is that efficient techniques exist to support the secure use of public cloud computing resources by a large, federated enterprise. The primary contributions of this research are the novel cryptographic system MA-AHASBE (Multi-Authority Anonymous Hierarchical Attribute-Set Based Encryption), and the techniques used to incorporate MA-AHASBE in a real world application. MA-AHASBE was developed specifically because the attribute-based encryption systems available did not offer everything that would be required in a large, federated organization. MA-AHASBE allows users to create ciphertext that enforces an access policy, allows policies to include attributes from multiple authorities in either a centralized or decentralized environment, allows for protected attributes that are not disclosed in the ciphertext, and allows an authority to delegate key generation capabilities to a hierarchy of subdomains. The system is described using asymmetric bilinear maps over pairing friendly elliptic curves, which have been shown to be the most efficient pairings in practice. The second significant contribution of this research is the application of MA-AHASBE to a microblogging application. Two new security models were developed called A-trusted and ACE-trusted. These models focus on availability as the key trust requirement between a client and a cloud service provider. The A-trusted model requires only availability trust, while the ACE-trusted model adds certain computability and access

control enforcement requirements. While this does increase the trust required in a cloud service provider, the efficiency gains can be significant and the tradeoff in security seems reasonable for reputable cloud service providers. Neither model requires confidentiality trust, so cloud service providers cannot learn the contents of data. In order to support the security models, a framework called Axon is presented. This framework demonstrates a layered, data structure oriented approach to building complex, secure data structures and protocols. This method is used to build Critter, a secure microblogging application. Performance results indicate that while there is a cost associated with enforcing an ACE-trusted model in Critter versus processing the data in plaintext, the cost is not unreasonable and the benefits in security can be significant. The contributions of this research give the DoD additional tools for supporting the mission while taking advantage of the cost efficient public cloud computing resources that are becoming widely available.

Acknowledgments

I would like to thank my research advisor Dr. Hopkinson for his support and guidance over the last three years. I would also like to thank my research committee for their support and feedback on the research. I also want to thank Mike Clark and Allison Lewko for the many fruitful crypto discussions and for their careful review of the MA-AHASBE paper. Finally, I would like to thank my family for their support.

Kyle E. Stewart

Table of Contents

Page

Abstract iv

Acknowledgments vi

Table of Contents vii

List of Figures xi

List of Tables xiii

List of Abbreviations xiv

I. Introduction 1

II. Cloud Computing 4

2.1 Cloud Computing 4
2.1.1 Cloud Service Models 5
2.2 Federal Use of Cloud Computing 7
2.2.1 Federal Data Center Consolidation Initiative 8
2.2.2 Federal Risk and Authorization Management Program (FedRAMP) 10
2.2.3 Example from Department of the Treasury 11
2.2.4 Example from Homeland Security 12

III. Pairing Based Cryptography 14

3.1 Mathematics Primer 14
3.1.1 Abstract Algebra 15
3.1.1.1 Notation 15
3.1.1.2 Groups 16
3.1.2 Elliptic Curves 19
3.2 Bilinear Maps (Pairings) 21
3.2.1 Pairing Types 22
3.2.2 Pairing Algebra 23
3.3 Cryptography 24
3.3.1 Semantic Security 24
3.3.2 Complexity Assumptions 25

Page

3.3.3 Dual System Encryption 28

IV. MA-AHASBE: Multi-Authority Anonymous Hierarchical Attribute-Set Based Encryption 31

4.1 Introduction 31
4.2 Background 33
4.2.1 Enterprise Attribute Based Access Control Requirements 33
4.2.1.1 Definitions 33
4.2.1.2 Requirements 33
4.2.1.3 Features 35
4.2.2 Related Work 36
4.3 MA-AHASBE 39
4.3.1 Preliminaries and Notation 39
4.3.1.1 Bilinear Pairings 39
4.3.1.2 Notation 39
4.3.2 MA-AHASBE Overview 40
4.3.2.1 Decentralization 41
4.3.2.2 MA-AHASBE Fundamentals 42
4.3.2.3 Key Structures, Attribute Sets and Multiple Domain Policies 49
4.3.3 Algorithm Summary 54
4.4 MA-AHASBE Construction 55
4.5 Security 66
4.5.1 Security Model 66
4.5.2 Security Intuition 68
4.5.2.1 Single Authority 69
4.5.2.2 Multiple Authorities 70
4.5.2.3 Key Escrow 71
4.5.3 Complexity Assumptions 72
4.5.4 Security Theorem 74
4.5.5 Proof Overview 74
4.5.6 Chosen Ciphertext Security Security 77
4.6 Conclusions and Future Work 78

V. MA-AHASBE Based Access Control in a Cloud Supported Publish-Subscribe Data Model 79

5.1 Introduction 79
5.2 Background and Related Work 82
5.2.1 Definitions 82
5.2.2 Related Work 84

Page

5.2.3 MA-AHASBE . . . 85
5.2.4 Tools . . . 88
5.3 Security Model . . . 90
5.3.1 The A-trusted and ACE-trusted Security Models . . . 91
5.4 Axon: A Data Structure Approach to Security, Scalability and Dependability . . . 97
5.4.1 Architecture . . . 98
5.5 Applications . . . 102
5.5.1 Secure Map . . . 103
5.5.2 Secure Remote Procedure Call . . . 104
5.5.3 Micro-Messaging . . . 104
5.5.4 Experimental Results . . . 110
5.6 Conclusions and Future Work . . . 115

VI. Summary of Contributions and Further Research . . . 119

6.1 Contributions . . . 119
6.2 Future Work . . . 120
6.3 Conclusion . . . 121

Appendix A: MA-AHASBE Security Proof Details . . . 123

A.1 Notation and Preliminaries . . . 123
A.2 Lemma 4.1. . . . 126
A.2.1 Assumption . . . 126
A.2.2 Setup(λ, n_{auth}) . . . 126
A.2.3 Phases 1 and 2 . . . 127
A.2.4 Challenge . . . 129
A.2.5 Guess . . . 130
A.3 Lemma 4.2. . . . 130
A.3.1 Assumption . . . 130
A.3.2 Setup(λ, n_{auth}) . . . 131
A.3.3 Phases 1 and 2 . . . 132
A.3.4 Challenge . . . 133
A.3.5 Guess . . . 134
A.4 Lemma 4.3. . . . 134
A.5 Lemma 4.4. . . . 134
A.5.1 Assumption . . . 135
A.5.2 Setup(λ, n_{auth}) . . . 135
A.5.3 Phases 1 and 2 . . . 136
A.5.4 Challenge . . . 137
A.5.5 Guess . . . 138

Page

A.6 Lemma 4.5. 138
A.6.1 Assumption 139
A.6.2 Setup(λ, n_{auth}) 139
A.6.3 Phases 1 and 2 140
A.6.4 Challenge 141
A.6.5 Guess 142

Appendix B: MA-AHASBE Implementation Details 144

Appendix C: Notes on Encrypted Search and Real-Time Collaboration 153

List of Figures

Figure Page

2.1 Usage of the phrase *cloud computing* 5

3.1 Elliptic curves over the real numbers with different values for constants a and b [3] 20

3.2 Elliptic curve defined over the finite field $\mathbb{Z}_{61}$ [3] 21

4.1 Full Attribute Chart 43

4.2 Example policy with attributes from a single domain 44

4.3 Example policy with a protected attribute 46

4.4 Example key structure 50

4.5 Example policies with translation nodes 52

4.6 Example policy with multiple authorities and domains 53

4.7 Example key ring 53

5.1 MA-AHASBE encryption flow with AEAD only. Shaded portions indicate private data and dashed arrows indicate the result of an encryption operation. . 88

5.2 MA-AHASBE encryption flow with KP-PKE and AEAD. Shaded portions indicate private data and dashed arrows indicate the result of an encryption operation. 89

5.3 Axon Information Flow 99

5.4 Example Session for Data Structure Creation and Authorization Protocols . . . 101

5.5 Secure RPC Message Sequence 105

5.6 Critter Information Flow 106

5.7 General Critter Message Policy Structure 107

5.8 Example message mapped to security policy 108

Figure Page

5.9 The number of mentions in a tweet per the number of tags based on percentiles. For example, of all the sample tweets with exactly two topic tags, 95% have three mention tags or less and 99% have six mention tags or less. All sample tweets with two topic tags had ten mentions or less. 112

5.10 The number of topic tags in a tweet per the number of mentions based on percentiles. For example, of all the sample tweets with exactly two mention tags, 95% have three mention tags or less and 99% have five mentions or less. All sample tweets with two mentions had 14 topic tags or less. 113

5.11 The amount of time to encrypt a tweet for a given number of topics and the percentiles of mentions. 114

5.12 The amount of time to decrypt a tweet for a given number of topics and the percentiles of mentions. 115

5.13 The size of ciphertext for a given number of topics and the percentiles of mentions. 116

5.14 Throughput Performance per Number of Servers and Payload Size. The dashed lines represent the sample mean and the solid line indicates the median. The *S* label indicates the number of servers and the *P* label indicates the payload size. P1=140 bytes, P2=3,500 bytes, P3=5,000 bytes. 117

B.1 Scenario 1 149

B.2 Scenario 2 150

List of Tables

Table Page

2.1 Data center consolidation plans by branch 10

3.1 Mathematical Notation 16

3.2 Dual System Components (w/HP indicates with high probability) 29

4.1 Functions 40

5.1 Axon Data Structures 98

5.2 Summary Statistics 111

B.1 Policy Sizes 150

B.2 Key Sizes 151

B.3 Timing 151

List of Abbreviations

Abbreviation		Page
MA-AHASBE	Multi-Authority Anonymous Hierarchical Attribute-Set Based Encryption	2
NIST	National Institute of Technology and Standards	4
CSP	Cloud Service Provider	5
SaaS	Software-as-a-Service	5
PaaS	Platform-as-a-Service	6
IaaS	Infrastructure-as-a-Service	6
DoD	Department of Defense	7
FCCI	Federal Cloud Computing Initiative	7
FDCCI	Federal Data Center Consolidation Initiative	8
OMB	Office of Management and Budget	9
CIO	Chief Information Officer	9
GAO	Government Accountability Office	9
C&A	Certification and Accreditation	11
FedRAMP	Federal Risk and Authorization Management Program	11
CSP	Cloud Service Providers	11
AWS	Amazon Web Services	11
DHS	Department of Homeland Security	12
PPTA	Probabilistic, Polynomial-Time Algorithm	24
AHIBE	Anonymous Hierarchical Identity Based Encryption	31
CP-ASBE	Ciphertext-Policy Attribute-Set Based Encryption	31
ABAC	Attribute Based Access Control	31

Abbreviation		Page
IBE	Identity Based Encryption	31
CP-ABE	Ciphertext-Policy Attribute Based Encryption	32
HIBE	Hierarchical Identity Based Encryption	36
SMPC	Secure Multi-Party Computation	41
AWS SDK	Amazon Web Services Software Development Kit	144

NOVEL TECHNIQUES FOR SECURE USE OF
PUBLIC CLOUD COMPUTING RESOURCES

I. Introduction

Cloud computing can be a divisive term. For some it means lower operating costs, flexible deployment options, and increased security. For many others it can mean something very different. Some view cloud computing as inherently less secure, less flexible, and that the lure of lower operating costs is not as attractive as it might seem. Others see it as a meaningless, marketing buzzword that only serves to create confusion [48]. Either way, the federal government has an expressed interest in moving data and services to the cloud in order to take advantage of potential cost savings [67]. Cloud computing is particularly important in the context of the federal strategy to consolidate the thousands of data centers owned and operated by the government [66]. In order to truly leverage the cost advantages of cloud computing, the federal government must utilize public cloud computing resources. However, naïvely moving sensitive data and services to third party servers may involve unnecessary tradeoffs in security. The purpose of this research is to examine methodologies that may be employed in order allow users secure access to public cloud computing resources. The thesis for this research can be stated as:

> *Efficient techniques exist to support the secure use of public cloud computing resources by a large, federated enterprise.*

This dissertation describes the work done in supporting this claim. Aside from the background material presented, the research presented in this document can be divided into two main areas. The first area is the development of a new cryptographic system

called Multi-Authority Anonymous Hierarchical Attribute-Set Based Encryption (MA-AHASBE). This new system presents a solution to the problem of key management that is often not addressed by research on secure cloud computing. The second area presents an application of MA-AHASBE for performing secure services using untrusted servers. This research presents a layered approach to building secure services, in the context of an example microblogging application called Critter. This research also presents a novel security model based on the information security principle of availability. These two areas of the research, along with background material, are divided into the following chapters:

Chapter 2 Provides background material on cloud computing and its use in the federal government. It reviews important cloud computing terminology and service models that will be used throughout this document.

Chapter 3 Presents the mathematical background and context necessary to understand the cryptographic system presented in Chapters 4 and 5. Specifically, this covers the basics of group theory, elliptic curve cryptography, and bilinear pairings.

Chapter 4 Introduces a novel cryptographic system called MA-AHASBE. MA-AHASBE is a public key cryptographic system designed to support the security requirements of a large, federated enterprise attribute based access control (ABAC) system. This chapter describes the system components in detail, presents a mathematical construction, and provides an argument for its security in the context of a formal security model.

Chapter 5 Examines the performance characteristics of the MA-AHASBE system introduced in Chapter 4 through a research implementation. This includes aspects such as the size of ciphertexts as well as the time required to perform encryption and decryption operations. This is done in the context of a microblogging application that uses a simple publish-subscribe mechanism. This chapter also introduces a security

model based on the information security principle of availability that offers a different perspective than the popular *honest-but-curious* security model. The research also presents a framework called Axon that facilitates building complex applications under the new security model through a layered, data structure oriented approach.

Chapter 6 Provides a summary of the research contributions in this document as well as some thoughts on where future work may take place.

The primary contributions of this research are the novel cryptographic system MA-AHASBE, and the techniques used to incorporate MA-AHASBE in a real world application. MA-AHASBE was developed specifically because the attribute-based encryption systems available did not offer everything that would be required in large, federated organization such as the DoD. A primary goal of this research is to demonstrate that attribute-based encryption is a viable option for future applications and that it provides significant flexibility that is not possible with the current DoD public key infrastructure. Specifically, it allows applications to be built that can take advantage of the low cost of public cloud computing without making significant concessions in terms of security. The storage and processing needs of the DoD are increasing, but there is also constant budgetary pressure to do more with less. The contributions of this research give the DoD additional tools for supporting the mission while taking advantage of the cost efficient public cloud computing resources that are becoming widely available.

II. Cloud Computing

This chapter discusses the basics of cloud computing and its use in the U.S. Government. It is divided into two sections. The first section introduces cloud computing terminology and concepts that are used throughout this document. The second section presents some key examples of why the government is interested in cloud computing and provides a few examples of how the government has, albeit conservatively, leveraged the public cloud.

2.1 Cloud Computing

The term *cloud computing* is a phrase that has become increasingly popular over the last ten years. Figure 2.1 shows the usage of the phrase according to Google's Ngram Viewer [51], which indexes books from 1800 to 2008. Over time, cloud computing has become the industry buzzword for the long held dream of large scale computing as a utility. Especially during its initial growth, cloud computing was met with both enthusiasm and indifference. On the one hand, it did not represent anything fundamentally new about computing as much of the technology (e.g., virtualization, network security, distributed systems, computing as a utility) had been researched extensively since the 1960s. On the other hand, the combination of an expanding Internet infrastructure combined with the excess capacity in large scale data centers allowed computing to be packaged and sold in a way that was not previously feasible. One of the initial hurdles in the federal adoption of cloud computing was the lack of a clear, widely accepted definition. For some, cloud computing was simply any type of computing that involved the Internet. For others it was the growth of Internet based applications such as Gmail, YouTube or SalesForce.com. In 2009, the National Institute of Technology and Standards (NIST) began work on standardizing the definition of cloud computing. The definition itself [83] is a

short, two-page description of the key elements of cloud computing such as its essential characteristics, service models and deployment models. This definition helped establish the conceptual properties of cloud computing and gave additional support to why these properties were distinct from other types of computing.

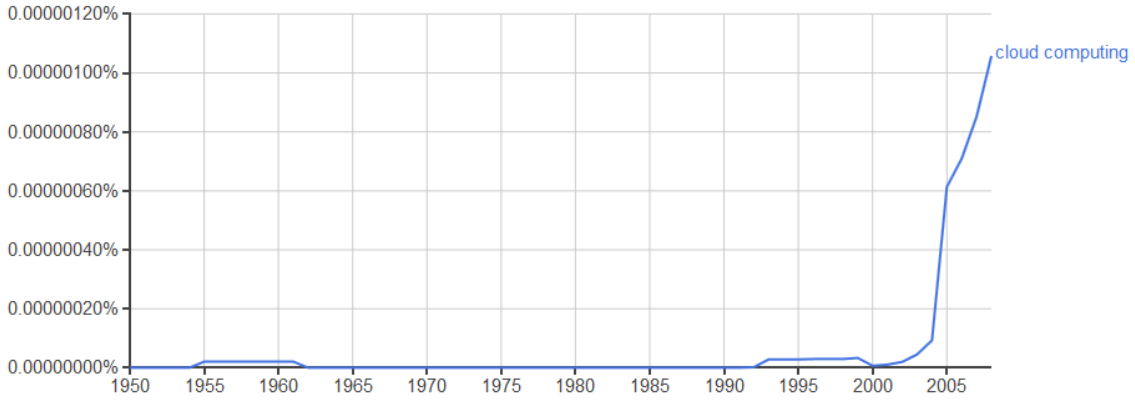


Figure 2.1: Usage of the phrase *cloud computing*

2.1.1 Cloud Service Models.

In order to facilitate discussion of cloud computing, NIST was tasked to provide a standardized definition of cloud computing. As part of this definition [83], NIST defines three types of service models that are characteristic of cloud service providers (CSP). A CSP may provide service through one or more of the following *service models*:

Software-as-a-Service (SaaS) The capability provided to the consumer is to use the provider's applications running on a cloud infrastructure. The applications are accessible from various client devices through a thin client interface such as a web browser (e.g., web-based email). The consumer does not manage or control the underlying cloud infrastructure including network, servers, operating systems, storage, or even individual application capabilities, with the possible exception of limited user-specific application configuration settings.

Platform-as-a-Service (PaaS) The capability provided to the consumer is to deploy onto the cloud infrastructure consumer-created or acquired applications created using programming languages and tools supported by the provider. The consumer does not manage or control the underlying cloud infrastructure including network, servers, operating systems, or storage, but has control over the deployed applications and possibly application hosting environment configurations.

Infrastructure-as-a-Service (IaaS) The capability provided to the consumer is to provision processing, storage, networks, and other fundamental computing resources where the consumer is able to deploy and run arbitrary software, which can include operating systems and applications. The consumer does not manage or control the underlying cloud infrastructure but has control over operating systems, storage, deployed applications, and possibly limited control of select networking components (e.g., host firewalls).

The NIST definition of cloud computing also provides the following four *deployment models*:

Private Cloud The cloud infrastructure is operated solely for an organization. It may be managed by the organization or a third party and may exist on premise or off premise.

Community Cloud The cloud infrastructure is shared by several organizations and supports a specific community that has shared concerns (e.g., mission, security requirements, policy, and compliance considerations). It may be managed by the organizations or a third party and may exist on premise or off premise.

Public Cloud The cloud infrastructure is made available to the general public or a large industry group and is owned by an organization selling cloud services.

Hybrid Cloud The cloud infrastructure is a composition of two or more clouds (private, community, or public) that remain unique entities but are bound together by

standardized or proprietary technology that enables data and application portability (e.g., cloud bursting for load balancing between clouds).

Typically, the tradeoffs of choosing between deployment models are often between cost, convenience and security. Public clouds often offer the greatest cost benefit, but traditionally cause the most uneasiness with regards to security. The federal and Department of Defense (DoD) strategy for cloud computing as discussed in §2.2 involve all four deployment models [67, 100]. The purpose of this research is to look at methods for securely adopting cloud computing using as much of the public cloud deployment model as possible, while at the same time limiting the information available to the CSP. By limiting the information available to the CSP, this will hopefully soften the cultural barriers that exist for adopting cloud computing more broadly within the DoD. Attribute based encryption techniques, such as the one presented in Chapter 4 require some type of trusted server to perform some of the cryptographic operations. These trusted servers could run inside a private computing cloud such as DISA's RACE [2]. This trusted component should represent a very small, relatively static portion of the overall processing and storage requirements for the overall application. The bulk of processing and storage could still be securely outsourced to public cloud computing resources. This configuration would fall under the hybrid cloud model in the NIST definition.

2.2 Federal Use of Cloud Computing

Since 2009, the federal government has gradually taken a more aggressive approach to cloud computing adoption. Since that time, this federal push known as the Federal Cloud Computing Initiative (FCCI) has met with both enthusiasm and resistance. The topic of cloud computing in the federal government is a broad topic, and in fact entire books have been written on this specific topic [14, 78]. It seems the most common government use of public cloud services are for public websites. Private cloud services are also being built up using the internal resources of the various federal agencies. Using virtualization alone does

not necessarily turn an internal data center into a private cloud. To fit the NIST definition, these resources must be made available as an on-demand, metered service. Many federal agencies are adopting this approach as well, using central data centers to provide cloud services to smaller compartments within the agency. This section discusses some of the key elements of federal adoption of cloud computing including small case studies on the approach of the Department of the Treasury and the Department of Homeland Security.

2.2.1 Federal Data Center Consolidation Initiative.

One of the modern drivers for cloud computing, both in industry and government, is the allure of reducing or eliminating the operating costs associated with running large internal data centers. In 1993, then President Bill Clinton asked Vice President Al Gore to perform an internal review of the federal government in order to determine areas where the government could be more efficient. This review was called the National Performance Review. This review contains a report titled Reengineering Through Information Technology [91], that identifies "consolidating and modernizing government data processing centers" as one of its four proposed actions. The report cites a DoD plan to consolidate 100 data centers into 16 efficient centers.

Over the next several years, the need for information processing would continue to drive up the number of federal data centers in operation. In 2009, another call to action came when Federal Chief Information Officer Vivek Kundra sent a memo to all federal agency CIOs discussing the need for data center consolidation [66]. The memo emphasized that the number of federal data centers had grown considerably over the previous decade (from 432 in 1998 to over 1,100 in 2009). The memo asked the CIOs to evaluate their agencies to find ways to either reduce data center requirements either through server virtualization or through cloud computing. This feedback led to the Federal CIO publishing a 25 point implementation plan for the Federal Data Center Consolidation Initiative (FDCCI) [65]. Point 3 in this plan called for a "shift to a cloud first policy".

This required government agencies to default to a cloud-based solution when appropriate for all new IT deployments. These cloud solutions could leverage private government clouds, commercial public clouds, or regional (i.e., community) clouds for state and local governments. In order to support this cloud first policy, the Federal CIO was required to publish a federal cloud strategy. This strategy document was published three months later in February of 2011 [67].

The progress of the FDCCI to reduce data centers and realize cost savings of $2.4 billion by 2015 has been difficult to measure. One reason for this difficulty is that the definition for what constitutes a data center has changed throughout the program. Initially the Office of Management and Budget (OMB) laid out specific requirements in terms of size, purpose and availability. For example, the data center must provide at least 500 sq ft of floor space for equipment. The initial FDDCI memo [66] (published in February of 2010) stated that there were "more than" 1,100 federal agency data centers in 2009. The 25 Point Implementation Plan (published in December of 2010) states that there were 2,094 data centers in 2010. This would be a near doubling in the course of only a year. In October 2011, the Federal Chief Information Officer (CIO) redefined data center to include facilities of any size. Using this new definition, the OMB reported that there were 3,133 data centers. As agencies began applying the new definition, the number of data centers increased dramatically. In 2013, the Government Accountability Office (GAO) reported 6,836 data centers. However, 2,200 of these "data centers" were actually server closets in the county offices of the Department of Agriculture that typically only house a single server. The GAO also reported that the majority of federal agencies involved in the program had not yet submitted complete inventories or plans for consolidation. The GAO also reported that there is no established rule for validating any cost savings estimates [40, 44].

The DoD published a review of its implementation of the FDCCI in April 2013 [39]. The DoD owns the largest number of data centers (772) than any other federal agency

(based on definition as stated before). According to the report, the DoD planned on reducing the number of data centers by 30% by the end of 2013 and the number of servers by 25%. The report contains an interesting breakdown of consolidation plans by branch, as shown in Table 2.1.

Table 2.1: Data center consolidation plans by branch

Owner	Current	Planned
Air Force	137	117
Army	250	154
Navy	78	77
Combatant Commands	25	21
Other	282	163
Total	772	532

Noteworthy is that the Army is listed as having the most data centers, but also plans on reducing the most. The report states that the Army's consolidation efforts began in 2006 and that it plans on continuing even greater reductions to only 65 data centers by 2015. So although the effectiveness of the FDCCI at reducing the number of federal data centers is a matter of some debate, the initiative has spurred significant interest at the federal level in transitioning to cloud computing. It marked the beginning of a policy that put cloud computing at the forefront in federal acquisitions of new IT systems.

2.2.2 Federal Risk and Authorization Management Program (FedRAMP).

In the spring of 2009, Federal CIO Vivek Kundra began an inter-agency program labeled the Federal Cloud Computing Initiative [76, 78]. Coordinated through the General Services Administration, the FCCI would bring in technical experts from various federal agencies in order to foster the adoption of cloud computing. In September 2009, the FCCI

was more broadly announced by Mr. Kundra in coordination with the launch of a new cloud computing online shopping portal apps.gov [64]. Apps.gov was envisioned to be an inter-agency shopping portal where agency CIOs could purchase pre-approved cloud computing products quickly and easily. The goal was to move agencies away from their traditional, stove-piped IT acquisition approaches and foster a shared approach to IT acquisition.

One barrier to this inter-agency adoption was that each agency had its own internal processes for conducting certification and accreditation (C&A) of information systems. To address this issue, the FCCI established the Federal Risk and Authorization Management Program (FedRAMP). FedRAMP was designed to promote a "do once, use many times" approach to C&A. Cloud Service Providers (CSPs) could be certified once using a comprehensive set of C&A metrics, then the certification could be used by all federal agencies. This would significantly reduce the burden on the federal agencies to conduct the C&A on their own.

FedRAMP has taken some time to develop. The first formal policy memo was published in 2011 [104]. The FedRAMP Concept of Operations was published in February 2012 [55]. Initial operating capability was expected by the end of FY12, with full operational capability by summer of 2013. In May, 2013 there were only two certified CSPs. As of September 2013, the number of CSPs has grown to over nine and includes popular public CSP Amazon Web Services (AWS). So far, the focus appears to be on certifying Infrastructure-as-a-Service (IaaS) providers as all nine certified CSPs are for IaaS.

2.2.3 Example from Department of the Treasury.

The Department of the Treasury has been leveraging cloud computing primarily through hosting of its public facing websites. These sites include treasury.gov, recovery.gov, mymoney.gov, financialstability.gov, and makinghomeaffordable.gov [79]. These sites are all hosted through AWS. According to the Treasury's FDCCI report, it was

the first cabinet level agency to move its public facing site to the cloud where it saved over 13% in monthly costs relative to the previous hosting solution [102].

Like many federal agencies, the Treasury operates a large internal IT infrastructure as well. In their 2011 FDCCI report [102], they reported to operate 55 data centers with around 7,000 servers. This document also describes the Treasury's plans to leverage private cloud concepts with this internal infrastructure. The Treasury plans on designating a limited set of data centers referred to as "Department Data Centers" which would provide IaaS, PaaS and SaaS services to other internal Treasury bureaus. However, the document cites some consolidation concerns that may delay the implementation of the plan.

One interesting exception to the consolidation approach in the Treasury plan was the Supervisory Control and Data Acquisition (SCADA) systems of the Bureau of Engraving and Printing (BEP) and the Mint. The BEP website, which is hosted publicly on AWS, was recently taken offline due to an intrusion in 2010 [56]. These types of attacks often raise the most questions regarding public cloud computing security. It is important to recognize that these types of attacks rarely exploit any characteristic that is unique to cloud computing. While the details of how the site was compromised are not public, the most likely explanation is that the attack was possible through either some security vulnerability in the website code or out of date web server software. These are both issues that would affect a website regardless of where it is hosted. In these types of cases, it is important to differentiate between generic IT security and cloud security [54].

2.2.4 Example from Homeland Security.

The DHS also seems to be very active in adopting both public and private cloud computing. In October of 2011, DHS CIO Richard Spires described a variety of cloud initiatives in a congressional testimony given to the House Subcommittee on Cybersecurity, Infrastructure Protection, and Security Technology [97]. The testimony provides a fairly comprehensive view of DHS adoption of federal "Cloud First" policy.

DHS, like many other agencies, is focused primarily on the two deployment models of public clouds and private clouds. Mr. Spires cites three examples of how DHS is using public cloud services. The first is an identity proofing solution designed to meet United States Citizenship and Immigration Services' E-Verify Self Check requirement that allows individuals and employers to check their employment eligibility status seeking employment. Mr. Spires states that this is the first online E-Verify program to offer such as service directly to workers. Next, DHS uses the cloud to provide Enterprise Delivery Network (EDN) services for its public facing websites. This service is used to deliver content for over 70% of DHS websites. Mr. Spires cites that in 2009, when several federal agencies came under a denial-of-service attack (and DHS sites themselves experienced over a 100 fold increase in traffic) that the cloud EDN service was able to scale and as a result DHS sites experienced no outages. Finally, DHS plans to move additional sites to public cloud hosted infrastructures over the next two years. These include sites from U.S. Immigration and Customs Enforcement (ICE), United States Citizenship and Immigration Services (USCIS), and Federal the Emergency Management Agency (FEMA).

In addition to using public cloud services, DHS also plans on building private cloud services using its internal infrastructure. DHS plans on leveraging its two enterprise data centers as hubs for private cloud services to its internal components. DHS components would follow a cloud model of a pay-as-you-go system in order to use centralized services from these data centers. Some of the cloud services that are already implemented or currently planned include authentication, email, Sharepoint, project management and IaaS. DHS projects a cost avoidance savings of 8-10% from utilizing these private clouds. Mr. Spires cites security as a top concern and explains that DHS private cloud services will be used to house unclassified but sensitive data while public clouds are used for non-sensitive data.

III. Pairing Based Cryptography

The purpose of this chapter is to introduce the reader to some of the fundamental concepts used throughout this document with regards to pairing based cryptography. In a sense, this chapter aims to guide someone who is new to the world of pairing based cryptography through the mathematical minefield. Specifically, the goal is to convey implementation concepts at a level that prepares the reader to understand the mathematics presented in Chapter 4. The chapter begins by discussing some of the fundamental concepts of number theory and abstract algebra that are used in pairing based cryptography. Then some of the basic properties of bilinear maps and their implementations are presented. This helps the reader better understand the implementation presented in Chapter 5 which relies heavily on bilinear maps. Finally, some basic cryptographic concepts are presented so the reader has a better understanding of the security properties discussed in this research. The scope of this chapter is to briefly introduce the topics at a level so that the more formal mathematical constructions presented in Chapter 4 are more accessible. References are provided throughout the chapter to more established work for readers who desire a deeper understanding of the concepts.

3.1 Mathematics Primer

The world of pairing based cryptography has many real world practical applications, but also presents an interesting challenge to newcomers to the field. On the one hand, pairings can be treated in the abstract as black box functions. These pairings, also known as bilinear maps, have relatively straightforward algebraic properties that are fairly easy to understand and apply. However, if one desired to make one of these black boxes, the mathematical sophistication required increases significantly and one can quickly be lost in the mathematics. Moreover, these more sophisticated concepts tend to come from the

study of number theory, abstract algebra, and elliptic curves. These topics have a rich (perhaps daunting) set of terminology and concepts, only which a small subset that applies to pairings.

Even though pairings can largely be viewed as black boxes, not all pairing black boxes behave the same way. For example in [43], the authors explain the pitfalls and assumptions that need to be made about how these black boxes are influenced by the underlying implementation. This implies that a user of a cryptographic library that offers pairing support needs to understand some aspects of the underlying implementation in order to avoid misusing the library (perhaps leading to significant security flaws).

3.1.1 Abstract Algebra.

Abstract algebra is both a broad and deep field. This section serves to briefly introduce a few elementary concepts to help the reader understand the notation and mathematics presented in Chapter 4. This section also forms the basic foundation needed to understand the next section on bilinear maps. The material from this section including all definitions and theorems is primarily derived from the lecture notes for the AFIT course MATH 631 [38] and from the book on identity based encryption by Martin [74]. The latter contains an excellent introduction to the mathematics behind pairings. Another book by Chatterjee [30] also provides a good review of the relevant mathematics. These accepted theorems are presented in this section without proof, but the proofs can be found in any standard reference text on abstract algebra such as [**?**].

3.1.1.1 Notation.

Some common mathematical notation used throughout this document is presented in Table 3.1.

Table 3.1: Mathematical Notation

Symbol	Definition
$\mathbb{Z}$	The integers $\{..., -2, -1, 0, 1, 2, ...\}$
$\mathbb{Z}_n$	The integers modulo n. The modulus n typically indicates a composite number, while p or q indicate a prime modulus.
$\mathbb{Z}_n^*$	The non-zero elements of $\mathbb{Z}_n$
$\mathbb{F}_p$ or $\mathbb{F}_q$	A finite field of size p where p is prime. When q is used as the field size, it represents $q = p^k$ where k is a positive integer. This represents an extension field of degree k.
$E(\mathbb{F}_p)$	An elliptic curve defined over a finite field.
$\forall$	"for all". Typically used to refer to all elements of a set.
ϵ	Used to denote a *negligible function*. Informally, a negligible function is one whose inverse $\frac{1}{\epsilon}$ cannot be bounded by a polynomial function. See [73] for a formal definition.
$\in_U$	Indicates that the element is sampled uniformly (i.e., randomly) from the set.

3.1.1.2 Groups.

Groups form an important part of cryptography and are used extensively throughout Chapter 4. A group is essentially a set of items along with a binary operation, formally defined below:

Groups

(a) A **binary operation** on a set G is a function that assigns an element of G to every pair of elements of G, that is, a function $\star : G \times G \mapsto G$. Given $a, b \in G$, we denote $\star(a, b)$ simply as $a \star b$.

(b) A **group** is a set G along with a binary operation $\star$ that has:

(i) associativity: $a \star (b \star c) = (a \star b) \star c$ for all $a, b, c \in G$

(ii) an identity element: there exists $e \in G$ such that $e \star a = a = a \star e$ for all $a \in G$

(iii) inverses: for every $a \in G$, there exists $b \in G$ such that $a \star b = e = b \star a$

A group is called **commutative** (**Abelian**) if we further have:

(iv) commutativity: $a \star b = b \star a$ for all $a, b \in G$

As long as the set and the operation satisfy these properties, the set and operation are said to form a group. There are numerous mathematical constructs that satisfy these properties. The two that are of interest in this work are:

- The integers modulo a prime under multiplication ($\mathbb{Z}_p^*$)
- Points on an elliptic curve (discussed in more detail in §3.1.2)

When writing group operations, it can be convenient to borrow conventions from basic addition and multiplication. This is called either *additive* or *multiplicative* notation respectively. Take for example the following group operations:

$$((a \star a) \star a) \star b = a \star a \star a \star b$$

Note that we can drop the parentheses since group operations are associative. If we use an additive notation (the $\sim$ symbol here indicates the change in formal notation to additive notation), we use the rules of addition for combining like terms:

$$a \star a \star a \star b \sim a + a + a + b = 3a + b \sim (3a) \star b = 3a \star b$$

The a terms combine and an integer coefficient is used to determine how terms have been combined. This coefficient has higher precedence than the group operation and so the parenthesis can still be removed. Here is the same set of operations using multiplicative notation:

$$a \star a \star a \star b \sim a * a * a * b = a^3 b \sim a^3 \star b = a^3 b$$

When using multiplicative notation, the number of combined terms is represented in the exponent. Also, unlike additive notation, the group operator often does not need to be explicitly represented since it is implied by writing terms next to each other. Traditionally, additive notation is used only for commutative (also called Abelian) groups. Multiplicative notation is used more generally, often when it is either unknown or unimportant if the group is commutative. In this document, members of the group $\mathbb{Z}_p^*$ are written using multiplicative notation and points on elliptic curves are written additively.

The following are some important facts to know about groups:

For any group G:

(a) *The identity element is unique: there is exactly one element* $1 \in G$ *(when written in multiplicative notation) that satisfies* $1 * a = a = a * 1$ *for all* $a \in G$*. When written in additive notation, the identity element is* $0 \in G$ *such that* $0 + a = a = a + 0$.

(b) *Inverses are unique: for any* $a \in G$*, there is exactly one element* $b \in G$ *such that* $a * b = 1 = b * a$*; this element is usually denoted as* a^{-1} *when written in multiplicative notation or as* $-a$ *when written in additive notation.*

(c) $(a^{-1})^{-1} = a$ *for any* $a \in G$.

(d) $(a * b)^{-1} = b^{-1} * a^{-1}$ *for any* $a, b \in G$.

3.1.2 Elliptic Curves.

Elliptic curves are an important part of modern cryptography and are used as the underlying implementation for the bilinear maps described in the next section. This section briefly describes elliptic curves. For a more detailed treatment of elliptic curves and their uses in cryptography see [15]. Elliptic curves are curves that can be written in the form:

$$y^2 = x^3 + ax + b$$

When elliptic curves are plotted over the real numbers, the shape of the curve is influenced by the values of the constants a and b. The shape of the curve for various values of a and b are shown in Figure 3.1. However, when using elliptic curves in cryptography, elliptic curves are almost always used over a finite field. The result of plotting an elliptic curve over a finite field is shown in Figure 3.2.

There are two aspects of elliptic curves that make them useful for doing cryptography. First is that there is an operation in which the points of an elliptic curve form a group. In other words, it is possible to take any two points on an elliptic curve and apply the group operation to get another point on the same curve. This group operation fulfills all the requirements for points on an elliptic curve to be group elements (e.g., it is associative, there is an inverse for every element, there is an identity element). Although the group operation is a relatively simple algorithm, it is not described here and instead the reader is directed to [15] for details. It is noteworthy that elliptic curve points form a commutative group, and are therefore often written additively. This is the convention followed in this research.

The second aspect of elliptic curves that make them useful for cryptography is that certain mathematical problems are particularly difficult to solve when using elliptic curves. The difficulty of solving mathematical problems typically forms the basis of a

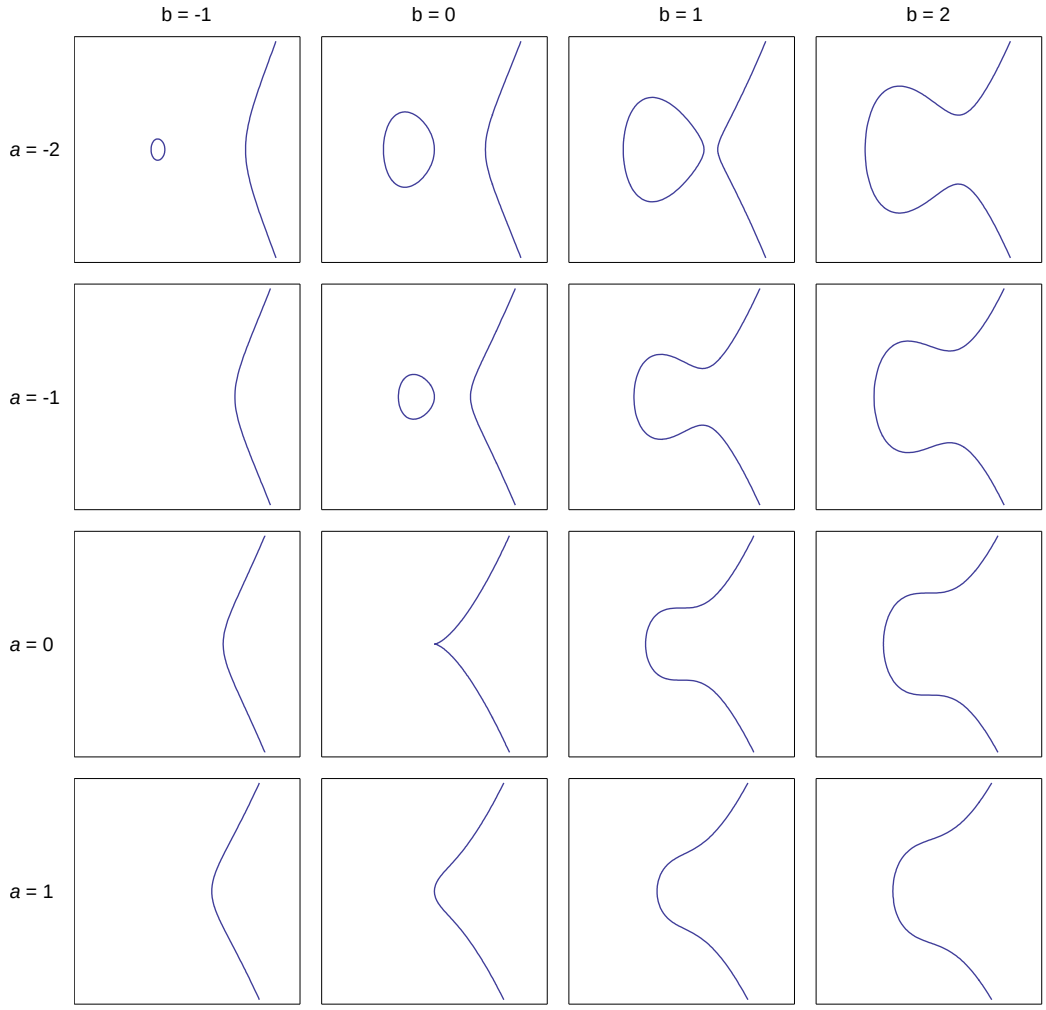


Figure 3.1: Elliptic curves over the real numbers with different values for constants a and b [3]

cryptographic algorithm, so when the problem is more difficult the resulting cryptographic algorithm can often be more efficient (i.e., use smaller parameters which often leads to better performance). In particular, the discrete logarithm problem (discussed in more detail later in *§3.3*) is harder for elliptic curve groups than it is for groups such as the multiplicative group of a finite field which is what cryptographic algorithms such as RSA use (even though the RSA algorithm is based on the difficulty of factoring, it has been shown that factoring reduces to solving discrete logarithms [5, 53]). Because of this,

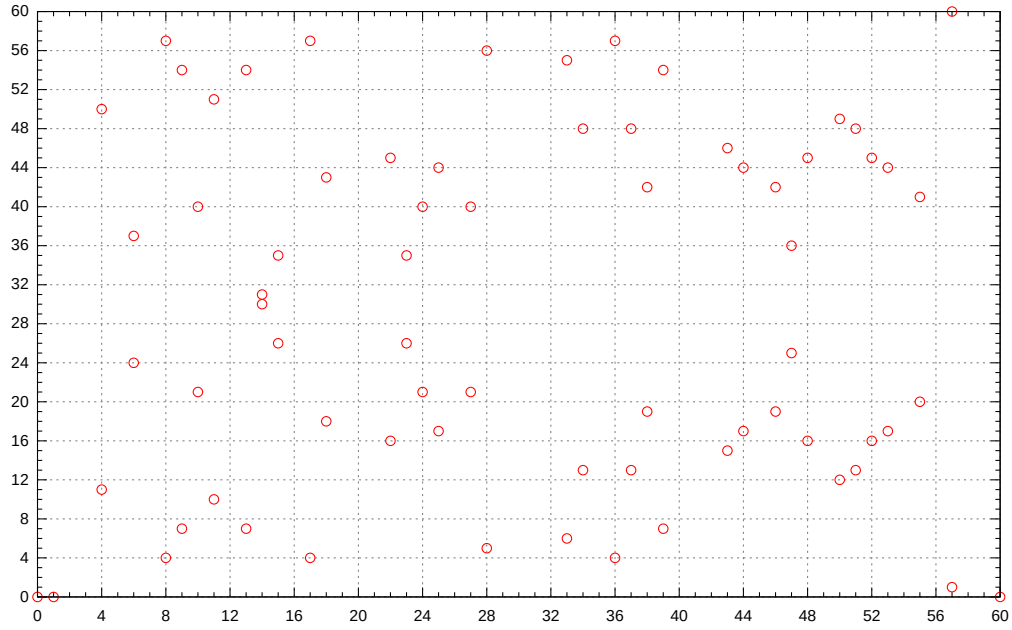


Figure 3.2: Elliptic curve defined over the finite field $\mathbb{Z}_{61}$ [3]

the parameters for elliptic curve group elements can be much smaller than RSA group elements.

3.2 Bilinear Maps (Pairings)

Bilinear maps (also known as pairings) have become an important part of cryptography, especially since their ground breaking role in solving the open problem of identity based encryption [18] in 2001. Since that time, pairings have played a role in increasingly more complex and expressive cryptosystems. Pairing based cryptography is something of a dichotomy. On the one hand, the algebraic rules for pairings are relatively straightforward and easy to apply and understand. On the other hand, the implementation of pairings requires complex mathematical concepts and algorithms.

This section on bilinear maps has two purposes. The first purpose of this section is to demonstrate the algebraic properties of pairings so that the system presented in Chapter 4 can be understood. Secondly, this section introduces some of the issues involved with

implementing pairings. Knowing some additional detail about how the pairings are actually implemented will also help the reader to understand the implementation performance details discussed in Chapter 5.

3.2.1 Pairing Types.

There are a few different ways to classify pairing implementations. One way to do so is by classifying an implementation as either symmetric or asymmetric. A symmetric pairing has the following form:

$$e : G_1 \times G_1 \rightarrow G_t$$

A symmetric pairing takes two elements from the same group and maps them to an element of the target group. An asymmetric pairing has the following form:

$$e : G_1 \times G_2 \rightarrow G_t$$

An asymmetric pairing takes elements from two different groups and maps them to a target group element. In both symmetric and asymmetric pairings, the order of all groups is the same. The order of the group can either be prime or composite. Current pairing implementations use points on an elliptic curve as the elements of G_1 and G_2. G_t is the multiplicative group of a finite field.

Another popular method of classifying pairings comes from the paper by Galbraith et. al [43]. Pairings are classified into one of three types:

Type-1: $G_1 = G_2$;

Type-2: $G_1 \neq G_2$ but there is an efficiently computable homomorphism $\phi : G_2 \rightarrow G_1$;

Type-3: $G_1 \neq G_2$ and there are no efficiently computable homomorphisms between G_1 and G_2;

For Type-2 and Type-3 pairings, there is also no efficiently computable homomorphism from G_1 to G_2 (since $G_1 = G_2$ for Type-1 pairings, there is a trivial homomorphism). In terms of the earlier classification, a Type-1 pairing would be considered a symmetric pairing, while Type-2 and Type-3 pairings are considered to be asymmetric. The choice of pairing implementation can have significant impact on what types of operations can be done and how efficient those operations will be. Some of these consequences are apparent to the cryptographic designer; for example, a security proof may rely on the fact that there is a homomorphism from G_2 to G_1 and therefore the scheme must be implemented with a Type-2 pairing. Other consequences can be more subtle. For example it may not be possible to hash to elements of G_2, elements of G_1 may be large (and therefore less efficient to compute with), or it may even be difficult to generate parameters with high security in polynomial time. Table 1 in [43] provides a good view of the functionality and limitations for each type of pairing.

In practice, Type-3 pairings are considered to be the most efficient [9, 10, 21, 41, 42]. As discussed in the next section, there are pairing friendly curves that support optimally efficient pairing algorithms at the 128-bit security level. Type-3 pairings also allow hashing to G_2, random sampling of G_2, use large characteristic finite fields (which make them immune from recent improvements [62] of discrete logarithm algorithms that require finite fields with small characteristic), and the system parameters are efficiently computable. As such, the research in this paper uses Type-3 pairings.

3.2.2 Pairing Algebra.

The defining characteristic of a bilinear map is of course its bilinearity. Take for example[1] the asymmetric case:

For $P_1, Q_1 \in \mathbb{G}_1$ and $P_2, Q_2 \in \mathbb{G}_2$ then:

$e(P_1, P_2 + Q_2) = e(P_1, P_2)e(P_1, Q_2)$ and $e(P_1 + Q_1, P_2) = e(P_1, P_2)e(Q_1, P_2)$.

[1]Note the convention used throughout this document is that the elements being paired are written using additive notation, while the result of the pairing operation is written using multiplicative notation

This leads to a relatively straightforward algebraic behavior. Exponents of the target element can be brought down as coefficients to either side of the pairing (or as exponents if the elements being paired are written using multiplicative notation). For example:

For $P_1 \in \mathbb{G}_1$ and $Q_2 \in \mathbb{G}_2$ then:

$e(P_1, 3Q_2) = e(P_1, Q_2)^3 = e(3P_1, Q_2)$

$e(3P_1, 4Q_2) = e(P_1, Q_2)^{3*4} = e(P_1, Q_2)^{12} = e(4P_1, 3Q_2) = e(P_1, 12Q_2)$

The example above could have been written completely in multiplicative notation by converting each coefficient to an exponent. Note how in the example, the coefficients are able to switch places not only with the exponent, but can be moved back down to either inner term. This ability to move around exponents and coefficients is what makes the bilinear map so useful for building cryptographic systems.

3.3 Cryptography

Cryptography is a diverse and complex field. This section is intended to briefly highlight some of the key aspects of cryptography that are used in this research. For more detailed treatment of cryptography, or of the topics listed in this section see [63, 73].

3.3.1 Semantic Security.

Semantic security is the idea that a ciphertext should leak no information about its corresponding plaintext. One way to achieve semantic security is through *perfect secrecy*, in other words through the use of a one-time pad. A one-time pad is a random bit string that is as long as the message. The pad is used as a key by performing an XOR operation, bit by bit, with the message to create the ciphertext. While this provides very strong security, it becomes very impractical with large messages and maintaining or transferring the one-time pads can become difficult. An alternative approach is to ensure that given a ciphertext, determining information about the plaintext should be computationally infeasible by any probabilistic, polynomial-time algorithm (PPTA). This idea was first introduced by Goldwasser and Micali as probabilistic encryption [49].

Goldwasser and Micali show that semantic security is equivalent to the notion of *ciphertext indistinguishability*. Ciphertext indistinguishability typically takes one of two forms, depending on the information available to an adversary. These two forms are chosen plaintext attack (CPA) and chosen ciphertext attacks (CCA). Typically the semantic security definition for a cryptosystem is presented in terms of a game between a challenger and an adversary. The challenger performs any setup work and presents any public parameters to the adversary. To model a chosen plaintext attack, the adversary is able to choose two, equal length messages to be encrypted and submits the messages to the challenger. The challenger picks one of the messages at random, encrypts the message and then returns the ciphertext back to the adversary. The adversary must determine from the ciphertext which of the two messages was chosen and makes a guess. The adversary is said to have an negligible advantage if the guess is correct 50% of the time plus some negligible amount. This negligible amount is usually defined mathematically as a function (represented in this document as ϵ) whose inverse cannot be bounded by a polynomial. The CCA game is very similar except that the adversary is given access to a decryption oracle, which allows decryptions of any ciphertext the oracle is given. The adversary can use this oracle on any ciphertext except the challenge ciphertext that is returned to it (otherwise the game would become trivial). If the adversary is allowed to adjust its strategy based on prior decryptions, this is called adaptive CCA (or sometimes CCA2). Non-adaptive CCA is sometimes called CCA1. The target level of security for a public key cryptosystem is either CCA1 or CCA2 security.

3.3.2 Complexity Assumptions.

Most cryptosystems are built from the idea that certain problems in mathematics are generally assumed to be computationally hard [75]. This means that for these problems, there are no known efficient (i.e., polynomial time) algorithms that can solve them. Some problems are considered to be widely studied and thus there is more confidence in the

validity of the assumptions. The assumptions corresponding to these problems are referred to as standard assumptions, and cryptosystems that only require these types of assumptions are said to be secure in the standard model. Standard model security is typically the goal for any cryptosystem as it provides the greatest confidence in terms of security; however, this security may often be difficult to obtain. Often, in order to prove security cryptosystems, one must rely on stronger[2] assumptions instead of weaker standard assumptions.

One way to define assumptions is through a statistical game. An algorithm is said to have an advantage in the game if it is able to produce results that are different than random guessing. The following is a list of a few common standard assumptions presented in this format:

Discrete Logarithm (DL)[1]: Let $\mathbb{G}$ be a cyclic group with generator g. Let $\mathcal{D}$ be the following distribution:

$$\mathcal{D} = (\mathbb{G}, g, g^x);\ x \in_U \mathbb{Z}$$

The DL problem to compute $x \in \mathbb{Z}$ when given $\mathcal{D}$. Let $\mathcal{A}$ be an polynomial-time algorithm that takes $\mathcal{D}$ as its input and produces as its output $\bar{x} \in \mathbb{G}$. The advantage of an algorithm $\mathcal{A}$ in solving the DL problem is given by:

$$\mathrm{Adv}_{\mathcal{A}}^{DL} = |Pr[x = \bar{x} \leftarrow \mathcal{A}(\mathcal{D})]|$$

The (ϵ, t)-DL assumption holds in $\mathbb{G}$ if for any adversary $\mathcal{A}$ running in time at most t, $\mathrm{Adv}_{\mathcal{A}}^{DL} \leq \epsilon$.

The discrete logarithm problem is a classical hard problem used in many cryptosystems. It represents a class of problems where computing forward is computationally easy, but reversing that computation is expensive. In this case, computing the discrete exponential

[2]With complexity assumptions, weaker assumptions assume less than strong assumptions. Therefore, weaker assumptions are usually more desirable in terms of security than strong assumptions. It is an ironic twist of mathematical terminology where weak is good and strong is not as good.

g^x in the problem above (i.e., computing forward) is a very efficient computation. However, the best known algorithms for reversing this computation (i.e., taking the discrete logarithm) are expensive. The cost of the algorithm depends on the structure of the underlying group. For generic groups (i.e., no special structure is assumed in the structure of the group other than those specified in the group definition), the best known algorithms are exponential. For some groups, such as points on elliptic curves, the generic algorithms are the most efficient that are known. For other groups, such as $\mathbb{F}_p^*$, then the structure of the group can be exploited and sub-exponential (but not polynomial) algorithms are known to exist. The security of a cryptosystem is dependent on the efficiency of these algorithms. Therefore, groups that have more efficient algorithms (e.g., $\mathbb{F}_p^*$) need to be larger in order to provide the same level of security as groups with only generic algorithms (e.g., elliptic curve groups).

Decisional Diffie-Hellman (DDH)[1, 63]: Let $\mathbb{G}$ be a cyclic group with generator g. Let $z \in_U \mathbb{G}$. Let $\mathcal{D}$ be the following distribution:

$$\mathcal{D} = (\mathbb{G}, g^x, g^y);\ x, y \in_U \mathbb{Z}$$

The DDH problem is to determine, given $(\mathcal{D}, h \in \mathbb{G})$ if $h = g^{xy}$ or if $h \in_U \mathbb{G}$. Let $\mathcal{A}$ be an polynomial-time algorithm that takes $(\mathcal{D}, h)$ as its input and produces 1 as its output if $h = g^{xy}$ and 0 otherwise. The advantage of an algorithm $\mathcal{A}$ in solving the DL problem is given by:

$$\mathrm{Adv}_{\mathcal{A}}^{DDH} = |Pr[A(\mathcal{D}, g^{xy}) = 1] - Pr[A(\mathcal{D}, z) = 1]|$$

The (ϵ, t)-DDH assumption holds in $\mathbb{G}$ if for any adversary $\mathcal{A}$ running in time at most t, $\mathrm{Adv}_{\mathcal{A}}^{DDH} \leq \epsilon$.

The DDH assumption is the classical decision type assumption. Many security proofs rely on the difficulty of determining if group elements have a specific structure or if they

are randomly sampled. All of the assumptions presented in Chapter 4 follow this pattern. The DDH assumption is considered to be stronger than the discrete logarithm assumption, because there are groups where the DL assumption holds and the DDH assumption does not. One relevant example is with bilinear groups. If a symmetric pairing is known to exist for $\mathbb{G}$, then the adversary can easily determine if $h = g^{xy}$. The adversary is given g^x and g^y. The adversary can pair these elements and get an element m of the target group $e(g^x, g^y) = m$. When the adversary receives h, it pairs it with g. If $h = g^{xy}$, then this pairing becomes $e(g, g^{xy}) = e(g, g)^{xy} = e(g^x, g^y) = m$. If h is random, then the result is not equal to m. Thus, the adversary can guess correctly after only a single (efficient) pairing operation. To deal with the fact that the DDH problem is easy in bilinear groups, specialized DDH assumptions have been made specifically for pairings. An example of such an assumption is the Decisional Bilinear Diffie-Hellman Assumption for Type-3 pairings (DBDH-3) which is defined in Chapter 4.

3.3.3 Dual System Encryption.

When cryptosystems based on bilinear pairings were first introduced, they often relied on a proof technique that involved random oracles [11]. Cryptosystems whose security proofs use random oracles are said to be secure in the random oracle model. There are known limitations to random oracles and some in the cryptographic community see cryptosystems that rely on random oracles as being disadvantaged relative to cryptosystems that rely on more standard assumptions. Eventually, identity based encryption systems were built that did not rely on random oracles, but required a *selective-id* security model. This selective-id model required adversaries to announce their target identity before even seeing the public parameters. Eventually, fully secure systems in the standard model were presented, but required large public parameters [109].

Typically to show that a system is secure, a simulator is used to play out the required security game. Instead of being allowed to choose its own parameters, the simulator is given

an instance of some hard problem. Often this problem is in the form of distinguishing some element as either having a particular form or being randomly selected from the group. The simulator must use the terms in the problem to *simulate* the cryptosystem (i.e., perform all the relevant calculations for creating ciphertexts and keys). In order to achieve full security, a more powerful approach to building the simulator was required.

A breakthrough came with the introduction of the dual system encryption methodology by Waters [109]. In a dual system encryption system, there are three types of components: normal (sometimes called fully functional), semi-functional, and nominally semi-functional. This applies to both ciphertext and keys. Table 3.2 shows which combinations lead to successful decryption operations and which combinations lead to failure. Essentially, whenever semi-functional components interact with normal components, the decryption is successful. However, whenever two semi-functional components are used, the decryption fails. If nominally semi-functional components are used, there is typically some specific condition that must be met in order for decryption to be successful. This is typically used so that the simulator cannot test if a ciphertext it is creating is semi-functional or not. In this case, the simulator can typically create either normal or nominally semi-functional ciphertext and can only create either normal or nominally semi-functional keys. The simulator then cannot detect if the ciphertext is normal or (nominally) semi-functional, since decryption will be successful in either case.

Table 3.2: Dual System Components (w/HP indicates with high probability)

Keys \ Ciphertext	Normal	Semi-Functional	Nominally Semi-Functional
Normal	✓	✓	✓
Semi-Functional	✓	Fail	Fail w/HP
Nominally Semi-Functional	✓	Fail w/HP	✓ (if conditions are met)

The dual system encryption strategy works by making an argument over a sequence of games. The first game in the sequence is a real security game like those described in §3.3.1. The last game is a game where the advantage of the adversary is known to be negligible (e.g., the adversary's guess is of a truly random value). The games in the middle of the sequence are games that leverage the ability of the adversary to detect changes of the components from normal to semi-functional to also solve some complexity assumption (i.e., a known hard problem). For example, a game might reduce the ability of the adversary to distinguish between normal ciphertexts and semi-functional ciphertexts to solving DBDH-3. The use of semi-functional components allows the challenger/simulator to answer key queries from the adversary without requiring any information up front from the adversary (unlike the selective-id model described above). The dual system encryption approach has been a very powerful one for building pairing based cryptosystems with full security.

IV. MA-AHASBE: Multi-Authority Anonymous Hierarchical Attribute-Set Based Encryption

This chapter describes the merger of two existing pairing based cryptosystems, an anonymous hierarchical identity based encryption (AHIBE) scheme and a ciphertext-policy attribute-set based encryption (CP-ASBE) scheme, into a single system that provides a very rich feature set. The new system, dubbed MA-AHASBE (Multi-Authority Anonymous Hierarchical Attribute-Set Based Encryption), allows users to create ciphertext that enforces an access policy, allows policies to include attributes from multiple authorities in either a centralized or decentralized environment, allows for protected attributes that are not disclosed in the ciphertext, and allows an authority to delegate key generation capabilities to a hierarchy of subdomains. The system is described using asymmetric bilinear maps over pairing friendly elliptic curves, which have been shown to be the most efficient pairings in practice. The system is designed to support the security requirements of a large, federated enterprise attribute based access control (ABAC).

4.1 Introduction

The introduction of pairing based cryptography ignited the imaginations of cryptographers and created a wide array of unique and interesting cryptosystems. Although pairings had been known to cryptographers for some time in the context of cryptanalysis, the first pairing based encryption approach was proposed in 2001 by Boneh-Franklin [18]. This new system solved a long standing open problem proposed by Shamir [95] regarding the existence of identity based encryption (IBE) systems. In an IBE, the user's public key is a fixed identifier such as an email address or URL. Over time, the concepts evolved from identities to attributes. A user could have attributes assigned to them by an authority through their cryptographic keys and ciphertexts could be created that enforced an access policy. These

types of systems are known as ciphertext-policy attribute based encryption (CP-ABE) systems. This chapter presents a multi-authority, anonymous, hierarchical, attribute-set based encryption (MA-AHASBE) scheme, which is a type of CP-ABE.

MA-AHASBE was developed to fill a gap in CP-ABE systems. Many CP-ABE systems exist and come with a variety of features which are described in more detail in the next section. While each system presents a unique set of novel features, no single system possesses enough total features required for a large, federated system. Two systems in particular, Lewko-Waters anonymous hierarchical identity based encryption (LW-AHIBE) [89] and ciphertext-policy attribute-set based encryption (CP-ASBE) [17], each provide a powerful set of capabilities. However, they come from slightly different worlds, IBE and CP-ABE. MA-AHASBE is essentially the bootstrapping of LW-AHIBE into an CP-ASBE system in order to leverage the benefits of both systems. Some additional novel machinery is added to allow attributes to originate from multiple authorities. The end result is a CP-ABE with a rich set of features designed to support an attribute based access control (ABAC) [82] paradigm in a large, federated enterprise system.

In the following sections, we provide background information that includes definitions, requirements, and a brief review of the related work. We then describe the notation and functions used throughout the chapter. Then we give a detailed description of the data structures used in MA-AHASBE to represent keys and policies. This section also presents the concepts of MA-AHASBE in an informal setting, with motivating examples. We then give a summary of the algorithms that formally define MA-AHASBE, followed by a proposed construction. We conclude with a description of the security models and security proof, as well as some final thoughts on future work.

4.2 Background

4.2.1 Enterprise Attribute Based Access Control Requirements.

We first discuss some requirements we argue are necessary for an attribute based encryption (ABE) scheme to serve as an attribute based access control (ABAC) enforcement mechanism in a large, federated, enterprise environment. These requirements provide a common framework to discuss both the strengths and weaknesses of other systems for supporting such an environment. It also provides insight to the areas where MA-AHASBE helps to provide missing capability.

4.2.1.1 Definitions.

Here we provide a short list of term definitions. The ABAC and enterprise definitions come from the NIST guidance [58] on enterprise ABAC. The domain definition is our own contribution.

Attribute Based Access Control (ABAC) An access control method where subject requests to perform operations on objects are granted or denied based on assigned attributes of the subject, assigned attributes of the object, environment conditions, and a set of policies that are specified in terms of those attributes and conditions.

Enterprise A collaboration or federation among entities for which information sharing is required and managed.

Domain A hierarchical organizational unit of an enterprise entity.

4.2.1.2 Requirements.

This section briefly describes the requirements we argue are necessary for an ABE to support enterprise level ABAC. These requirements reflect our interpretation of the guidance provided by NIST applied to the context of ABE systems. For each requirement, we give a definition as well as a rationale for why it is included. This list is not meant to

be exhaustive, though it is meant to represent a minimum set of requirements that an ABE needs in order to support an enterprise ABAC.

Multiple Authorities The system must allow attributes from cooperating, though authoritatively distinct, authorities within a single access control policy.

Rationale: This capability allows a federated enterprise to share a common attribute and policy infrastructure.

Hierarchical Structure The attributes in the system must belong to a hierarchical structure that reflects the domain ownership of the attribute.

Rationale: This is necessary in order to properly determine which attributes belong to which organizations. A large, federated organization may have thousands of attributes that each have domain specific meanings. Allowing hierarchical organization of these attributes is necessary in order to properly administer them.

Hierarchical Computation The computational burden of key generation and management must primarily take place at the same domain in a hierarchy as the ownership of the relevant attributes.

Rationale: Key generation and management may become a bottleneck for organizations with hundreds of thousands of users. A system must allow for hierarchical computation in order to distribute the workload more evenly across the enterprise. Note that this does not prevent a domain from utilizing third party computation or storage resources. Instead the purpose is to limit the computational dependency between domains or between enterprise entities.

Hierarchical Autonomy Each domain of the hierarchy must be able to generate arbitrary attributes that reflect its ownership with a minimum amount of coordination with other enterprise entities, including domains within the same entity.

Rationale: This requirement provides maximum flexibility at the ABE level to define

the hierarchical structure of attributes. Note that this requirement applies to the ABE, not necessarily to other factors (e.g., organization policy) that determine if a domain *should* generate a particular attribute. Also this requirement reinforces the requirement that a domain cannot generate attributes for other domains (i.e., each attribute generated reflects the domain that owns that attribute).

Attribute Protection Policies must provide a mechanism to prevent sensitive attributes from being disclosed to unauthorized parties.

Rationale: Often the attributes used in access policies are as sensitive as the objects the policies are protecting. The attributes used in a policy can disclose to third parties what type of information is being protected or may indicate its value. Some mechanism to protect these sensitive attributes is critical to the overall security posture of the system.

Key Revocation The system must provide a mechanism to revoke user keys that prevents such keys from satisfying policies created after the revocation.

Rationale: This supports the common scenario of users leaving the system or losing privileges. It is assumed that users will be able to decrypt messages created before their key was revoked. The critical part of the requirement is that new messages will be protected from users with revoked keys.

4.2.1.3 Features.

Here we list two features that on their own do not justify themselves as requirements, but are useful in the comparison of systems. We consider systems that provide these features to be more supportive of an enterprise environment than those that do not.

Compound Attributes The system should allow single attributes to be bound together into compound attributes such that they can only be used together to satisfy a policy.

Rationale: Compound attributes correspond well to scenarios where attributes

should only apply as a group. For example a course and a grade could be represented by two attributes, but should be semantically bound together so that the grade cannot be used for courses to which it is not bound. In an enterprise environment, this often occurs when people have a combination of roles and membership such as "Supervisor" and "Human Resources". Binding these attributes into a compound attribute might indicate that the user is a *supervisor* in the Human Resources department. Keeping them separate may indicate that the user is a supervisor (somewhere, not necessarily of Human Resources) and a *member* of the Human Resources department.

Numeric Attributes and Comparisons The system should provide a way to generate numeric attributes and should allow policies to make numeric comparisons (e.g., equality, less than, greater than).

Rationale: Numeric attributes and their comparisons in policies are important in many scenarios. Critically, this includes the ability to check timestamps as well as numeric levels of access (e.g., distinguishing a Level 3 supervisor from a Level 5 supervisor).

4.2.2 Related Work.

A significant body of work has been done in the area of CP-ABE systems. The first construction was given by Bethencourt et. al in [13]. Many others have followed that provide various enhancements such as proofs in the standard model [32, 52], policy secrecy [69, 80], and multi-authority [28, 29, 70, 72]. In [71, 108], the authors present work that use identities in a hierarchical IBE (HIBE) as attributes in an ABE. The dual system encryption technique was first developed by Waters [109], and has been a very important development as it allows many systems to be proven with fewer security restrictions. In [105], the authors extend CP-ASBE [17] to a hierarchical scheme called HASBE. However, the delegation in HASBE is restrictive in the sense that at each level in the hierarchy, each domain can

only restrict and cannot add to the set of attributes available to the next lower level in the hierarchy. While proposed as a feature by the authors, we see this as a violation of hierarchical autonomy. In HASBE, if a subdomain needs access to additional attributes after its creation, there is a substantial cost in adding that attribute.

There has also been some recent work that uses a pairing based system to support role based access control RBAC [111]. Attribute based access control (ABAC) can be seen as a generalization of RBAC [58]. The authors in [111] provide a comparison of many types of different systems including hierarchical identity based encryption, attribute based encryption, role based encryption, and hierarchical key management. The comparison looks very unfavorable to ABE systems, however the reality is more subtle. For example, ABE is cited as not having constant size ciphertext or constant size keys when supporting only one role. However, the first system proposed in [111] achieves constant size ciphertext by limiting the encryption to a single target role. This is analogous to only having a single attribute in a policy. Since the depth of the hierarchy in MA-AHASBE is bounded by a constant at system setup (thus bounding the size of the attributes), ciphertext in MA-AHASBE scales only with the number of attributes in the policy. A policy with a single attribute (and the associated ciphertext) in MA-AHASBE is also bounded by a constant in this case. Furthermore, key structures in MA-AHASBE scale with the number of attributes a user possesses. So if the number of attributes assigned to a user is bound by a constant (in this comparison the constant is 1), then key size is also constant. The authors in [111] extend their first system by allowing multiple target roles, which forces the ciphertext to scale with the number of roles. Finally, the decryption algorithm in both systems scales linearly in both the number of users in the system as well as with the number of ancestor roles (parent nodes in a role hierarchy) of the target roles. While the authors suggest this can be mitigated since the bulk of the computation can be done without secret data (and hence

can be securely outsourced to a third party), this seems to be overly burdensome when deploying to enterprise environments with large numbers of users and roles (or attributes).

There has also been work applying attribute-based encryption systems and ABAC to an enterprise messaging system called Attribute-Based Messaging (ABM) [16]. The focus of ABM is on the authorization and delivery of email messages based on attributes, but ABM uses a CP-ABE to provide end-to-end confidentiality. The CP-ABE used by the authors of ABM is the system proposed in [13] and implemented using the CP-ABE toolkit. The Bethencourt CP-ABE is similar in its delegation capability as HASBE, where delegated keys may only generate attribute keys for a restricted subset of attributes. The authors of [16] estimate that their implementation can handle up to 68,000 users based on a weekly key refresh rate of one key per week. The authors note that without a more robust key management solution, key generation for the ABE would likely become the bottleneck for the entire system at larger scale. Since MA-AHASBE offers a more robust and flexible key delegation mechanism, it could serve as a drop-in replacement in a system such as ABM.

The work on MA-AHASBE relies fundamentally on three previous works. MA-AHASBE bootstraps the AHIBE system described in [89] into an attribute-set system described in [17]. The CCA transform described in [19] is then applied, leveraging the capabilities of the AHIBE to be able to generate private keys for arbitrary identities. Since the design of MA-AHASBE is heavily influenced by both CP-ASBE and LW-AHIBE, we attempt to provide enough material in the description to make this chapter self-contained. However, the reader who is unfamiliar with these two systems is highly encouraged to review the cited references as they naturally provide a much more detailed motivation and exposition.

4.3 MA-AHASBE

4.3.1 Preliminaries and Notation.

4.3.1.1 Bilinear Pairings.

The construction for the MA-AHASBE system is described using asymmetric pairings, also known as Type-3 pairings [43]. An asymmetric pairing is a bilinear map from $\mathbb{G}_1 \times \mathbb{G}_2$ to $\mathbb{G}_T$ where all groups have prime order. A bilinear pairing $e : \mathbb{G}_1 \times \mathbb{G}_2 \rightarrow \mathbb{G}_T$ must have the following properties:

1. *Bilinear*: For $P_1, Q_1 \in \mathbb{G}_1$ and $P_2, Q_2 \in \mathbb{G}_2$ then:

   $$e(P_1, P_2 + Q_2) = e(P_1, P_2)e(P_1, Q_2) \text{ and } e(P_1 + Q_1, P_2) = e(P_1, P_2)e(Q_1, P_2).$$

2. *Non-degenerate*: If $e(P_1, P_2) = 1_T$, the identity element in $\mathbb{G}_T$, then either P_1 is the identity of $\mathbb{G}_1$ or P_2 is the identity of $\mathbb{G}_2$.

3. *Efficiently computable*: The function e should be efficiently computable.

Three types of pairings have been identified. Type-3 pairings have the shortest representations and the fastest algorithms [89]. Well known examples of such pairings exists as well as elliptic curves that support their efficient computation [35]. Such curves are known as pairing-friendly curves [10, 41]. The term *efficient* is used in the informal sense. Pairing computations on modern desktop hardware for finite fields of relevant size for practical security typically execute on the order of milliseconds.

4.3.1.2 Notation.

For a set $\mathcal{S}$, the notation $x \xleftarrow{U} \mathcal{S}$ means that the element x is selected randomly from $\mathcal{S}$ according to a uniform distribution. For two integers $a,\ b$, the notation $[a, b]$ represents the set $\{x \in \mathbb{Z} : a \leq x \leq b\}$. Let $\mathbb{G}$ be a finite cyclic group and $\mathbb{G}^\times$ denote the set of generators of $\mathbb{G}$. Fix generators $P_1 \in \mathbb{G}_1^\times$ and $P_2 \in \mathbb{G}_2^\times$. The discrete logarithm of an element $Q \in \mathbb{G}_i$ to base P_i is written as $\mathsf{dlog}_{P_i} Q$ where $i = 1, 2$. The notation $R_1 \sim R_2$ indicates that $\mathsf{dlog}_{P_1}(R_1) = \mathsf{dlog}_{P_2}(R_2)$. Elements in angle brackets represented by $\langle ... \rangle$ indicate a tuple

(an ordered list of elements), while braces such as {...} represent a set. Functions used throughout this chapter are shown in Table 4.1.

Table 4.1: Functions

Function	Definition
$depth(\mathbf{i})$	Returns the number of elements in the label **i**.
$index(\tau)$	Returns the index of the access policy node τ. The children of a node are uniquely indexed from the set $[1, n_c]$ where n_c is the number of children.
$att(t)$	Returns the attribute assigned to a leaf node in an access policy or the index into an array of protected attribute ciphertexts.
$pindex(\mathsf{attr})$	The index in the array of protected attribute ciphertexts that represents the attribute attr.
$parent(\tau)$	Returns the parent node in an access policy tree.
$\mathsf{MAC}_k(A)$	The result of executing a message authentication code algorithm on input A with key k.
$\mathcal{P}(\mathbb{A})$	Similar to the policy satisfaction algorithm described in [17]. The function takes an access policy $\mathcal{P}$ and a key structure $\mathbb{A}$ and returns a set (possibly empty) of labels at each node that can be used to satisfy the policy at that node.

4.3.2 MA-AHASBE Overview.

This section introduces the informal mechanics of how MA-AHASBE operates. This work is a combination of two previous works, LW-AHIBE [89] and CP-ASBE [17]. The goal is to leverage the identities from the AHIBE scheme as protected attributes in the CP-ASBE scheme. Since the attributes are not fixed during system setup, MA-AHASBE system would be considered a large universe system. This section introduces the notions

of attributes, attribute hierarchies, policies, and key structures in the context of small, tangible examples. This informal description is intended to provide intuition for the formal definitions provided later in the chapter.

4.3.2.1 Decentralization.

In order to provide the ability to work with different authorities, a trusted central authority must create a set of global parameters. The global parameters scale linearly with the number of users in the system since they contain global identifier information for each user. The trusted central authority can be replaced by any decentralized secure multi-party computation (SMPC) protocol that supports finite field arithmetic and random number generation. Furthermore, the results of the computations (global identifiers) are all made public and no secret information from the global setup is needed for any other computations (aside from generating more identifiers). This limits the need for the SMPC protocol to only computations required during global setup. However, depending on the SMPC protocol used, further considerations must be made. For example, SMPC protocols that assume a dishonest majority of participants often cannot simultaneously support robustness to node failure (or refusal to participate). An example is when a single party may unilaterally halt the computation process if it chooses to stop participating in the protocol, leaving the remaining members unable to recover. The effect on a decentralized MA-AHASBE implementation is that there may not be enough global identifiers generated when the protocol stops executing to support the total number of users in the system. One way to mitigate this is to pre-generate enough global identifiers during global setup (and not proceed with any other algorithms until global setup is complete) so that each user has a unique identifier for the lifetime of the system. This might be a large up front computational cost if there are a large number of users. It may also be difficult to predict a priori either the total of number users the system will support or even the system lifetime itself. This particular risk could be mitigated by choosing a protocol that allows fewer dishonest

participants but provides more robustness guarantees. With less risk of the protocol halting due to a small number of participants, members of the SMPC might then be willing to use the system while computing the identifiers on-demand as new users join. These types of tradeoffs along with other details on replacing the trusted central authority with an SMPC, while important to any real world decentralized deployment, are considered orthogonal to the principal design of MA-AHASBE and we do not address them further in this chapter.

4.3.2.2 MA-AHASBE Fundamentals.

MA-AHASBE belongs to the category of ciphertext-policy attribute based encryption or CP-ABE and therefore has notions of attributes and policies. We begin by describing attributes. In MA-AHASBE, attributes belong to an attribute hierarchy which is represented as a tree. Nodes in this hierarchy belong to one of four categories: *global parameter*, *authority parameter*, *domain*, and *leaf*. There is just one global parameter node that represents the root of the hierarchy. The immediate children of the global root node must be authority parameter nodes. The immediate children of the authority parameter nodes must be domain nodes. The domains that are immediate children of authority parameter nodes are called *top level domains*. Domains may have either domain or leaf nodes as children. Nodes without any children are leaf nodes and leaf nodes must be children of domain nodes (not global or authority parameter nodes). Attributes in MA-AHASBE are fundamentally identities in LW-AHIBE. As with most HIBE systems, the attributes are described as a tuple of identities. We write elements of the tuple as strings in double quotations (e.g., "University", "Physics", or "Student") and the tuple itself within angle brackets (e.g., ⟨"University", "Physics", "Student"⟩). Often when written as a tuple, the double quotes are dropped from the individual entries (e.g., ⟨University, Physics, Student⟩). The order of the elements represents a path along the attribute hierarchy. An attribute name is *fully qualified* if it contains the path from a global parameter node to a leaf node. When the context is clear, an attribute will usually be referenced by only its last few elements

(e.g., "Intern", ⟨Registrar, Fail⟩). When the elements given is enough to uniquely identify an attribute, the attribute is called *inferentially qualified*. Otherwise the attribute is called *partially qualified*. For example, in Figure 4.1, "Student" is partially qualified since it could refer to the attribute in either the "Physics" or "Electrical Engineering" domain. The attribute references ⟨Physics, Student⟩ or "Intern" are inferentially qualified.

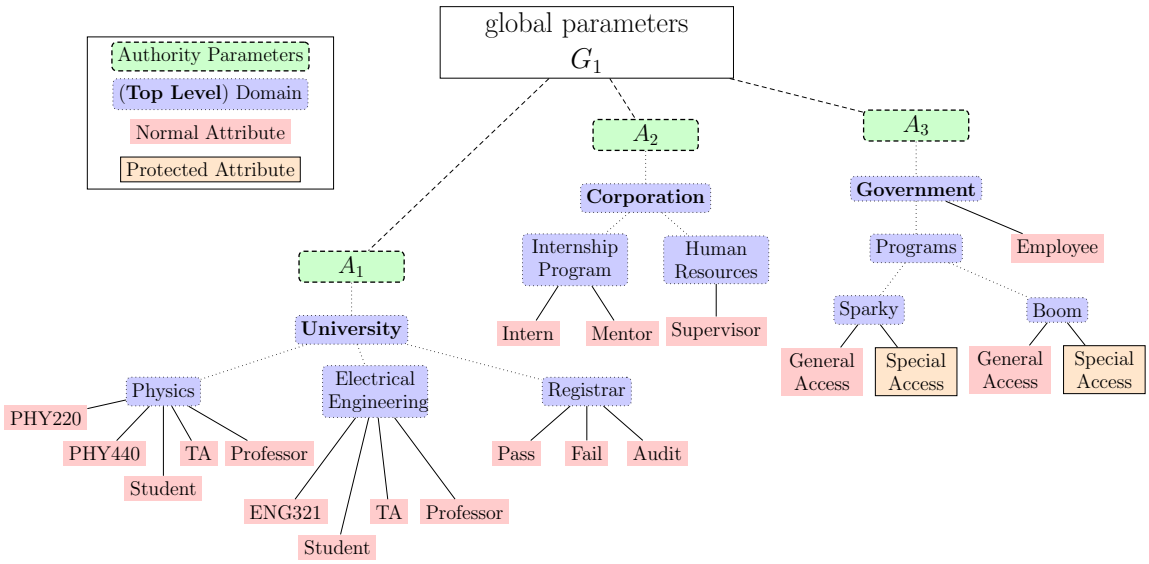


Figure 4.1: Full Attribute Chart

We use the example shown in Figure 4.1 throughout this section which represents a hierarchy of attributes. In Figure 4.1, there are three authority parameter nodes that each share the same global parameter root node. In MA-AHASBE, each global and authority parameter node is uniquely identifiable (in Figure 4.1 the global identifier is G_1 and the authority identifiers are A_1, A_2 and A_3). Each authority parameter node represents an authority in the system. Examples of domain nodes include "University" (a top level domain), "Registrar", "Programs", and "Sparky". Note that domains can have other domain nodes as children. A domain is also referred to as a *subdomain* of its parent. In the example, "Government" is a subdomain of the authority parameter node A_3, and "Sparky"

is a subdomain of "Programs". The depth of the hierarchy is defined to be the length of the longest path from an authority parameter node to a leaf node. For example, the depth of the hierarchy shown in Figure 4.1 is 4 (see for example the path $\langle A_3$, "Government", "Programs", "Sparky", "General Access"$\rangle$). Finally, Figure 4.1 shows a special type of leaf node called a *protected* node. These nodes are used when the attribute itself is sensitive to disclosure and will be discussed in greater detail next when we discuss policies.

Policies in MA-AHASBE follow the same mechanics as those in CP-ASBE. The policy is a tree of policy nodes where each leaf node represents an attribute and each non-leaf node acts as a threshold node. The threshold indicates how many children nodes must be satisfied in order to satisfy that particular node's policy. The threshold t is a value in the range $[1, n_c]$ where n_c is the number of children nodes. A threshold of 1 is equivalent to an *OR* operation and a threshold equal to n_c is equivalent to an *AND* operation. The most straightforward policies are those where all the attributes come from the same domain. Such a policy is shown in Figure 4.2.

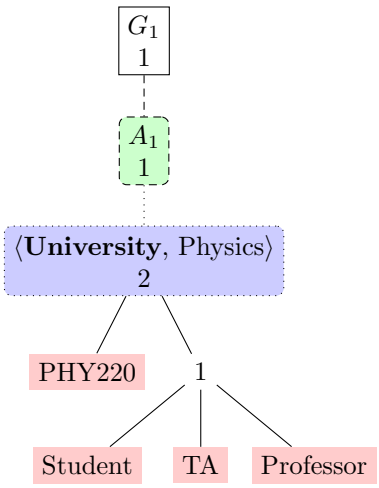


Figure 4.2: Example policy with attributes from a single domain

Policies consist of four types of nodes that play similar (though not identical) roles as attribute hierarchy nodes. They are *global*, *authority*, *domain*, and *leaf*. Policies can contain multiple global nodes, however, there is typically only one. All global nodes must all have the same label, though they can have different threshold values. The immediate children of global nodes must be either global nodes or authority nodes. Authority nodes are labeled according to their corresponding identifier in the attribute hierarchy. The immediate children of authority nodes are domain nodes. In contrast to the domain nodes in an attribute hierarchy, domain nodes in policies are inferentially qualified references (implicitly prepended with the label of their authority node parent) beginning with a top level domain and ending with the domain that contains the relevant attributes. Finally, leaf nodes represent attributes. In order to satisfy the policy such as the one shown in Figure 4.2, a user must present attribute keys that satisfy each node starting from the leaf nodes and proceeding up the policy to the root node. A leaf node is satisfied if the owner possesses a key for that attribute. Note that in Figure 4.2, since the domain is inferentially qualified, the attribute names have been abbreviated since their lineage in the hierarchy can also be inferred. The policy node named "Student" in Figure 4.2 would correspond to the attribute $\langle G_1, A_1$, University, Physics, Student$\rangle$ (and not the similarly named attribute $\langle G_1$, A_1, University, Electrical Engineering, Student$\rangle$ from Figure 4.1). In Figure 4.2, we see that the user only needs to satisfy one of the policy nodes "Student", "TA", or "Professor". This is an example of the *OR* operation, since possession of only a single attribute satisfied the threshold requirements. By meeting this threshold, the policy node labeled "1" is now satisfied. The domain node labeled "⟨University, Physics⟩" however requires both of its children to be satisfied. The right child "1" provides one satisfied child node. In order to meet the threshold of two, the user must also satisfy the node labeled "PHY220". The user must of course have a key corresponding to the attribute ⟨University, Physics, PHY220⟩ in order to satisfy this node. If this is the case, then the "Physics" node is satisfied. The

authority node "A_1" and the root node "G_1" are then satisfied in turn since each only has a threshold of one. Once the root node is satisfied, the policy corresponding to the tree is also considered satisfied. A formal description of the policy satisfaction algorithm $\mathcal{P}(\mathbb{A})$ can be found in [17].

In MA-AHASBE, the access policy is explicitly bundled in plaintext with the ciphertext. This is necessary because the structure of the ciphertext mirrors the structure found in the policy, so the decrypting party must have access to the policy in plaintext in order to determine which keys to apply to which locations. For many use cases, this is not a problem since the access policy itself is often not sensitive information. However, it may be the case that certain attributes are sensitive and should not be made public.

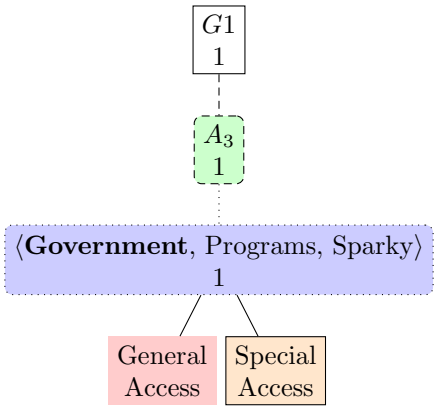


Figure 4.3: Example policy with a protected attribute

Consider the policy shown in Figure 4.3. This enforces a policy where the user must have either the "General Access" or "Special Access" attributes. Here, the knowledge that a "Special Access" attribute exists under the domain "Sparky" is considered sensitive. The attribute ⟨Government, Programs, Sparky, Special Access⟩ should not be allowed in plaintext policies. MA-AHASBE deals with this issue in one of two ways depending on the nature of the protected attribute. The first case applies when the protected attribute can

itself be protected under a policy which does not contain sensitive attributes. For example, perhaps any user with the attribute ⟨Corporation, Human Resources, Supervisor⟩, which is not sensitive, is allowed to know about the "Special Access" attribute for the Sparky program (even if they don't actually possess the "Special Access" attribute). This can be done by creating another ciphertext that encrypts the plaintext "⟨Government, Programs, Sparky, Special Access⟩" under a policy that does not have any protected nodes (in the example just given, this policy would require the "Supervisor" attribute). A reference can then be made to this ciphertext from the position of the protected node in the original policy. We call the referenced policy an *explicitly referenced protection policy*. During decryption, if the user satisfies the reference policy they can then determine the name of the original protected attribute, thereby recovering the original policy.

This technique for protecting attributes is not, for lack of better words, entirely satisfying. It is not difficult to imagine a scenario where a sensitive attribute cannot be protected through only attributes which are not sensitive. For example, perhaps only users who possess the "Special Access" attribute are allowed to know about the attribute "Special Access". To address this challenge, we fall back on the anonymity property of LW-AHIBE. For a formal definition of anonymity in LW-AHIBE, see [89]. Informally, anonymity in LW-AHIBE means that a ciphertext does not reveal any information about the identity used to create it, except potentially to a user who possesses the private key for the relevant identity. This latter caveat is necessary since a user in possession of the appropriate key could simply attempt to decrypt and determine if the result contained meaningful information. We leverage anonymity by initially proceeding in a manner similar to the method described above. However, instead of creating a regular ciphertext, a ciphertext is created based on a fixed, single attribute policy. This policy contains only the protected attribute, and the plaintext message contains some type of readily identifiable message such as a numeric index or a cryptographic hash of the attribute name. Note that for this reference

ciphertext, the plaintext used to create the ciphertext is not what is being protected, but rather the policy (and therefore the attribute) used to create it. Now, instead of explicitly including this fixed, single attribute policy with the ciphertext, it is not included at all. Rather it is implied through the fact that the only way to determine the attribute used in the policy (which recall are really identities in LW-AHIBE) is to perform a decryption using the appropriate attribute key. We call this type of policy an *implicitly referenced protection policy*.

Protecting attributes by using implicitly referenced protection policies creates an additional computational cost to the decrypting party. The scenario is not too different from having a key ring full of labeled keys and a number of unmarked locked doors. Without knowing which key goes to which door, the user has to try each key with each door until a match is found. If a match is found, we can now label the door according to the label of the key that unlocks it. In MA-AHASBE, the keys are the attributes the user possesses, and the doors are the protected attributes in a policy (protected through implicit protection policies). The time required to attempt to decrypt a single protected attribute with a single attribute key is O(1). The policy tree has a fixed size (it only contains one attribute) and a single attempt to decrypt can immediately be determined to be successful or not (with high probability) since the recovered plaintext should have some clear meaning. Each attempt also only uses a single attribute key (since only one key is necessary to satisfy the policy). If there are m protected attributes in a policy and the user possesses n keys, the total number of attempts is O($m * n$). This could still be impractical depending on the constants (e.g., time required per attempt, m, n), though we expect that in practical usage the number of protected attributes in a policy ($m < 10$) and number of total keys ($n < 100$) to be relatively small. It is important to note that this key discovery process is distinct from and occurs before the regular decryption algorithm. This technique is admittedly not as elegant as other attribute protection mechanisms such as those found in inner product type systems [69].

However, we feel that the other aspects of MA-AHASBE make it a better overall fit for the enterprise requirements discussed earlier and that the performance of protecting attributes this way is still within the limits of practicality. Note that this additional computation is not incurred when attributes are protected using explicitly referenced protection policies. In that case, the original policy is recovered (or determined to be irrecoverable) in a direct manner that does not depend on the number of keys or the number of protected attributes.

4.3.2.3 Key Structures, Attribute Sets and Multiple Domain Policies.

So far we have only discussed the most straightforward use of attributes in MA-AHASBE, single attributes from a single domain. MA-AHASBE also supports compound attributes in the form of attribute sets, and the capability to mix and match attributes from multiple domains and multiple authorities within the same policy. As the leaves of the policy inherit their design from LW-AHIBE, the methodology for combining policy nodes to satisfy a policy is inspired by CP-ASBE. The reader unfamiliar with CP-ASBE is encouraged to review [17] for a more detailed exposition of its merits. Essentially, CP-ASBE offers the capability to combine singleton attributes into compound attributes, while giving the encrypting party the choice of allowing such compound attributes to be split back apart in order to satisfy the policy. This becomes useful for representing complex attributes that naturally occur in access control policies, but are difficult to implement as singletons. Also it provides a rather elegant way of allowing numerical attributes with numerical comparisons in policies. Finally, it also allows for an efficient key revocation methodology.

Attribute sets work by combining attributes keys into recursive sets. These recursive sets are called *key structures*. A key structure is issued to a specific user by a domain. Most commonly, it only contains attributes from the issuing domain, however it may contain attributes from other domains within the same authority. Each user in the system is associated with a global identifier. This identifier allows multiple authorities and domains

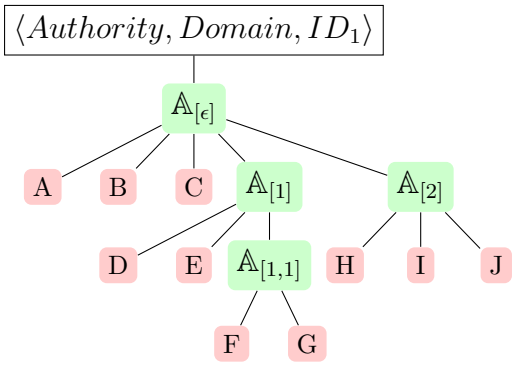


Figure 4.4: Example key structure

to issue keys to a single user. It also prevents users with different identifiers from colluding to satisfy policies they could not satisfy on their own. When a user possesses key structures from multiple domains, the set is called a *key ring*. An example key structure is shown in Figure 4.4. Here we see that it is a tree structure with three types of nodes: a root node, leaf nodes and set nodes. The root node indicates the authority and domain that generated the structure as well as the global identifier associated with the key structure. The child of the root node is a set node. Set nodes can have as children either leaf nodes or set nodes, while leaf nodes have no children. In addition the set nodes have been labeled in a particular way. The label $[\epsilon]$ represents an empty label and the root set node (the child of the root node) is given the empty label. Then the set nodes that share a parent are numbered starting with 1. This number is then appended to the label of the parent to form the label of the child. The leaf node children of a set node implicitly belong to a set whose label is made by appending 0 to the label of the parent set node. This special set is called an *outer set*. For example, in Figure 4.4 the leaf nodes {A, B, C} belong to an outer set $\mathbb{A}_{[0]}$ and {F, G} belong to $\mathbb{A}_{[1,1,0]}$. Finally, the depth of a set is determined by the number of elements in its label, ignoring any final zero entries. For example, both $\mathbb{A}_{[\epsilon]}$ and $\mathbb{A}_{[0]}$ have depth 0, $\mathbb{A}_{[1]}$ has depth 1, and $\mathbb{A}_{[1,1,0]}$ has depth 2.

By default, a policy in MA-AHASBE can only be satisfied with leaf node children of a particular set node. In other words, all the attributes used to satisfy a policy must belong to the same outer set. This is what enables single attributes to form compound attributes. For example, in Figure 4.4 the attributes *D* and *E* form a compound attribute. In this case, without any other conditions, a user with the key structure shown in Figure 4.4 could not satisfy a policy that required both *A* and *D* since they belong to different outer sets. On the other hand, the same user could satisfy a policy that only required *A* or only required *D*. The user could also satisfy a policy that required both *D* and *E*, since they both belong to the same outer set. In this way, outer sets like {A, B, C} or {H, I, J} form compound attributes that can only be used together and cannot by default be mixed and matched piecemeal with other compound attributes. In a sense this can be seen as a generalization of traditional attribute based encryption systems. In that case, all of the attributes in the system belong to the top most outer set (i.e., they form one large compound attribute).

One way to visualize the policy satisfaction algorithm is to imagine assigning the label of the outer set used to satisfy any particular leaf node to that leaf node. Threshold nodes can only combine children nodes that have been satisfied with the same label. While this certainly prevents mixing singleton attributes that belong to compound attributes, it also prevents multiple compound attributes from being used in the same policy. Figure 4.5(a) shows a policy that uses attributes from different sets. Assuming the key structure in Figure 4.4, the user can satisfy either the left branch or the right branch, but not both. This is because the label of the left node (in this case [0] since the attributes that satisfy the relevant leaf nodes come from the outer set $\mathbb{A}_{[0]}$) does not match the label of the right child node (which is [1,0] from the outer set $\mathbb{A}_{[1,0]}$). For the policy in Figure 4.5(a), it is not required to satisfy both. However, if the root node threshold was changed to 2 as in Figure 4.5(b) the policy cannot be satisfied with the key structure in Figure 4.4.

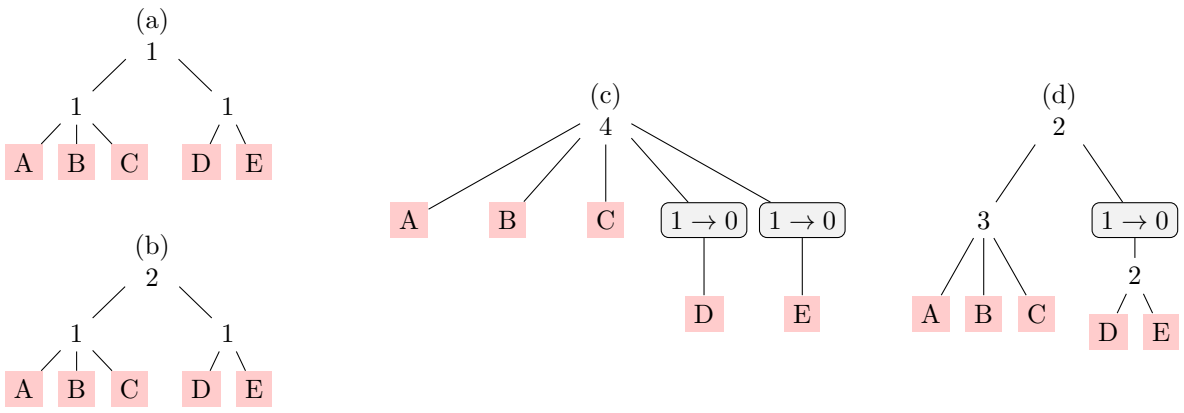


Figure 4.5: Example policies with translation nodes

Attribute sets also allow policies to control how compound attributes can be combined with each other. The mechanism for this control is called a translation node. Translation nodes allow users to translate attributes in their key structures from one depth to another. Figure 4.5(c) shows how a translation node might look in a policy. The translation nodes have been labeled "1 $\rightarrow$ 0". This means that if the child node of the translation node was satisfied using an outer set at depth 1, the translation node is considered to be satisfied by the label of the parent of that outer set which itself is at depth 0. For an example we use the policy Figure 4.5(c) and the key structure in Figure 4.4. Here, the leaf node "D" can only be satisfied by the outer set $\mathbb{A}_{[1,0]}$ which has depth 1. The translation node translates the label of this outer set to its parent which is $\mathbb{A}_{[0]}$. The translation node now can be combined with other nodes that have been satisfied with $\mathbb{A}_{[0]}$ such as "A", "B" and "C". The same process applies to "E". The policy in Figure 4.5(c) represents how a compound attribute can be split back into singletons and combined with other attributes. Figure 4.5(d) gives an example of using multiple compound attributes within the same policy, but not allowing the compound attributes to be split apart. Note that the threshold has to be met first, and then the result is translated. Unlike the policy in Figure 4.5(b), the policy in Figure 4.5(d) can be satisfied by the key structure in Figure 4.4.

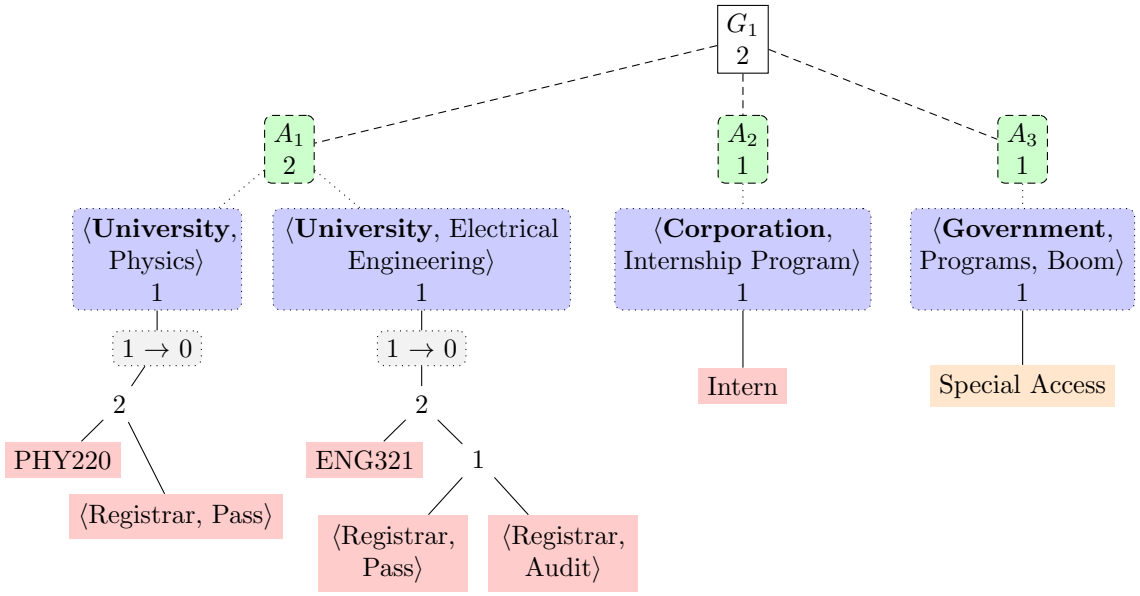


Figure 4.6: Example policy with multiple authorities and domains

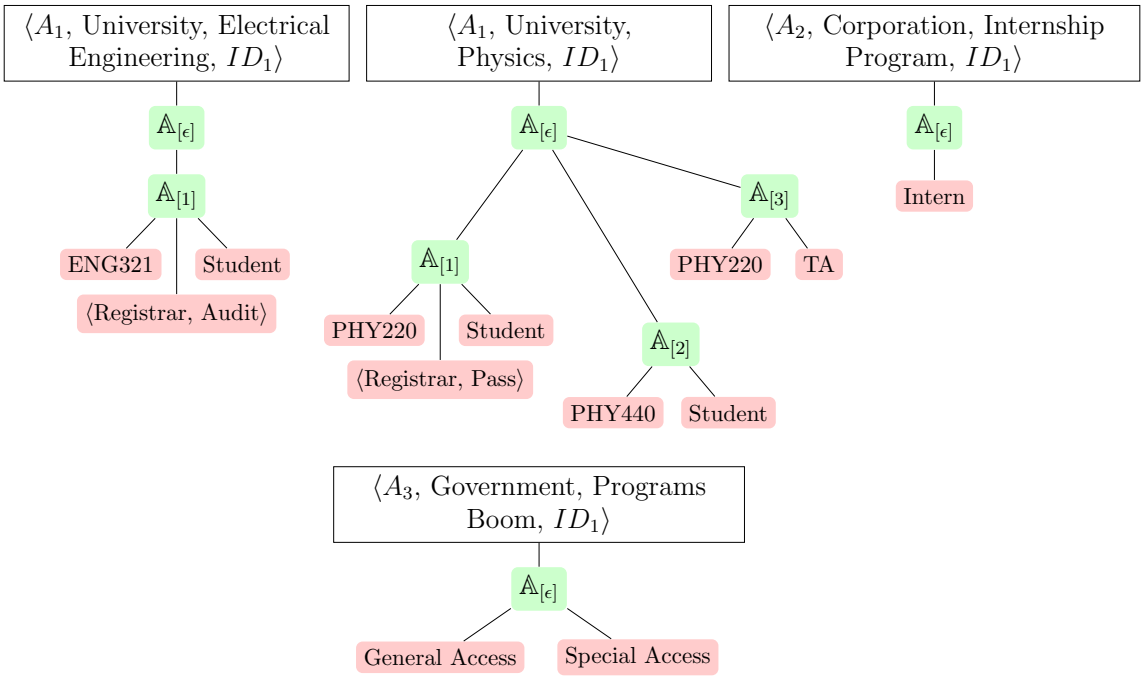


Figure 4.7: Example key ring

To see an example of how attribute sets can be utilized with multiple authorities and domains, we take an additional example from Figure 4.1. Say that we want to create a policy that restricts access to those interns in Corporation's Internship Program who have passed PHY220 and have either passed or audited ENG321. Such a policy is shown in Figure 4.6 and a key ring that will satisfy this policy is shown in Figure 4.7. The key ring shows that the attributes for the course have been combined into an attribute set. This makes it so that a policy can be created that prevents a user from applying a ⟨Pass⟩ attribute from one course to another. For the policy, we include an extra translation node beneath two of the domains. In a multiple domain policy, each domain node must have a label of depth 0 in order to combine with other domains. Since the ⟨Intern⟩ attribute is already at depth 0 in the key ring, it does not require a translation node.

4.3.3 Algorithm Summary.

The following algorithms define a multi-authority, hierarchical, attribute-set based encryption system:

GlobalSetup(λ, n_{user}) $\rightarrow GP$ The global setup algorithm takes in a security parameter λ and the number of users n_{user} the system must support. It outputs the public global parameters GP for the system.

AuthoritySetup(GP) $\rightarrow MSK_A$, AP_A Each authority runs the authority setup algorithm with the global parameters GP and produces a master secret key MSK_A and some public authority parameters AP_A for an authority A.

DomainKeyGeneration(GP, MSK_A, AP_A, $\mathcal{ID}_{domain} = \langle \mathsf{id}_1, ..., \mathsf{id}_\ell \rangle$, h') $\rightarrow DSK$ *or* $\perp$ An authority can create domains that may in turn create further subdomains or issue user secret keys (USK). The term h' determines how many levels of delegation are allowed by the key. Domain names are represented as a tuple of identities corresponding to a path in an attribute hierarchy where id_1 represents a top level domain.

Delegate(GP, DSK *or* USK, AP_A, $\mathcal{ID}_{domain}$ *or* $\mathcal{ID}_{attr} = \langle \mathsf{id}_1, ..., \mathsf{id}_\ell \rangle$, $\mathsf{id}_{\ell+1}$, h') $\rightarrow$ DSK *or* USK *or* $\perp$ When used with a DSK and $\mathcal{ID}_{domain}$ as input, this algorithm outputs the DSK for a subdomain. The term h' determines how many levels of delegation are allowed by the key. When used with a USK and $\mathcal{ID}_{attr}$ as input, the algorithm outputs another USK that represents an attribute that has been extended one level to a commitment value created during encryption.

UserKeyGeneration(GP, DSK, $\mathbb{A}$, $\mathcal{ID}_{user}$) $\rightarrow USK$ This takes as input a domain's secret key, a unique global identifier for a user, and the key structure for that user $\mathbb{A}$ (i.e., the attributes that have been assigned to them) and generates a user secret key.

Encrypt(GP, $\{AP_A, AP_B, ...\}$, $\mathcal{M}$) $\rightarrow \mathcal{CT}$ This encrypts a message $\mathcal{M}$ under an access policy $\mathcal{P}$ with attributes controlled by authorities $\{A, B, ...\}$.

Decrypt(GP, $\mathcal{CT}$, USK) $\rightarrow \mathcal{M}$ *or* $\perp$ Decrypts a ciphertext and produces a message if the decryption is successful. If decryption is not successful, it returns the special symbol $\perp$.

4.4 MA-AHASBE Construction

GlobalSetup(λ, n_{user}) $\rightarrow GP$ Let $\mathbb{G}_1$, $\mathbb{G}_2$ and $\mathbb{G}_T$ be groups of the same prime order p. Let $e : \mathbb{G}_1 \times \mathbb{G}_2 \rightarrow \mathbb{G}_T$ be an asymmetric bilinear map. Let $\mathcal{ID}_{user} = \langle \mathsf{id}_{user} \rangle$ where $\mathsf{id}_{user} \in \mathbb{Z}_p$. Choose random generators $P_1 \in \mathbb{G}_1$ and $F_2, Q_{id}, U_{id} \in \mathbb{G}_2$. Let $\mathcal{H}_{id}(\mathcal{ID}_{user}) = \mathsf{id}_{user} Q_{id} + U_{id}$. The parameter n_{user} is the total number of users in the system that require a globally unique identifier. Let $\{\mathcal{ID}_{user:j}\}_{j \in [1, n_{user}]}$ be the set of all user identifiers in the system. Note that Q_{id} and U_{id} are fixed and that the number of elements in $\{\mathcal{ID}_{user:j}\}$ is n_{user}. Let r_{max} denote the maximum recursion depth for the CP-ASBE. Choose $\alpha_g, \beta_{id}, \{\beta_i\}_{i \in [0, r_{max}]} \xleftarrow{U} \mathbb{Z}_p^*$. If the global authority needs to generate additional identifiers after setup, then it must retain the global secret key GSK. Let

$\mathcal{H} : \{0,1\}^* \to \{0,1\}^\lambda$ be a cryptographic hash function. Let $\mathsf{h} : \{0,1\}^* \to \{0,1\}^\lambda$ be a hash function from a family of pairwise independent hash functions. If $\lambda = 128$, then per the analysis in [19] the size of the input space of h should be 448 bits.

GP: $(p, e, \mathcal{H}, \mathsf{h}, P_1, F_2, \beta_{id}P_1, \frac{-\alpha_g}{\beta_0}F_2, \{\beta_i P_1, \frac{1}{\beta_i}F_2\}_{i\in[0,r_{max}]}, e(P_1,F_2)^{\alpha_g}$
$\{\mathcal{ID}_{user:j} \xleftarrow{U} \mathbb{Z}_p, \mathcal{H}_{id}(\mathcal{ID}_{user:j}), \frac{1}{\beta_0}\mathcal{H}_{id}(\mathcal{ID}_{user:j}), \frac{-\alpha_g}{\beta_0}\mathcal{H}_{id}(\mathcal{ID}_{user:j}), \frac{\alpha_g}{\beta_{id}}\mathcal{H}_{id}(\mathcal{ID}_{user:j})\}_{j\in[1,n_{user}]})$
GSK: $(\frac{1}{\beta_0}, \frac{-\alpha_g}{\beta_0}, \frac{\alpha_g}{\beta_{id}}, Q_{id}, U_{id})$

AuthoritySetup$(GP) \to MSK_A, AP_A$ Let h_{max} denote the maximum depth of the delegation hierarchy. Choose a random generator $P_2 \in \mathbb{G}_2$; elements $\{Q_{1,j}\}_{j\in[1,h_{max}]}, U_1 \xleftarrow{U} \mathbb{G}_1$ and$\{Q_{2,j}\}_{j\in[1,h_{max}]}, U_2 \xleftarrow{U} \mathbb{G}_2$ such that $(Q_{1,j} \sim Q_{2,j})_{j\in[1,h_{max}]}$ and $U_1 \sim U_2$. Choose α_A, a, v, $v' \xleftarrow{U} \mathbb{Z}_p^*$. Set $V_2 = vF_2$, $V_2' = v'F_2$ and $\tau = v + av'$ so that $\tau F_2 = V_2 + aV_2'$.

AP_A: $(aP_1, \tau P_1, U_1, aU_1, \tau U_1, \{Q_{1,j}, aQ_{1,j}, \tau Q_{1,j}\}_{j\in[1,h_{max}]}, V_2, V_2', e(P_1,P_2)^{\alpha_A})$
MSK_A: $(\alpha_A P_2, \{Q_{2,j}\}_{j\in[1,h_{max}]}, U_2)$

DomainKeyGeneration$(GP, MSK_A, AP_A, \mathcal{ID}_{domain} = \langle \mathsf{id}_1, ..., \mathsf{id}_\ell \rangle, h') \to DSK \text{ or } \perp$ If $h' \notin [\ell + 2, h_{max} - 2]$, then return $\perp$. Otherwise, choose w_1, w_2, r_1, r_2, r_3, r_4, $(z_{1,j}, z_{2,j})_{j\in[\ell+1,h']} \xleftarrow{U} \mathbb{Z}_p^*$. Let $\mathcal{H}_2(\mathcal{ID}_{domain}) = (\sum_{j=1}^{\ell} \mathsf{id}_j Q_{2,j}) + U_2$.

DSK_{domain}:

$K_{1,1} = w_1P_2 + r_1V_2$, $K_{1,2} = r_1V_2'$, $K_{1,3} = r_1F_2$
$K_{2,1} = \alpha_A P_2 + w_1\mathcal{H}_2(\mathcal{ID}_{domain}) + r_2V_2$, $K_{2,2} = r_2V_2'$, $K_{2,3} = r_2F_2$
$J_{1,1} = w_2P_2 + r_3V_2$, $J_{1,2} = r_3V_2'$, $J_{1,3} = r_3F_2$
$J_{2,1} = w_2\mathcal{H}_2(\mathcal{ID}_{domain}) + r_4V_2$, $J_{2,2} = r_4V_2'$, $J_{2,3} = r_4F_2$
For $j \in [\ell + 1, h']$
$D_{j,1} = w_1Q_{2,j} + z_{1,j}V_2$, $D_{j,2} = z_{1,j}V_2'$, $D_{j,3} = z_{1,j}F_2$
$E_{j,1} = w_2Q_{2,j} + z_{2,j}V_2$, $E_{j,2} = z_{2,j}V_2'$, $E_{j,3} = z_{2,j}F_2$

Delegate(GP, DSK_{domain} *or* USK, AP_A, $\mathcal{ID}_{domain}$ *or* $\mathcal{ID}_{attr} = \langle \mathsf{id}_1, ..., \mathsf{id}_\ell \rangle$, $\mathsf{id}_{\ell+1}$, h') $\rightarrow$ DSK *or* USK *or* $\perp$ If $h' \notin [\ell+1, h_{max}]$, then return $\perp$. Otherwise, choose w'_1, w'_2, r'_1, r'_2, r'_3, r'_4, $(z'_{1,j}, z'_{2,j})_{j\in[\ell+2,h']} \xleftarrow{U} \mathbb{Z}_p^*$. Note that the algorithm performs the same operations for both DSK and USK inputs. For DSK_{domain} input, the output is $DSK_{subdomain}$. For USK input, the output is $USK_{\mathsf{id}_{\ell+1}}$.

$DSK_{subdomain}$ *or* $USK_{\mathsf{id}_{\ell+1}}$:

$K_{1,1} \leftarrow K_{1,1} + w'_1 J_{1,1} + r'_1 V_2$ $\quad$ $K_{2,1} \leftarrow K_{2,1} + \mathsf{id}_{\ell+1} D_{\ell+1,1} + w'_1 (J_{2,1} + \mathsf{id}_{\ell+1} E_{\ell+1,1}) + r'_2 V_2$

$K_{1,2} \leftarrow K_{1,2} + w'_1 J_{1,2} + r'_1 V'_2$ $\quad$ $K_{2,2} \leftarrow K_{2,2} + \mathsf{id}_{\ell+1} D_{\ell+1,2} + w'_1 (J_{2,2} + \mathsf{id}_{\ell+1} E_{\ell+1,2}) + r'_2 V'_2$

$K_{1,3} \leftarrow K_{1,3} + w'_1 J_{1,3} + r'_1 F_2$ $\quad$ $K_{2,3} \leftarrow K_{2,3} + \mathsf{id}_{\ell+1} D_{\ell+1,3} + w'_1 (J_{2,3} + \mathsf{id}_{\ell+1} E_{\ell+1,3}) + r'_2 F_2$

$J_{1,1} \leftarrow w'_2 J_{1,1} + r'_3 V_2$ $\quad$ $J_{2,1} \leftarrow w'_2 (J_{2,1} + \mathsf{id}_{\ell+1} E_{\ell+1,1}) + r'_4 V_2$

$J_{1,2} \leftarrow w'_2 J_{1,2} + r'_3 V'_2$ $\quad$ $J_{2,2} \leftarrow w'_2 (J_{2,2} + \mathsf{id}_{\ell+1} E_{\ell+1,2}) + r'_4 V'_2$

$J_{1,3} \leftarrow w'_2 J_{1,3} + r'_3 F_2$ $\quad$ $J_{2,3} \leftarrow w'_2 (J_{2,3} + \mathsf{id}_{\ell+1} E_{\ell+1,3}) + r'_4 F_2$

For $j \in [\ell+2, h']$

$D_{j,1} \leftarrow D_{j,1} + w'_1 E_{j,1} + z'_{1,j} V_2$ $\quad$ $D_{j,2} \leftarrow D_{j,2} + w'_1 E_{j,2} + z'_{1,j} V'_2$ $\quad$ $D_{j,3} \leftarrow D_{j,3} + w'_1 E_{j,3} + z'_{1,j} F_2$

$E_{j,1} \leftarrow w'_2 E_{j,1} + z'_{2,j} V_2$ $\quad$ $E_{j,2} \leftarrow w'_2 E_{j,2} + z'_{2,j} V'_2$ $\quad$ $E_{j,3} \leftarrow w'_2 E_{j,3} + z'_{1,j} F_2$

UserKeyGeneration(GP, DSK, $\mathbb{A}$, $\mathcal{ID}_{user}$) $\rightarrow$ USK Here, $\mathbb{A}$ is the key structure issued by a domain with $\mathcal{ID}_{domain} = \langle \mathsf{id}_1, ..., \mathsf{id}_\ell \rangle$ for a user with a global identifier $\mathcal{ID}_{user} = \langle \mathsf{id}_{user} \rangle$. A key structure contains a number of recursive sets, each set associated with a recursive depth. A set in $\mathbb{A}$ is labeled with an integer tuple $\mathbf{i}$, where the number of elements in $\mathbf{i}$ corresponds to the recursive depth of the set and each element of $\mathbf{i}$ comes from the set $[0, \infty)$. The tuple $\mathbf{i}$ is called the *label* for the set $\mathbb{A}_{\mathbf{i}}$. The notation $\mathbb{A}_{authority,domain,user,\mathbf{i}}$ fully qualifies a recursive set in the system, though we generally omit all but the tuple portion for convenience and use the simpler notation $\mathbb{A}_{\mathbf{i}}$ when the context is clear. The notation $\mathbb{A}_{\mathbf{i}'\|j}$ represents the set $\mathbb{A}_{\mathbf{i}}$ where $\mathbf{i}$ is the concatenation of $\mathbf{i}'$ and the integer $j \in [0, \infty)$.

At each recursive depth d, there is at least one (possibly empty) set $\mathbb{A}_{\mathbf{i}'\|0}$ where d is the number of elements in $\mathbf{i}'$. This set is referred to as the *outer set* at depth d. The outer set $\mathbb{A}_{\mathbf{i}'\|0}$ contains $n_k \geq 0$ number of attributes such that $\mathbb{A}_{\mathbf{i}'\|0} = \{\mathsf{attr}_{\mathbf{i},k}\}_{k\in[1,n_k]}$. Note that $\mathbb{A}_{[0]}$ is the outer set at depth 0, and therefore the outer set for all of $\mathbb{A}_{authority,domain,user}$. An outer set can only contain attributes and cannot contain other sets (i.e., the outer set itself is not recursive). This means there are no labels where the last two elements are both 0.

To generate a user key, $\forall \mathbb{A}_{\mathbf{i}} \in \mathbb{A}_{authority,domain,user}$, $\forall \mathsf{attr} \in \mathbb{A}_{\mathbf{i}}$ the algorithm calls **Delegate**$(DSK,\ AP_A,\ \mathcal{ID}_{domain},\ \mathsf{id}_{\ell+1} = \mathsf{attr},\ \ell+2)$ and stores the result in $\mathbb{K}'_{\mathbf{i},\mathsf{attr}} = \{K'_{\gamma,\delta}, J'_{\gamma,\delta}, D'_{\ell+2,\delta}, E'_{\ell+2,\delta}\}_{\gamma\in[1,2],\delta\in[1,3]}$. Then the algorithm chooses $\{r_{\mathbf{i}} \xleftarrow{U} \mathbb{Z}_p^*\}_{\forall \mathbb{A}_{\mathbf{i}}\in\mathbb{A}_{authority,domain,user}}$ with the restriction that $\forall r_{\mathbf{i}'\|j}$ such that $j \neq 0$ that $r_{\mathbf{i}'\|j} = r_{\mathbf{i}'\|j\|0}$. The algorithm then runs the **RestrictToUser**$(\mathbb{K}'_{\mathbf{i},\mathsf{attr}}, \mathcal{ID}_{user}, r_{\mathbf{i}})$ algorithm described below and stores the result in $\mathbb{K}_{\mathbf{i},\mathsf{attr},user}$. Let $\mathbb{K}_{\mathbf{i},user} = \{\mathbb{K}_{\mathbf{i},\mathsf{attr},user}\}_{\forall\mathsf{attr}\in\mathbb{A}_{\mathbf{i}}}$.

RestrictToUser$(\mathbb{K}'_{\mathbf{i},\mathsf{attr}}, \mathcal{ID}_{user}, r_{\mathbf{i}}) \rightarrow \mathbb{K}_{\mathbf{i},\mathsf{attr},user}$

$$\forall K'_{\gamma,\delta} \in \mathbb{K}'_{\mathbf{i},\mathsf{attr}} : K_{\gamma,\delta} \leftarrow \begin{cases} K'_{\gamma,\delta} & \text{if } \gamma \neq 2, \delta \neq 1 \\ K'_{2,1} + r_{\mathbf{i}} F_2 + r_{[0]}\mathcal{H}_{id}(\mathcal{ID}_{user}) & \text{if } \gamma = 2, \delta = 1 \end{cases}$$

$$\forall J'_{\gamma,\delta} \in \mathbb{K}'_{\mathbf{i},\mathsf{attr}} : J_{\gamma,\delta} \leftarrow J'_{\gamma,\delta}$$

$$\forall D'_{\ell+2,\delta} \in \mathbb{K}'_{\mathbf{i},\mathsf{attr}} : D_{\ell+2,\delta} \leftarrow D'_{\ell+2,\delta}$$

$$\forall E'_{\ell+2,\delta} \in \mathbb{K}'_{\mathbf{i},\mathsf{attr}} : E_{\ell+2,\delta} \leftarrow E'_{\ell+2,\delta}$$

Essentially, **RestrictToUser** appends two additional components onto the $K_{2,1}$ portion of the key in the input set. These two elements are critical in that they create additional blinding factors during decryption that provide certain security protections. How these elements work will be described in more detail in the decryption algorithm. The value $r_{\mathbf{i}}$ acts as a label for the attribute set. The method described here requires that the attributes being combined into a set originate from the same domain. This can be extended to

other domains within the same authority (but not domains from different authorities) if domains are willing to share these labels. The details of this extension are not described in this chapter. Finally, the user key USK is defined as follows:

USK:

$\forall \mathbb{A}_{\mathbf{i}} \in \mathbb{A}_{authority,domain,user}$: $(\mathbb{A}_{\mathbf{i}}, \mathbb{K}_{\mathbf{i},user})$

$K_{id} = \frac{\alpha_g}{\beta_{id}} \mathcal{H}_{id}(\mathcal{ID}_{user})$

$K_{F_2,[0]} = \frac{-\alpha_g + r_{[0]}}{\beta_0} F_2$, $K_{id,[0]} = \frac{-\alpha_g + r_{[0]}}{\beta_0} \mathcal{H}_{id}(\mathcal{ID}_{user})$

$\forall r_{\mathbf{i}||j}$ such that $j \neq 0$, let $d = depth(\mathbf{i} \,||\, j)$ where $d \geq 1$

$K_{F_2,\mathbf{i}||j} = \frac{-r_{\mathbf{i}||0} + r_{\mathbf{i}||j}}{\beta_d} F_2$

Encrypt$(GP, \{AP_A, AP_B, ...\}, \mathcal{M}) \rightarrow \mathcal{CT}$ The general mechanism for generating each node is very similar to the CP-ASBE system. The primary difference is that the generated ciphertext reflects the multi-authority and AHIBE enhancements to the underlying systems. To support multiple authorities, we introduce the notion of global nodes. Global nodes can be either global threshold or global leaf nodes. A global leaf node marks the root of a subtree where all the nodes in the subtree belong to a single authority. Global leaf nodes also act as threshold nodes, and therefore have an associated threshold value. Global threshold nodes work in the same way as local threshold nodes, except that the children of a global threshold node must be global (either leaf or threshold) and the parent must be a global threshold node (or no parent in the case of the global root node). Also, we introduce the notion of a domain threshold node. When a policy contains leaf nodes from different domains within the same authority, there must be a threshold node such that all nodes in the associated subtree all come from the same domain. Since a global leaf acts as threshold node, a global leaf node can fulfill the role of the domain threshold node in the case where the policy only uses a single domain from the relevant authority. Otherwise, an additional local threshold node must be in the policy to act as

the domain threshold node. When this additional node is required, then it is always the child of a global leaf node. The children of a domain threshold node are either local leaf or local threshold nodes.

The encryption begins in a top down manner, starting with a global root node. At each node, a polynomial q_τ is generated for all nodes (global and local) and a similar polynomial g_τ is generated for all global nodes. For each threshold node τ in the tree, the degree d_τ of both polynomials q_τ and g_τ are set to be one less than the threshold value k_τ (i.e. $d_\tau = k_\tau - 1$). The degree of a local leaf node is 0. For the global root node R, the algorithm chooses a random $s \xleftarrow{U} \mathbb{Z}_p^*$ and sets $q_R(0) = s$ and $g_R(0) = 1$. From this point onward, all calculations for both q_τ and g_τ are performed in the exact same manner. The only difference would be that the root values for q and g are different (s and 1 respectively). The calculations are explained in terms of q. The algorithm chooses d_R other points randomly to define the polynomial q_R. For any other threshold node τ, it sets $q_\tau(0) = q_{parent(\tau)}(index(\tau))$ and chooses d_τ other points randomly to completely define q_τ. Here **parent**(τ) represents the parent node of τ. Let $\mathbb{W}$ represent the global leaf nodes of the access tree. $\mathbb{Y}$ represents the local leaf nodes, and $\mathbb{X}$ represents local translation nodes. $\mathbb{T}_\tau$ represents the translation indices that are allowed at a translation node. Their use will be explained in the decryption algorithm. Let $\mathbb{V}$ represent the set of domain threshold nodes.

Let $\mathbb{A}_p$ indicate the set of attributes that the encryptor wishes to keep anonymous and let n_p represent the number of attributes in $\mathbb{A}_p$. The function *pindex*(attr) returns a unique index from $[1, n_p]$ for each attribute in $\mathbb{A}_p$. The algorithm **AttributeEncrypt** is the same algorithm as **Encrypt** with a fixed access policy of a single attribute. Also, for **AttributeEncrypt** the access policy is not included in the resulting ciphertext.

Let $|\mathbb{G}_T|_\ell$ be the number of bits required to represent an element of $\mathbb{G}_T$. Let $m \leftarrow \{0,1\}^{m_\ell}$ be the input message. Choose $x \xleftarrow{U} \{0,1\}^{x_\ell}$ such that $m_\ell + x_\ell \leq |\mathbb{G}_T|_\ell$. If the

message is a symmetric key (or the seed of a pseudorandom number generator), the recommended values [19] for 128-bit security are $m_\ell = 128, x_\ell = 448$. Let $k' \xleftarrow{U} \mathbb{G}_T$. Let $[\{0,1\}^*]^n$ represent the first n bits of the underlying bit string. The $\|$ symbol indicates concatenation. The attribute $\mathsf{attr}_{\mathbf{com}}$ represents the identity tuple $\langle \mathcal{ID}_{domain}, \mathsf{id}_{\ell+1} = \mathsf{attr}, \mathsf{id}_{\ell+2} = \mathcal{H}(x)\rangle$. Let $\mathcal{H}_1(\mathcal{ID} = \langle \mathsf{id}_1,...,\mathsf{id}_\ell\rangle) = (\sum_{j=1}^{\ell} \mathsf{id}_j Q_{1,j}) + U_1$. Now we can define the ciphertext $\mathcal{CT} = (CT_{\mathbf{com}}, CT, CT_{\mathbf{tag}})$ as follows:

CT:

$\mathcal{P}, C_{id} = s\beta_{id}P_1, C_\sigma = [k']^{m_\ell+x_\ell} \oplus (m \,\|\, x)$

$\forall v \in \mathbb{V} : C_v = q_v(0)\beta_0 P_1$

$\forall x \in \mathbb{X}, \forall d \in \mathbb{T}_x : C'_{x,d} = q_x(0)\beta_d P_1$

$\forall w \in \mathbb{W} : C_w = (k' \cdot e(P_1, F_2)^{\alpha_g s})^{g_w(0)} e(P_1, P_2)^{\alpha_A q_w(0)}$

$\forall y \in \mathbb{Y} : C_{y,1,1} = q_y(0)\mathcal{H}_1(\mathsf{attr}_{\mathbf{com}}),\ C_{y,1,2} = aq_y(0)\mathcal{H}_1(\mathsf{attr}_{\mathbf{com}}),\ C_{y,1,3} = -\tau q_y(0)\mathcal{H}_1(\mathsf{attr}_{\mathbf{com}})$

$C_{y,2,1} = q_y(0)P_1,\ C_{y,1,2} = aq_y(0)P_1,\ C_{y,1,3} = -\tau q_y(0)P_1$

$$\mathcal{P}_y = \begin{cases} \mathsf{attr} & \text{if } \mathsf{attr} \notin \mathbb{A}_p \\ pindex(\mathsf{attr}) & \text{if } \mathsf{attr} \in \mathbb{A}_p \end{cases}$$

$\forall \mathsf{attr} \in \mathbb{A}_p : \mathcal{P}_{pindex(\mathsf{attr})} = \mathbf{AttributeEncrypt}(GP, AP_A, \mathcal{M} = pindex(\mathsf{attr}), \mathcal{P} = \mathsf{attr}))$

$CT_{\mathbf{com}} = \mathcal{H}(x), CT_{\mathbf{tag}} = \mathsf{MAC}_k(CT)$ where $k = \mathsf{h}(x)$

Decrypt$(GP, \mathcal{CT}, USK) \rightarrow \mathcal{M}$ *or* $\perp$ The decryption process again follows a very similar path as the decryption process for CP-ASBE. First, the policy satisfaction algorithm $\mathcal{P}(\mathbb{A})$ is run to determine if the policy associated with the ciphertext is satisfied by the key structure provided in the user key. If the algorithm succeeds, it returns a set of labels on the root node that can be used to satisfy the policy. Otherwise the algorithm returns the empty set and the ciphertext cannot be decrypted. If the policy satisfaction algorithm succeeds, the decryption algorithm picks one of the labels **i** from the set returned by $\mathcal{P}(\mathbb{A})$ and calls a recursive function **DecryptNode**$(CT, USK, t, \mathbf{i})$ on the root node of the tree.

If $t \in \mathbb{Y}$ (i.e., t is a local leaf node), then **DecryptNode** is defined as follows. If $att(t) \neq pindex(\mathsf{attr})$ and $att(t) \notin A_{\mathbf{i}}$ where $A_{\mathbf{i}} \in \mathbb{A}_{domain,user}$ then return $\perp$. If $att(t) = pindex(\mathsf{attr})$, then the attribute is protected and must be recovered. This is done by iteratively attempting to decrypt $\mathcal{P}_{pindex(\mathsf{attr})}$ until a key is found that results in $\mathcal{M} = pindex(\mathsf{attr})$. This is an $O(n_p m)$ operation where n_p is the number of protected attributes in the policy and m is the number of keys the user possesses. Once the appropriate key is found, it is used for the remaining decryption steps. If the attribute is not protected then $att(t) = \mathsf{attr} \in A_{\mathbf{i}}$ where $A_{\mathbf{i}} \in \mathbb{A}_{domain,user}$ and the decryption key is the associated $\mathbb{K}_{\mathbf{i},user}$. Let $\mathcal{ID}_{attr} = \langle \mathsf{id}_1, ..., \mathsf{id}_\ell, \mathsf{id}_{\ell+1} = \mathsf{attr} \rangle$. Compute the decryption key set $\mathbb{K}_{\mathbf{i},\mathsf{attr}_{\mathbf{com}},user}$ = **Delegate**$(USK = \mathbb{K}_{\mathbf{i},\mathsf{attr},user}, AP_A, \mathcal{ID}_{attr}, \mathsf{id}_{\ell+2} = CT_{\mathbf{com}}, \ell + 2)$. Using the ciphertext at node t and $\mathbb{K}_{\mathbf{i},\mathsf{attr}_{\mathbf{com}},user}$ node decryption is defined as follows:

DecryptNode$(CT, USK, t, \mathbf{i})$

$$= \frac{e(C_{t,1,1}, K_{1,1})e(C_{t,1,2}, K_{1,2})e(C_{t,1,3}, K_{1,3})}{e(C_{t,2,1}, K_{2,1})e(C_{t,2,2}, K_{2,2})e(C_{t,2,3}, K_{2,3})}$$

$$= \frac{e(q_t(0)\mathcal{H}_1(\mathsf{attr}_{\mathbf{com}}), w_1 P_2 + r_1 V_2)e(aq_t(0)\mathcal{H}_1(\mathsf{attr}_{\mathbf{com}}), r_1 V_2')e(-\tau q_t(0)\mathcal{H}_1(\mathsf{attr}_{\mathbf{com}}), r_1 F_2)}{e(q_t(0)P_1, \alpha P_2 + w_1\mathcal{H}_2(\mathsf{attr}_{\mathbf{com}}) + r_2 V_2 + r_{\mathbf{i}} F_2 + r_{[0]}\mathcal{H}_{id}(\mathcal{ID}_{user}))e(aq_t(0)P_1, r_2 V_2')e(-\tau q_t(0)P_1, r_2 F_2)}$$

$$= \frac{e(\mathcal{H}_1(\mathsf{attr}_{\mathbf{com}}), P_2)^{q_t(0)w_1} e(\mathcal{H}_1(\mathsf{attr}_{\mathbf{com}}), V_2)^{q_t(0)r_1} e(\mathcal{H}_1(\mathsf{attr}_{\mathbf{com}}), V_2')^{aq_t(0)r_1} e(\mathcal{H}_1(\mathsf{attr}_{\mathbf{com}}), F_2)^{-\tau q_t(0)r_1}}{e(P_1, P_2)^{\alpha q_t(0)} e(P_1, \mathcal{H}_2(\mathsf{attr}_{\mathbf{com}}))^{q_t(0)w_1} e(P_1, V_2)^{q_t(0)r_2} e(P_1, F_2)^{q_t(0)r_{\mathbf{i}}} e(P_1, \mathcal{H}_{id}(\mathcal{ID}_{user}))^{q_t(0)r_{[0]}} e(P_1, V_2')^{aq_t(0)r_2} e(P_1, F_2)^{-\tau q_t(0)r_2}}$$

$$= \frac{1}{e(P_1, P_2)^{\alpha q_t(0)} e(P_1, F_2)^{r_{\mathbf{i}} q_t(0)} e(P_1, \mathcal{H}_{id}(\mathcal{ID}_{user}))^{r_{[0]} q_t(0)}}$$

At this point, the similarity to the original CP-ASBE system becomes more apparent. The first term in the denominator will combine with the decryption of other nodes within the same authority subtree in the access policy. As long as all thresholds are met, the polynomial interpolation will eventually produce the term $e(P_1, P_2)^{\alpha q_w(0)}$ where w is the

global leaf node that sits at the root of this authority subtree. This term in the denominator will clear the original blinding factor imposed during encryption on k' at node w. However, the last two terms produced in the denominator by the **DecryptNode** process introduce new blinding factors. These terms themselves are blinded by the attribute set values $r_{\mathbf{i}}$, $r_{[0]}$ as well as the polynomial term q_t.

In order to enable the full expressive power of attribute sets, we must utilize the translation parameters included in the ciphertext in a manner similar to the original CP-ASBE system [17]. When $t \notin \mathbb{Y}$ but it is still a local node (i.e., t is a local threshold node), then **DecryptNode** runs as follows:

1. Compute $\mathbb{B}_t$ which contains a subset of any k_t (i.e., the threshold at node M) child nodes z such that $z \in \mathbb{B}_t$ only if either: label $\mathbf{i} = (\mathbf{i}' \parallel j) \in S_z$, or label $(\mathbf{i}' \parallel j') \in S_z$ for some $j' \neq j$ and z is a translating node with $depth(\mathbf{i}) \in \mathbb{T}_z$.

2. For each node $z \in \mathbb{B}_t$ such that label $\mathbf{i} = (\mathbf{i}' \parallel j) \in S_z$ call **DecryptNode**$(CT, USK, t, \mathbf{i})$ and store the output in F_z.

3. For each node $z \in \mathbb{B}_t$ such that the label $(\mathbf{i}' \parallel j') \in S_z$ and $j' \neq j$ call **DecryptNode**$(CT, USK, t, (\mathbf{i}' \parallel j'))$ and store the output in F'_z. If $j = 0$ then $r_{\mathbf{i}} = r_{\mathbf{i}'\|0} = r_{\mathbf{i}'}$ (recall the restriction in the user key generation that $r_{\mathbf{i}} = r_{\mathbf{i}\|0}$) and $depth(\mathbf{i}' \parallel j') = d$. Translate F'_z to F_z as follows:

$$F_z = e(C'_{z,d}, K_{F2,\mathbf{i}'\|j'})F'_z = e(q_z(0)\beta_d P_1, (\frac{-r_{\mathbf{i}'\|0}+r_{\mathbf{i}'\|j'}}{\beta_d}F_2))F'_z = e(P_1, F_2)^{q_z(0)(-r_{\mathbf{i}}+r_{\mathbf{i}\|j'})}F'_z$$

$$= \frac{1}{e(P_1, P_2)^{\alpha q_z(0)} e(P_1, F_2)^{r_{\mathbf{i}} q_z(0)} e(P_1, \mathcal{H}_{id}(\mathcal{ID}_{user}))^{r_{[0]} q_z(0)}}$$

If $j \neq 0$ then translate F'_z to F_z as follows:

$$F_z = e(C'_{z,d}, K_{F2,\mathbf{i}'\|j'} - K_{F2,\mathbf{i}'\|j})F'_z = e(q_z(0)\beta_d P_1, \frac{r_{\mathbf{i}'\|j'} - r_{\mathbf{i}'\|j}}{\beta_d} F_2)F'_z$$

$$= \frac{1}{e(P_1, P_2)^{\alpha q_z(0)} e(P_1, F_2)^{r_{\mathbf{i}} q_z(0)} e(P_1, \mathcal{H}_{id}(\mathcal{ID}_{user}))^{r_{[0]} q_z(0)}}$$

Note that the above translations are between sets whose recursive depth differs by at most one. More specifically, the translation is from the outer set of $\mathbb{A}_{\mathbf{i}'\|j'}$ to the outer set of $\mathbb{A}_{\mathbf{i}'\|j}$ if $j \neq 0$ or the outer set of $\mathbb{A}_{\mathbf{i}}$ if $j = 0$. Deeper translations can be made in a similar manner as demonstrated above provided the relevant translation indices exist in $\mathbb{T}_z$. Recall that these translation indices are created during encryption and therefore are part of the access policy. Hence, they are available only at the discretion of the encrypting party.

4. Compute F_t using polynomial interpolation in the exponent as follows:

   $F_t = \prod_{z \in \mathbb{B}_t} F_z^{\Delta_{k,B'_z}(0)}$ where $k = index(z), B'_z = \{index(z) : z \in \mathbb{B}_t\}$

   and the Lagrange coefficient $\Delta_{i,S}(x) = \prod_{j \in S, j \neq i} \frac{x-j}{i-j}$

   $$= \frac{1}{e(P_1, P_2)^{\alpha q_t(0)} e(P_1, F_2)^{r_{\mathbf{i}} q_t(0)} e(P_1, \mathcal{H}_{id}(\mathcal{ID}_{user}))^{r_{[0]} q_t(0)}}$$

   The computations above work up the access policy until the node t is a domain threshold node. When this occurs, the algorithm performs all the steps for a local threshold node with following additional step:

5. Using the methodology established in Step 3 above, translate from the set $\mathbb{A}_{\mathbf{i}}$ to the outer set of $\mathbb{A}_{authority,domain,user}$ using the translation parameters $C'_{w,d}$ along with the translation components of the user key $K_{F_2,\mathbf{i}}$. If the access policy is satisfied by $\mathbb{A}_{authority,domain,user}$, it will then be possible to use the pairing of C_v with $K_{F_2,[0]}$ and $K_{id,[0]}$ to perform a final translation. This changes the blinding factor to α_g which is

shared amongst all nodes. Recall the parent of a domain threshold node is a global leaf node in the multi-domain case, or itself in the single domain case. Let $w \in \mathbb{W}$ be the relevant global leaf node. Once enough domain threshold nodes are decrypted to satisfy the threshold of w, store the result of the interpolation (as outlined in Step 4) of the decrypted domain threshold node values in F'_w. F_w is calculated as follows:

$$F_w = C_w F'_w = (k' \cdot e(P_1, F_2)^{\alpha_g s})^{g_w(0)} e(P_1, P_2)^{\alpha q_w(0)} F'_w$$

$$= \frac{(k' \cdot e(P_1, F_2)^{\alpha_g s})^{g_w(0)} e(P_1, P_2)^{\alpha q_w(0)}}{e(P_1, P_2)^{\alpha q_w(0)} e(P_1, F_2)^{\alpha_g q_w(0)} e(P_1, \mathcal{H}_{id}(\mathcal{ID}_{user}))^{\alpha_g q_w(0)}}$$

$$= \frac{(k' \cdot e(P_1, F_2)^{\alpha_g s})^{g_w(0)}}{e(P_1, F_2)^{\alpha_g q_w(0)} e(P_1, \mathcal{H}_{id}(\mathcal{ID}_{user}))^{\alpha_g q_w(0)}}$$

At this point, no more translation is required and nodes can continue to be combined using interpolation outlined in Step 4 to satisfy the global thresholds outlined in the policy. Once enough nodes of the access policy are interpolated into the root node, F_R takes the following form (recall that $g_R = 1$ and $q_R = s$):

$$F_R = \frac{k' \cdot e(P_1, F_2)^{\alpha_g s}}{e(P_1, F_2)^{\alpha_g s} e(P_1, \mathcal{H}_{id}(\mathcal{ID}_{user}))^{\alpha_g s}} = \frac{k'}{e(P_1, \mathcal{H}_{id}(\mathcal{ID}_{user}))^{\alpha_g s}}$$

Now k' can be recovered using $e(C_{id}, K_{id})$ and F_R. Using k' and C_σ, recover m and x. If $\mathcal{H}(x) \neq CT_{\mathbf{com}}$ then return $\perp$, otherwise calculate $k = \mathsf{h}(x)$. If $\mathsf{MAC}_k(CT) \neq CT_{\mathbf{tag}}$ then return $\perp$. Otherwise, return m. The equations for the recovery steps are listed below:

$$k' = e(C_{id}, K_{id}) F_R = \frac{e(P_1, \mathcal{H}_{id}(\mathcal{ID}_{user}))^{\alpha_g s}(k')}{e(P_1, \mathcal{H}_{id}(\mathcal{ID}_{user}))^{\alpha_g s}} \qquad [k']^{m_\ell + x_\ell} \oplus C_\sigma = (m \parallel x)$$

4.5 Security

In this section, we discuss the security aspects of MA-AHASBE. We begin by discussing the security model in terms of a security game. We then discuss the intuition for the security of the system from both the single authority and multiple authority perspectives. Next, we discuss the complexity assumptions that the security proof requires. The complexity assumptions are static, but are non-standard. However, they are not novel to MA-AHASBE and in fact are the same assumptions that are used in LW-AHIBE [89]. We conclude by providing a sketch of the proof. The proof follows the same structure as the one presented for LW-AHIBE [89]. The details of the instantiations required modifications from the LW-AHIBE proof in order to account for multiple authorities and the additional components for ciphertexts and keys that exist in MA-AHASBE but not in LW-AHIBE. The full proof details are provided in Appendix A.

4.5.1 Security Model.

MA-AHASBE-CPA Security The following is the definition of both anonymity and semantic security in terms of a security game between a challenger and an adversary. The game generalizes the original CP-ASBE game to include multiple authorities. The game definition is given in terms of a central authority that generates the global parameters. The functionality of this central authority could be replaced by the local authorities running a secure, multi-party computation (SMPC) protocol. There are n local authorities. The game reduces to the CP-ASBE game in the case where $n = 1$. In this case, the local authority may also act as the central authority. The game definition relies on the concept of **structural equivalence** for policies. Two policies are said to be structurally equivalent if they share the same tree structure. This means that starting at the root node of each tree, nodes from both trees have the same type (e.g., translation node, global leaf node, local leaf node), the same number of children, and the same threshold values. For example, if two policies are structurally equivalent, the same q coefficients as described in the **Encrypt**

algorithm can be used to satisfy both policies. Two policies that are structurally equivalent do not need to use the same attributes as leaf nodes, or even use the same authorities in the policy. With this definition in place, the security game is defined as follows:

1. **Setup:** The setup is divided between a central authority and n local authorities.

   (a) A central authority runs the algorithm **GlobalSetup** and gives the global parameters GP to every local authority $A_k \in \{A_k\}_{k\in[1,n]}$ and to the adversary.

   (b) Every local authority $A_k \in \{A_k\}_{k\in[1,n]}$ takes the global parameters GP and runs **AuthoritySetup** to generate the set of authority parameters $\{AP_k\}_{k\in[1,n]}$. The set $\{AP_k\}_{k\in[1,n]}$ is given to the adversary.

2. **Phase 1:** The adversary makes up to q_1 queries for private keys corresponding to user keys USK_i with key structures $\mathbb{A}_i$, or domain keys DSK_i where $i \in [1, q1]$.

3. **Challenge:** The adversary submits two message-policy pairs $(M_0, \mathcal{P}_0^*)$ and $(M_1,\mathcal{P}_1^*)$. Both challenge access policies must satisfy two conditions. First, both policies must be structurally equivalent. Second, none of the private keys received in Phase 1 corresponding to user keys USK_i or domain keys DSK_i for $i \in [1, q_1]$ may satisfy either challenge access policy. The challenger chooses a random bit $b \xleftarrow{U} \{0, 1\}$ and encrypts M_b under $\mathcal{P}_b^*$. The resulting ciphertext $\mathcal{CT}^*$ is created without explicitly including the policy $\mathcal{P}_b^*$ and is given to the adversary.

   Note: If the challenge access policy only contains one global leaf node, then none of the private keys issued during Phase 1 are allowed to satisfy the challenge policy. If the challenge access policy contains at least two global leaf nodes created with the authority parameters PP_{k_1} and PP_{k_2} such that $k_1 \neq k_2$ and $k_1, k_2 \in [1, n]$, then this means the adversary could satisfy all global leaf nodes in the challenge policy with

private keys from Phase 1, but there must exist at least one global threshold that is not satisfied due to insufficient private keys issued to a single $\mathcal{ID}_{user}$.

4. **Phase 2:** Phase 1 is repeated with the same restrictions as described above for total of q queries where $q \geq q_1$.

5. **Guess:** The adversary outputs a guess b' of b.

6. **Advantage:** The advantage of an adversary $\mathcal{A}$ in this game is defined as $\text{Adv}_{\mathcal{A}} = |Pr[b' = b] - \frac{1}{2}|$.

4.5.2 Security Intuition.

The security for the MA-AHASBE system reflects the security of the two underlying systems. The leaf nodes in the access policy are essentially problem instances of LW-AHIBE, split across the access policy using an information theoretic linear secret sharing scheme. The LW-AHIBE system is proven using the dual-system encryption technique using static, but non-standard assumptions. The LW-AHIBE method for creating semi-functional, partially semi-functional and nominally functional ciphertexts and keys require no modification to work in MA-AHASBE. The only difference between the two is the form of a successful decryption. In LW-AHIBE, a successful decryption reveals the original message. In MA-AHASBE, a successful decryption of a leaf node takes the form described in the algorithm **DecryptNode**. Also, the anonymity from LW-AHIBE is used to protect selected attributes from being disclosed in the access policy. Since an attribute in MA-AHASBE is an identity in the underlying AHIBE, the security of this feature is a direct consequence of the anonymity of the underlying AHIBE. Once enough leaf nodes are decrypted, translating and combining them to satisfy the linear secret sharing thresholds becomes an instance of CP-ASBE. CP-ASBE is proven in the random oracle and generic group models [17]. However, since MA-AHASBE builds on LW-AHIBE and uses the same

dual system encryption proof structure, the entire system can be proven without oracles (random or group).

Before describing the structure of the dual system encryption proof, we first provide some additional intuition behind the security of MA-AHASBE for both the single authority and multi-authority scenarios.

4.5.2.1 Single Authority.

We first look at the single authority case. Notice that in the ciphertext, the random value k' is xor'd with the message. Thus, the adversary must recover k', otherwise the contents of the message m are information theoretically secure. The value k' only appears in one place, which is the single global leaf C_w. In particular we have:

$$C_w = k' \cdot e(P_1, F_2)^{\alpha_g s} \cdot e(P_1, P_{2A})^{\alpha_A s}$$

If we make the following definitions, where x, y, and z are all part of the authority parameters available to the adversary and s is the random value chosen during encryption:

$$x = e(P_1, F_2)^{\alpha_g}$$
$$y = e(P_1, P_{2A})^{\alpha_A}$$
$$z = xy$$
$$C_w = k' \cdot (x^s y^s) = k' \cdot z^s$$

It becomes clear that to recover k', the adversary must be able compute the inverse of the z^s term. The adversary has z through the authority parameters, but does not have s directly. However, s has been split into linear secret shares according to the challenge access policy. These shares of s are found in the leaf nodes of the ciphertext, represented by the value $q(0)$. Based on the security properties of LW-AHIBE, the only way to "recover" $q(0)$

is through successful decryption with the appropriate attribute key. In LW-AHIBE, a successful decryption reveals the original message. In MA-AHASBE, two random blinding factors have been added to the key which are unknown to the adversary. This makes a successful decryption of a leaf node take the form described in the algorithm **DecryptNode**. Note that this "recovery" of $q(0)$ actually places it in the exponent of three base elements. In particular it is of the form:

$$\frac{1}{e(P_1, P_2)^{\alpha q_t(0)} e(P_1, F_2)^{r_i q_t(0)} e(P_1, \mathcal{H}_{id}(\mathcal{ID}_{user}))^{r_{[0]} q_t(0)}}$$

At this point, the adversary must only have keys that do not satisfy the challenge policy. This may occur for one of two reasons. Either the adversary cannot properly decrypt a leaf node, or the adversary cannot perform all necessary translations. In the case where the adversary cannot properly decrypt a leaf node, we are done since there is no way to satisfy the linear secret sharing thresholds. Otherwise we are in the case where the inability to satisfy the policy occurs purely because of the inability to translate between recursive depths. At this point, the problem closely represents an embedded instance of CP-ASBE. Note that here, the domain threshold represents the root node of an CP-ASBE system.

4.5.2.2 Multiple Authorities.

We now examine the case when the challenger submits an challenge policy with multiple authorities. Unlike the single authority case, the adversary is allowed to possess keys that successfully decrypt entire subtrees rooted at global leaf nodes. In order to prevent the adversary from trivially satisfying the entire policy, the restriction is that these decryption operations must be done such that any global threshold node cannot be satisfied with global leaf nodes that have been decrypted with the same $\mathcal{ID}_{user}$. This decrypted global leaf node takes the following form:

$$F_w = \frac{(k' \cdot e(P_1, F_2)^{\alpha_g s})^{g_w(0)}}{e(P_1, F_2)^{\alpha_g q_w(0)} e(P_1, \mathcal{H}_{id}(\mathcal{ID}_{user}))^{\alpha_g q_w(0)}}$$

Here it is evident why the decryption of global leaf nodes must have been decrypted with the same $\mathcal{ID}_{user}$. The attacker at this point has the additive shares of s in the terms $q_w(0)$ given above. If the attacker attempts to interpolate the decryption of leaf nodes with different $(ID)_{user}$, then the second term in the denominator of F_w will not properly combine the shares of s. Furthermore, the attacker does not have anything that can be paired that will eliminate this blinding factor. The only potentially useful terms available are:

$$C_{id} = s\beta_{id} P_1$$
$$K_{id} = \frac{\alpha_g}{\beta_{id}} \mathcal{H}_{id}(\mathcal{ID}_{user})$$
$$K_{id,[0]} = \frac{-\alpha_g + r_{[0]}}{\beta_0} \mathcal{H}_{id}(\mathcal{ID}_{user})$$

The $K_{id,[0]}$ term is not useful, even if a translation parameter exists that would cancel the β_0 term. This is because the result would still be blinded by $r_{[0]}$ which is not available to the adversary. The remaining two terms C_{id} and K_{id} are useful for clearing the $\mathcal{ID}_{user}$ blinding factor only at the root node, in other words after enough global leaf nodes have been interpolated to meet the global root node threshold value. This is because C_{id} contains an s term and not $q(0)$ terms. Furthermore, these are the only terms available that contain β_{id}, as all the other terms used in the CP-ASBE scheme use separate β values.

4.5.2.3 Key Escrow.

In MA-AHASBE, it is important to recognize that the central authority does not have key escrow in the system. This is because the central authority does not have the α_A value which is kept private by the local authorities. The central authority does have access to the value α_g which does give the central authority the ability to facilitate a degree of collusion. In order to support collusion, a malicious central authority must find two or more

users from different domains that are willing to collude. These users can decrypt ciphertext nodes until they reach a global leaf node. At this point, the users would be prevented from combining the global leaf nodes since the leaf nodes would have been decrypted with different user keys. Critically though, at this point the local authority blinding factor has been removed and the only blinding factor remaining is tied to user identifiers. Recall a global leaf node at this point is in the following form:

$$F_w = \frac{(k' \cdot e(P_1, F_2)^{\alpha_g s})^{g_w(0)}}{e(P_1, F_2)^{\alpha_g q_w(0)} e(P_1, \mathcal{H}_{id}(\mathcal{ID}_{user}))^{\alpha_g q_w(0)}}$$

The central authority, with knowledge of α_g, can eliminate the blinding factor $e(P_1, \mathcal{H}_{id}(\mathcal{ID}_{user}))^{\alpha_g q_w(0)}$ since the ciphertext contains $q_w(0)P_1$ for each domain threshold node. The central authority can use these values to compute the $q_w(0)P_1$ value for each global leaf node through interpolation. It can now eliminate the blinding factor by pairing with $\mathcal{H}_{id}(\mathcal{ID}_{user})$ and multiplying the result with F_w. The result can now be interpolated up to the root node and the message can be recovered.

The ability to support this type of collusion is why the central authority is required to be trusted. One way to work around this is to decentralize the central authority as described previously in §4.3.2.1.

4.5.3 Complexity Assumptions.

We now review the complexity assumptions used in the MA-AHASBE security proof. Note that these assumptions are for Type-3 pairings and are based on earlier work by done in [89]. Here, $\mathcal{G} = (p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e, F_1, F_2)$ represents an asymmetric pairing and $\mathcal{A}$ is a probabilistic, polynomial time algorithm (PPTA) that outputs 0 or 1.

Assumption LW1: Define a distribution $\mathcal{D}$ as follows: $F_1 \xleftarrow{U} \mathbb{G}_1^{\times}$; $F_2 \xleftarrow{U} \mathbb{G}_2^{\times}$, $a, b, s \xleftarrow{U} \mathbb{Z}_p$, $Y_1 \xleftarrow{U} \mathbb{G}_1$;

$$\mathcal{D} = (\mathcal{G}, F_1, bsF_1, sF_1, aF_1, ab^2F_1, bF_1, b^2F_1, asF_1, b^2sF_1, b^3F_1, b^3sF_1, F_2, bF_2)$$

The LW1 problem is to decide, given $(\mathcal{D}, Z_1)$, if $Z_1 = ab^2sF_1$ or $Z_1 \in_U \mathbb{G}_1$. The advantage of an algorithm $\mathcal{A}$ in solving the LW1 problem is given by:

$$\text{Adv}_{\mathcal{A}}^{LW1} = |Pr[\mathcal{A}(\mathcal{D}, ab^2sF_1) = 1] - Pr[\mathcal{A}(\mathcal{D}, Y_1) = 1]|$$

The (ϵ, t)-LW1 assumption holds in $\mathcal{G}$ if for any adversary $\mathcal{A}$ running in time at most t, $\text{Adv}_{\mathcal{A}}^{LW1} \leq \epsilon$.

Assumption LW2: Define a distribution $\mathcal{D}$ as follows: $F_1 \xleftarrow{U} \mathbb{G}_1^{\times}$; $F_2 \xleftarrow{U} \mathbb{G}_2^{\times}$, d, b, c, $x \xleftarrow{U} \mathbb{Z}_p$, $Y_2 \xleftarrow{U} \mathbb{G}_2$;

$$\mathcal{D} = (\mathcal{G}, F_1, dF_1, d^2F_1, bxF_1, dbxF_1, d^2xF_1, F_2, dF_2, dF_2, bF_2, bF_2, cF_2)$$

The LW2 problem is to decide, given $(\mathcal{D}, Z_2)$, if $Z_2 = bcF_2$ or $Z_2 \in_U \mathbb{G}_2$. The advantage of an algorithm $\mathcal{A}$ in solving the LW2 problem is given by:

$$\text{Adv}_{\mathcal{A}}^{LW2} = |Pr[\mathcal{A}(\mathcal{D}, bcF_2) = 1] - Pr[\mathcal{A}(\mathcal{D}, Y_2) = 1]|$$

The (ϵ, t)-LW2 assumption holds in $\mathcal{G}$ if for any adversary $\mathcal{A}$ running in time at most t, $\text{Adv}_{\mathcal{A}}^{LW2} \leq \epsilon$.

Decisional Bilinear Diffie-Hellman in Type-3 pairings (DBDH-3)): Define a distribution $\mathcal{D}$ as follows: $F_1 \xleftarrow{U} \mathbb{G}_1^{\times}$; $F_2 \xleftarrow{U} \mathbb{G}_2^{\times}$, x, y, $z \xleftarrow{U} \mathbb{Z}_p$ and $Y_T \xleftarrow{U} \mathbb{G}_T$;

$$\mathcal{D} = (\mathcal{G}, F_1, xF_1, yF_1, zF_1, F_2, xF_2, yF_2)$$

The DBDH-3 problem is to decide, given $(\mathcal{D}, Z_T)$, if $Z_T = e(F_1, F_2)^{xyz}$ or $Z_T \in_U \mathbb{G}_T$. The advantage of an algorithm $\mathcal{A}$ in solving the DBDH-3 problem is given by:

$$\text{Adv}_{\mathcal{A}}^{DBDH-3} = |Pr[\mathcal{A}(\mathcal{D}, e(F_1, F_2)^{xyz}) = 1] - Pr[\mathcal{A}(\mathcal{D}, Y_T) = 1]|$$

The (ϵ, t)-DBDH-3 assumption holds in $\mathcal{G}$ if for any adversary $\mathcal{A}$ running in time at most t, $\text{Adv}_{\mathcal{A}}^{DBDH-3} \leq \epsilon$.

Assumption A1: Define a distribution $\mathcal{D}$ as follows: $F_1 \overset{U}{\leftarrow} \mathbb{G}_1^{\times}$; $F_2 \overset{U}{\leftarrow} \mathbb{G}_2^{\times}$, a, z, d, s, $x \overset{U}{\leftarrow} \mathbb{Z}_p$ and $Y_1 \overset{U}{\leftarrow} \mathbb{G}_1$;

$$\mathcal{D} = (\mathcal{G}, F_1, zF_1, dzF_1, azF_1, adzF_1, szF_1, F_2, zF_2, aF_2, xF_2, (dz - ax)F_2)$$

The A1 problem is to decide, given $(\mathcal{D}, Z_1)$, if $Z_1 = sdzF_1$ or $Z_1 \in_U \mathbb{G}_1$. The advantage of an algorithm $\mathcal{A}$ in solving the A1 problem is given by:

$$\mathrm{Adv}_{\mathcal{A}}^{A1} = |Pr[\mathcal{A}(\mathcal{D}, sdzF_1) = 1] - Pr[\mathcal{A}(\mathcal{D}, Y_1) = 1]|$$

The (ϵ, t)-A1 assumption holds in $\mathcal{G}$ if for any adversary $\mathcal{A}$ running in time at most t, $\mathrm{Adv}_{\mathcal{A}}^{A1} \leq \epsilon$.

4.5.4 Security Theorem.

We can now state the security theorem:

Theorem 4.1. *Given q is the number of adversary queries, if the (ϵ, t')-LW1, (ϵ, t')-LW2, (ϵ, t')-DBDH-3 and (ϵ, t')-A1 assumptions hold, then MA-AHASBE is (ϵ, t, q)-ANO-IND-ID-CCA secure where:*

$$\epsilon \leq \epsilon_{LW1} + 2q\epsilon_{LW2} + \epsilon_{DBDH-3} + \epsilon_{A1}$$

4.5.5 Proof Overview.

The proof of **Theorem 4.1** follows the same dual system encryption strategy as [89], with the adjustments necessary to simulate the unique aspects of MA-AHASBE. In this section we present the structural elements of the proof. The actual instantiations from the problem instances for each portion of the proof are presented in Appendix A. Intermediate steps to show that the components are well-formed are not presented in Appendix A, but since the terms follow the same structure as [89] the same intermediate steps presented there can be used for MA-AHASBE. The security for both LW-AHIBE and MA-AHASBE

depends on the following complexity assumptions: LW1, LW2, DBDH-3, and A1. In summary, the complexity assumptions play the following roles in the security reductions:

- **LW1**: The ability of the adversary to distinguish between a normal ciphertext and a semi-functional ciphertext is reduced to solving LW1.
- **LW2**: The ability of the adversary to distinguish between a normal key and a semi-functional key on the k'th key query is reduced to solving LW2.
- **DBDH-3**: The ability of the adversary to distinguish between a semi-functional encryption of the real message from a semi-functional encryption of a random element of $\mathbb{G}_T$ is reduced to solving DBDH-3.
- **A1**: The ability of the adversary to distinguish between a random message encrypted under a random, structurally equivalent policy and a chosen message encrypted under a chosen policy is reduced to solving A1.

The proof uses a simulator that works in a manner very similar to the simulator used in LW-AHIBE. The key difference is that the simulator for MA-AHASBE has to simulate a set of global parameters, authority parameters for multiple authorities, a more complex ciphertext relative to LW-AHIBE, as well as some additional key elements. This simulator takes as input LW1, LW2, DBDH-3 and A1 with either *real* components or *random* components. The same hybrid sequence of $2q + 4$ games defined in [89] are used here:

- $Game_{real}$: The real security game defined in Section 4.5.1.
- $Game_0$: The challenge ciphertext is semi-functional and all keys returned are normal.
- $Game_{k,0}$ for $1 \leq k \leq q$: The k'th key is partial semi-functional, the first $k-1$ keys are semi-functional and the rest of the keys are normal.
- $Game_{k,1}$ for $1 \leq k \leq q$: The k'th key is fully semi-functional.

- $Game_{M-random}$: All keys returned are semi-functional and the challenge ciphertext encrypts a random message.

- $Game_{final}$: All keys returned are semi-functional and the challenge ciphertext encrypts a random message to a random, structurally equivalent policy.

The games are ordered as $Game_{real}$, $Game_0$, $Game_{1,0}$, $Game_{1,1}$, ..., $Game_{q,0}$, $Game_{q,1}$, $Game_{M-random}$, $Game_{final}$. Let X_{real}, X_0, $X_{1,0}$, $X_{1,1}$, ..., $X_{q,0}$, $X_{q,1}$, $X_{M-random}$, X_{final} be the event that $\mathcal{A}$ wins in $Game_{real}$, $Game_0$, $Game_{1,0}$, $Game_{1,1}$, ..., $Game_{q,0}$, $Game_{q,1}$, $Game_{M-random}$, $Game_{final}$ respectively. $Game_{0,1}$ is the same as $Game_0$.

It is important to recognize that the main components in MA-AHASBE are *nearly* direct instances of LW-AHIBE. For user keys, the key difference is that the function **RestrictToUser** appends two additional components onto the $K_{2,1}$ portion of the key. During leaf node decryption, these additive terms become multiplicative terms in the decrypted result. In the ciphertext, there are q terms that represent the secret split over a policy. It is a straightforward exercise to show that these additional terms do not interfere with the interaction between normal and semi-functional ciphertexts with normal, semi-functional, partially semi-functional, and nominally functional keys. Therefore, MA-AHASBE can directly leverage the constructions as described in [89] for creating semi-functional components without modification. MA-AHASBE semi-functional ciphertext components are embedded in the local leaf nodes. MA-AHASBE domain keys map directly to LW-AHIBE keys so creating semi-functional domain keys is exactly the same as LW-AHIBE. A semi-functional user key is one that is generated from a semi-functional domain key. The same logic applies to all types of semi-functionality (partial, nominal, normal). It is straightforward to verify that MA-AHASBE semi-functional keys and ciphertexts behave as expected for a dual system encryption proof.

In $Game_{final}$, a random message is encrypted to random policy, so we know that $Pr[X_{final}] = \frac{1}{2}$. In order to support the inequality in Theorem 1, we prove that the same five lemmas presented in [89] hold for MA-AHASBE. These lemmas are:

- **Lemma 4.1**: $|Pr[X_{real}] - Pr[X_0]| \leq \epsilon_{LW1}$
- **Lemma 4.2**: $|Pr[X_{k-1,1}] - Pr[X_{k,0}]| \leq \epsilon_{LW2}$ for $1 \leq k \leq q$
- **Lemma 4.3**: $|Pr[X_{k,0}] - Pr[X_{k,1}]| \leq \epsilon_{LW2}$ for $1 \leq k \leq q$
- **Lemma 4.4**: $|Pr[X_{q,1}] - Pr[X_{M-random}]| \leq \epsilon_{DBDH-3}$
- **Lemma 4.5**: $|Pr[X_{M-random}] - Pr[X_{final}]| \leq \epsilon_{A1}$

The actual instantiations for each lemma are presented in Appendix A.

4.5.6 Chosen Ciphertext Security Security.

The construction for MA-AHASBE uses the generic chosen ciphertext security (CCA) transform described in [19]. Often systems are demonstrated only with CPA security, given the existence of such generic transforms. In the case of MA-AHASBE, care must be taken to ensure that the transform is properly applied. In particular, this is due to the requirement that the CCA transformation requires that the encrypting party be able to generate the private key for an arbitrary identity. In other words, it must act as a private key generator for an identity whose value is the commitment message. This functionality is provided by the LW-AHIBE system by allowing users to delegate to arbitrary identities, assuming there are enough levels of delegation authority present in that user's private key. The ability to finely control this amount of delegation authority is necessary for MA-AHASBE. If a user is allowed to delegate arbitrarily, they could then become unauthorized subdomains.

The key aspect of the security of MA-AHASBE with regards to the issues described above is that the user key is both *required* to delegate one level in order to decrypt and that the user key is *authorized* to delegate exactly one level. If the user were to try to misuse

their one level of delegation to forge a key for an unauthorized subdomain, the resulting key would not be able to create further delegations. Since at least one level of delegation is required to decrypt, this new key would be useless. Since this tight control over delegation authority is a strict requirement, the MA-AHASBE construction is described with the transform applied. The proof of security given above demonstrates that MA-AHASBE is CPA secure. It is a straight forward exercise to verify that the additional components involved in the construction are a direct application of CCA transform described in [19].

4.6 Conclusions and Future Work

The MA-AHASBE system fills a gap in CP-ABE systems, specifically through its combination of large universe, multi-authority, delegation and attribute-set capabilities. Future work involves analyzing its performance, especially in the decentralized case. Also, it may be possible to use the same techniques with another AHIBE [90] to build a system with stronger security features. It would also be useful in an enterprise environment to sign messages with attributes, either compound or singleton, in order to verify the authenticity of messages. In MA-AHASBE, a ciphertext can be created by anyone, so a signing mechanism would allow users to verify that messages have been viewed/approved by other users with certain attributes. A straight forward approach might be taken based on the suggestions found in [47] for the singleton case.

V. MA-AHASBE Based Access Control in a Cloud Supported Publish-Subscribe Data Model

This chapter presents an approach for securing data with public cloud service providers (CSPs). It defines two models of security which emphasize that CSPs may be trusted with availability operations, but not with confidentiality. One model called A-trusted provides no further trust beyond availability operations, while a more relaxed variant called ACE-trusted allows computation (i.e., system integrity) and certain access control decisions. The A-trusted model may be appropriate for peer-to-peer type networks, while the ACE-trusted model may be more appropriate for CSPs. In order to build applications with these security models, a framework called Axon is introduced. Axon bridges applications and CSPs by providing a small set of highly available data structures: maps, queues, and topics. Security can then be layered on these data structures to create ACE-trusted data structures and protocols. A microblogging application called Critter is built using this layered approach. Experimental results show that, while there is certainly cost associated with encryption, decryption and increased payload sizes, the security tradeoff may be worth it for situations where preventing disclosure of sensitive data is critical.

5.1 Introduction

Cloud computing is becoming an important method for storing and processing data. One of the primary draws of public cloud computing is that cloud service providers (CSPs) can leverage economy of scale to provide cost-effective services. However, outsourcing data to a third party brings with it important security concerns. This chapter explores an approach that helps to address these concerns.

To address the issue of security, we propose a security model based on the information security principle of availability. While the information security principles of confidentiality and integrity can typically be enforced using cryptographic techniques, the requirement to trust CSPs with availability of data is difficult to avoid without forcing clients to maintain an entire replica of all data provided to the CSP. In many cases, this approach defeats the purpose of using cloud services. Fortunately, the business model for CSPs usually depends on providing high availability of data. With CSPs capable of providing high availability service level agreements (SLAs) to their clients, the security concern shifts from availability to confidentiality and integrity. Our proposed security model, called *Availability-trusted* (or *A-trusted*), only trusts the CSPs with making sure data CSPs receive are made available to clients according to any applicable SLAs. A specialization of this security model called *Availability-Computability-Enforceability-trusted* (or *ACE-Trusted*) adds additional trust with computation and the enforcement of availability access controls (not confidentiality access control). The A-trusted model is appropriate for situations where there is a minimal amount of trust, such as in peer-to-peer networks. The ACE-trusted model is appropriate for situations such as a CSP, where some additional trust is warranted in order to avoid the extra computational cost associated with the A-trusted model.

We believe that by adopting such conservative security models, clients may be more willing to build cloud computing applications that must process sensitive data that clients may otherwise be unwilling to outsource to a CSP. In order to investigate how applications could be built under these models, we have developed a simple messaging application we call Critter. Critter supports the basic messaging capabilities of popular microblogging applications such as Twitter and Instagram. It allows users to post messages to their public feed, where any followers of that feed will receive them. Users may tag messages by including the hashtag symbol # followed by a keyword of the user's choice. Users may

then follow these tags for topics which interest them. Users may also direct messages to only be sent to specific users rather than their public feed by including the target users' handle preceded by the @ symbol. In Critter, all messages are protected from disclosure to the CSP through encryption. We believe that Critter represents an application that has sufficient complexity to be useful for understanding the challenges of these models, while being simple enough to study and understand how the underlying security components work.

Finally, in building Critter we realized that we needed a framework to help bridge the gap between the requirements of the security model and the services provided by a CSP. We describe Axon, which is the resulting framework that we developed. The core concept of Axon is that the interface between client applications and cloud infrastructure should be a small set of easily understood data structures. Currently, Axon supports three core data structures: maps (also called dictionaries or key-value stores), queues, and topics. Axon also supports a distributed event system that informs connected clients when changes to these data structures occur. By focusing on a small set of data structures, the cloud side of Axon can ensure that the data placed in these data structures is highly available and persistent. On the client side, the core data structures serve as building blocks for more complex structures that provide confidentiality and integrity guarantees that support the security model. Applications such as Critter can then use these secure data structures. This layered approach to security provides a way to manage the complexity that arises for applications that require both security and high availability.

This chapter is divided into six sections. The next section discusses background and related work. We then discuss our proposed security and dependability models in Section 3. In Section 4, we describe the Axon framework. We describe how we use Axon to build the Critter application in Section 5 and present experimental results. We conclude the paper in Section 6 along with thoughts on future work.

5.2 Background and Related Work

5.2.1 Definitions.

Attribute-Based Access Control An access control method where subject requests to perform operations on objects are granted or denied based on assigned attributes of the subject, assigned attributes of the object, environment conditions, and a set of policies that are specified in terms of those attributes and conditions (NIST definition [58]).

Key-Pair Public Key Encryption (KP-PKE) This type of public key cryptography refers to asymmetric (public/private) key pairs that are *tightly coupled* in that one cannot be changed independent from the other (e.g., RSA, ElGamal). This term is used to distinguish it from other types of public key cryptography such as identity-based encryption [95] (or attribute-based encryption [13, 94]) where the public keys are fixed strings and the private keys are updated independently by a private key generator.

Ciphertext-Policy Attribute-Based Encryption (CP-ABE) This term refers to a family of cryptographic systems [13, 31, 32, 52] where users are given cryptographic keys that represent attributes. In many CP-ABE systems, the attributes are fixed strings (e.g., "User", "Active", "Student"). The cryptographic keys corresponding to these fixed string attributes are provided by a trusted private key generator. Ciphertext is created by combining an access policy over a set of attributes, a message, and a set of public parameters provided by the private key generator.

Authenticated Encryption with Associated Data (AEAD) This is a type of symmetric key encryption which provides confidentiality, integrity, and authenticity for the encrypted data. The AEAD encryption takes as input a plaintext that requires confidentiality protection, plaintext that will not be encrypted but must not be altered

(i.e., the associated data), and the encryption key. As a result, the AEAD encryption algorithm produces ciphertext and an authentication tag. AEAD decryption takes as input the ciphertext, the plaintext associated data, authentication tag, and the encryption key. The decryption algorithm only succeeds if the data (both the ciphertext and the associated data) have not been altered in any way. If decryption succeeds it produces the original plaintext. If the data has been altered, the decryption will detect this condition and fail. For this paper, we assume that the AEAD algorithm is random in that each invocation of the algorithm produces different results even if the same inputs are used.

Message Authentication Code (MAC) A message authentication code is a cryptographic primitive that provides both integrity and authentication guarantees for messages. A MAC generation algorithm takes as input a message and a cryptographic key and produces a tag. A corresponding verification algorithm takes as input a message, a cryptographic key and a tag produced by the generation algorithm. If the tag was created by the same message and cryptographic key as the one provided to the verification algorithm, the verification succeeds and fails otherwise. Like a cryptographic hash function, a MAC ensures that a message has not been altered, thereby providing integrity. However, unlike a cryptographic hash function, the MAC algorithms require a secret key. This provides an authentication mechanism since only someone in possession of the secret key can create tags that verify with that particular key.

WebSockets / Long Polling WebSockets is a protocol that provides full-duplex communication over the Hypertext Transfer Protocol (HTTP). It is primarily designed to facilitate real-time applications that run in Internet browsers, but can be used in anywhere where full-duplex communication is desired. However, WebSocket connections may fail when behind certain types of proxies which do not support them. Long polling

is an alternative technique that also provides full-duplex communication over HTTP. Long polling will generally work through proxies and firewalls, but has increased latency and overhead costs when compared to a protocol such as WebSockets. In long polling, a client sends a normal HTTP request for data. If the server has data to send, it will send it immediately and close the HTTP connection. When the client receives the data, it immediately issues another request. When the server does not have data to send the client, it keeps its side of the HTTP connection open until it either has data to send or a timeout occurs. The worst case latency occurs when the server determines it has data to send the client immediately after it has closed the previous HTTP connection (either due to a previous message or a timeout occuring). In this case, the server must wait for the client to issue its next connection request in order to send the data.

5.2.2 Related Work.

Access control and confidentiality in publish/subscribe (pub-sub) networks has been an active research area for many years. Pub-sub systems can be described as either topic based or content based. In topic based pub-sub, users subscribe to topics by keyword. Any messages published to that topic will be received by users who are subscribed. Content based pub-sub is an enhancement where users submit filters that describe the type of content that they wish to recieve. When a message is published to the system, if the content of a message matches the filter provider it is routed to the user.

While work has been done in protecting confidentiality and integrity for both content based pub-sub ([8, 12, 59, 60, 77, 86]) and topic based systems, this work focuses on a topic based systems. Due to the recent popularity of social networks such as Twitter and Instagram using this type of messaging system, this type of messaging is also referred to as microblogging. One of the first examples of protecting confidentiality in a microblogging application is #h00t [6]. The goal of #h00t is to protect messages from being intercepted

and censored by the service provider or some other controlling entity such as a government. The system relies on shared group keys that are distributed in an ad-hoc fashion amongst group members. Another system Hummingbird [33] also protects the content of tweets by encrypting them first. Hummingbird uses efficient cryptographic primitives, but requires that followers request approval from publishers to follow certain topics. In [87], the authors propose a technique called k-subscription to protect the service provider from discovering which channels in which users have an interest. Users subscribe to additional noise channels in order to mask their true channels of interest. The technique is not designed to protect the confidentiality of the content of the tweets, instead relying on a system such as #h00t or Hummingbird for that functionality.

5.2.3 MA-AHASBE.

MA-AHASBE (Multi-Authority Anonymous Hierarchical Attribute-Set Based Encryption) is a ciphertext-policy attribute-based encryption (CP-ABE) [13] system with a rich set of capabilities. Here we provide a brief summary of its features. Full details of the system including security proofs can be found in its full technical report [99].

MA-AHASBE was developed to help facilitate the use of CP-ABE within an federated, enterprise system. In this type of system, there may be multiple distinct organizations that wish to work together and share a common attribute infrastructure. In MA-AHASBE, each of these organizations is called an authority. If these authorities cooperate and agree on a common set of global parameters, including a list of user identifiers, it is possible for users to receive attribute keys from each authority and use them to satisfy policies. This is useful in situations where a single user must possess attributes from multiple organizations. For example, say a student at a university is doing an research internship for a corporation. There may be documents that should only be accessible to students from the university who are participating in the internship. Under a multi-authority system such as MA-AHASBE, the student can receive a “Student” attribute

from the university and a “Intern” attribute from the corporation. The student could then satisfy an access policy that requires both attributes. The two authorities do not need to share any secret key material in order to do this, they must only agree on the user identifier to use (which is part of the public, global parameters) when generating the attributes.

MA-AHASBE also allows authorities to delegate responsible to lower level domains. In MA-AHASBE, these lower level domains have full autonomy to create keys for any attributes they need. The keys they generate, however, are bound to that specific domain. For example, the university in the previous example may wish to delegate some if its authority to the college of engineering. We distinguish attributes from different domains by writing the full hierarchy of the attribute in a tuple. For example, if the university created a “Student” attribute, that attribute would be written ⟨"University", "Student"⟩. If the university delegated attribute generation authority to the college of engineering, and the engineering college in turn created a “Student” attribute, it would be written ⟨"University", "College of Engineering", "Student"⟩. As with the multi-authority example, these attributes could be combined in a single access policy that would require a student to have credentials from both the university and the college of engineering. Allowing authorities to delegate the ability to generate attributes in an autonomous way is crucial for large organizations. It allows for the maintenance of attribute keys to be managed in the same structure as the organization itself.

The complexity of a CP-ABE system such as MA-AHASBE provides a great deal of expressiveness in protecting the confidentiality and integrity of data. However, this expressiveness comes at a cost. MA-AHASBE operations are typically slower than AEAD or even KP-PKE operations. The encryption time and size of the resulting ciphertext scales with the number of leaf nodes in an access policy. The decryption time scales with the number of nodes that are required to satisfy the policy. As such, MA-AHASBE is used in conjunction with AEAD and KP-PKE. By combining MA-AHASBE with AEAD and KP-

PKE, some of the performance cost can be mitigated while maintaining the expressiveness of MA-AHASBE.

There are two basic ways of employing MA-AHASBE. The first way is shown in Figure 5.1. The shaded portions of the figure indicate data that is meant to be private, while the unshaded portions may be put in a public location (such as a public cloud). Here, MA-AHASBE is used to protect a symmetric key with an access policy. The ciphertext produced by this encryption is much larger than the original message. A 128-bit symmetric key encrypted with the policy shown in Figure 5.1 may result in ciphertext size of a few kilobytes (experimental results are discussed in more detail in §5.5). The symmetric key is then used to encrypt the document using AEAD which is much faster and produces ciphertext that is close in size to the original message. MA-AHASBE and AEAD both protect against tampering with ciphertext. Ciphertexts that are not the result of following the encryption algorithms will produce an error when decryption is attempted, as opposed to producing an incorrect plaintext. Decryption reverses the process. A user that possess MA-AHASBE attribute keys that satisfies the policy can recover the symmetric key. This symmetric key can then be used to recover the document.

The method of using a symmetric key directly with MA-AHASBE is useful when a policy is only used once. When a policy needs to be used with multiple documents, it can be more efficient to augment the approach with a KP-PKE system. This modified encryption flow is shown in Figure 5.2. Here, instead of using MA-AHASBE to encrypt a symmetric key, it is used to encrypt the private key of a KP-PKE key pair. The corresponding public key part of the KP-PKE key pair then used to protect the symmetric key. The benefit of doing this is that to encrypt another document under the same policy, only KP-PKE operations are needed. KP-PKE operations can be much more efficient than MA-AHASBE operations, especially for large policies. This savings is carried over to decryption operations as well.

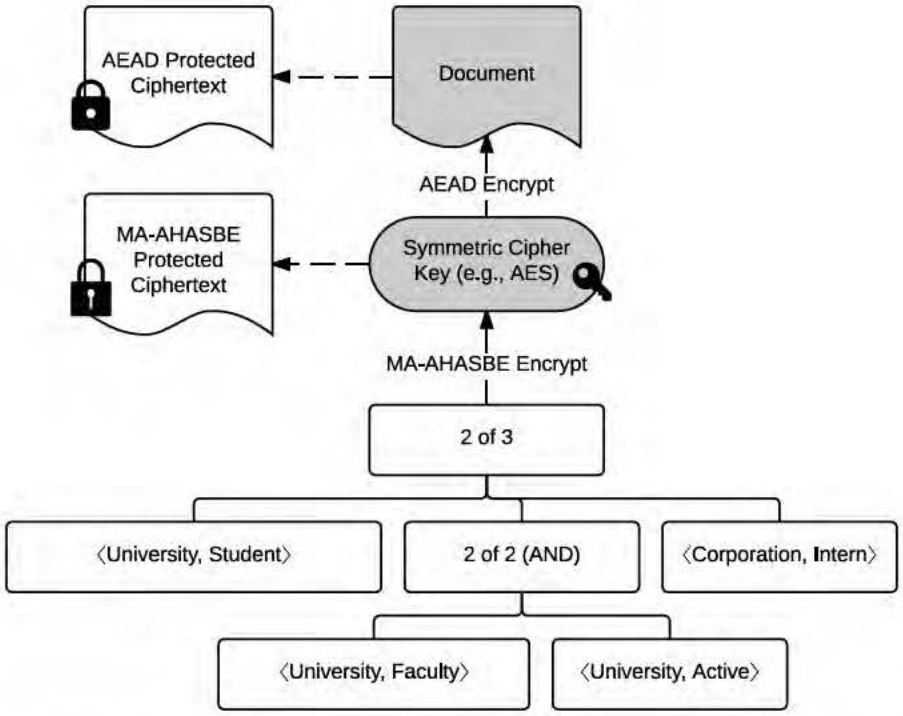


Figure 5.1: MA-AHASBE encryption flow with AEAD only. Shaded portions indicate private data and dashed arrows indicate the result of an encryption operation.

Finally, MA-AHASBE also allows documents to be signed with attributes. This is useful for providing authenticity to data. Although not used in this research, MA-AHASBE also allows protecting attributes from being disclosed in policies and combining attributes into sets [17] which allows for more complex policies.

5.2.4 Tools.

Amazon Web Services Amazon provides one of the most popular public cloud computing infrastructures. It offers a wide range of services that include file storage (S3 and Glacier), NoSQL (DynamoDB) and SQL (RDS) databases, and virtual machine instances (EC2) for general computing.

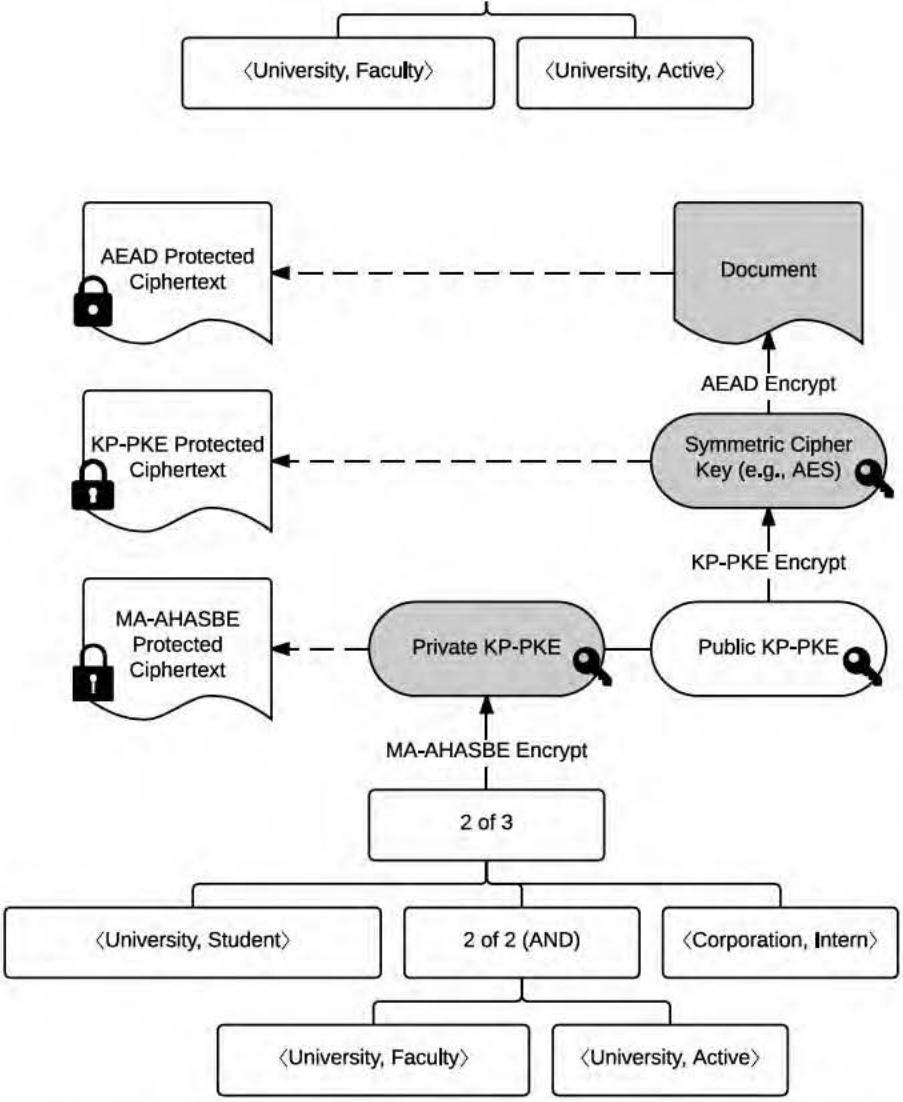


Figure 5.2: MA-AHASBE encryption flow with KP-PKE and AEAD. Shaded portions indicate private data and dashed arrows indicate the result of an encryption operation.

Hazelcast Hazelcast is an open source Java project that provides distributed, in-memory data structures such as sets, maps, queues and topics.

Sodium Sodium is a modern reimplementation of the NaCl cryptographic library. It contains implementations of a wide variety of cryptographic primitives such as AEAD, KP-PKE, digital signatures, and MAC. The project is open source and written in cross-platform C. Bindings exist for a variety of languages including Java, and there is a Javascript build that has been compiled with the Emscripten cross-compiler.

SocketIO SocketIO is a protocol for establishing real-time, full-duplex, event based connections. It uses WebSockets as the primary connection protocol with a fallback to long polling. The primary project is open source and provides both server and client implementations written in Javascript. There are other open source projects which provide Java clients and servers.

5.3 Security Model

When discussing security and cloud computing, it is important to describe who is trusted as well as the meaning of that trust. Many protocols related to secure cloud computing use some variation of the *honest-but-curious* (HBC) [27, 33, 59, 86, 87, 101] model with potentially *malicious* actors. In an HBC model, HBC actors faithfully follow the underlying protocol but remember all the messages they see and attempt to violate privacy if they can. Malicious actors are allowed to behave however they like (e.g., they may lie, cheat, attempt to impersonate others, fail to participate).

A traditional HBC model may not be a satisfying security model for cloud computing. For example, there may be instances where the cloud service provider (CSP) can reasonably be expected to follow some protocols in an HBC fashion, but may follow other parts of the protocol as a malicious actor. For example, in a simple protocol that only stores and retrieves data a CSP might fall under an HBC model where it is expected that the stored data is retrievable at some point in the future. In other words, it is reasonable to believe that a CSP wouldn't maliciously destroy data or otherwise make it unavailable. Realistically though, data becomes corrupted despite the large amount of effort CSPs put into data durability. The durability of the data is usually guaranteed by the CSP up to some small amount of loss specified through a service level agreement. Additionally, while any reasonable CSP wouldn't purposefully tamper with data, it is not impossible. Situations can arise such as malicious employees, or third party attackers gaining control of the CSP and tampering with data. If we model the CSP in this storage service protocol as HBC, we may

not be accurately capturing the fact that this loss occurs and we may be making any other protocols that depend on this protocol vulnerable to attack. While this seems unsettling, modeling the CSP as malicious doesn't seem to represent reality either. If the CSP is modeled as malicious, the CSP may arbitrarily decide to make the data unavailable or may make changes to any amount of data it wishes. Modeling the CSP as purely malicious seems to defeat the purpose of using the CSP in the first place.

Instead, we propose two security models called *A-trusted* and *ACE-trusted* that are still conceptually straightforward, but do a cleaner job of separating out the concerns that are unique to cloud computing. The key insight is that when data is stored with a CSP, an explicit trust relationship is formed with regard to availability if that data is ever expected to be retrieved in the future. Luckily, the core business model of most CSPs is to supply strong availability guarantees. Accepting this trust relationship seems unavoidable when dealing with exporting data to a third party. However, this trust does not necessarily extend to other aspects of information security such as confidentiality and integrity. The goal of the A-trusted and ACE-trusted security models is to more accurately capture this difference.

5.3.1 The A-trusted and ACE-trusted Security Models.

What is needed is something between HBC and malicious. We could perhaps call this middle ground *semi-malicious*, but this seems overly vague and unintuitive. Instead, a clearer approach may be to use the building blocks of information security: confidentiality, integrity, and availability. The NIST definitions for these terms are listed below [81]:

Confidentiality Confidentiality is the requirement that private or confidential information not be disclosed to unauthorized individuals. Confidentiality protection applies to data in storage, during processing, and while in transit.

Integrity Integrity can be one of two types:

Data Integrity The property that data has not been altered in an unauthorized manner while in storage, during processing, or while in transit.

System Integrity The quality that a system has when performing the intended function in an unimpaired manner, free from unauthorized manipulation.

Availability Availability is a requirement intended to assure that systems work promptly and service is not denied to authorized users. This objective protects against:

- Intentional or accidental attempts to either:
  - perform unauthorized deletion of data
  - cause a denial of service or data.
- Attempts to use system or data for unauthorized purposes

Although not part of the official NIST definition, it may be helpful to view the components of confidentiality in terms of the popular information gathering rules of five *W*s and one *H* or *5W1H*. Even though many encryption oriented security solutions only focus on the *what* component in terms of confidentiality, protecting each method of information gathering typically incurs some type of performance cost. Any proposed solution dealing with secure cloud computing should address which components of 5W1H are explicitly supported and which ones are assumed. For the applications in this research, only the *what* aspect of confidentiality is supported. Each component of 5W1H is listed here:

Who is the source that generated the information

What is the content of the information

When was the information generated

Where is the geographic or logical location in a network of the entity that generated the data

Why was the information produced or which events trigger its creation

How is the data protected (i.e., what is the encryption scheme or protocol)

As with confidentiality, it may be helpful to break the definition of integrity down into more familiar parts. Here we consider data integrity to consist of two components: **corruption** and **authentication**. Corruption occurs when a value is changed outside a predetermined computation or protocol. This occurs benignly in variety of situations such as component failure in storage drives or transmission hardware. If the source of the corruption is benign, it can be detected with techniques such as checksums or hash functions. Malicious corruption requires different techniques such as message authentication codes or digital signatures. Authentication is the process of certifying a value with a set of credentials along with the process of verifying the certification. This is critical to scenarios where validating the source of the information is as important as the data itself.

Finally, it is also helpful to examine the components of availability. One popular data storage model consists of the operations Create, Read, Update, Delete (CRUD). Variations of this model include Browse, Read, Edit, Add, Delete (BREAD) and Delete, Read, Update, Lock, Add, Browse (DRULAB). The CRUD model seems to be the closest fit that is parsimonious and captures the basic functions of CSP availability. However, in some ways CRUD does not perfectly align with how we may want to view CSP availability operations in our security model. For example, if a delete operation is sent to the CSP, there is generally no way for the client to know if the information is truly (or securely) being deleted. Requiring that the data is actually deleted may capture more trust than we actually require or expect from a CSP. A delete operation to a CSP is really just way for

the client to inform the CSP that it no longer needs access to the data and that it should no longer be charged by the CSP to make it available. This is weaker than the assumption that the CSP actually deletes the data upon request, and arguably more accurately captures the trust relationship. Therefore, in order to facilitate a more fine-grained discussion of what services a CSP is responsible to provide, we present the Function, Search, Create, Read, Annul, Write, Lock (F-SCRAWL) model.

Function The client submits a computation request to the CSP and the CSP returns a result that can be verified as correct by the client more efficiently than simply performing the computation directly.

Search The client may present a query (e.g., SQL or NoSQL) to the CSP and the CSP will return the subset of valid data that matches the query parameters.

Create The client requests that a datum be stored and marked as valid for later operations.

Read The client requests access to the current value of a datum that has previously been stored with the **Create** operation.

Annul The client marks a datum as invalid and will no longer perform operations for this datum (i.e., any future operations for this datum should be considered invalid).

Write The client wishes to update an datum previously stored with the **Create** operation to a new value (could include concurrency primitives such as compare and swap).

Lock The client wishes to prevent other clients access (e.g., **Read**, **Write**, **Lock**, **Annul**) to a datum previously stored with the **Create** operation until the lock is released (either by the client or the CSP).

While the CRAWL portion of F-SCRAWL seems directly applicable to the availability aspect of a CSP, it may not be as clear for function and search operations. The function

operation has been carefully defined to only allow computation which can be efficiently verified as correct by the client (e.g., the result of running a secure, multiparty computation protocol). This condition is what makes the function operation an *availability* operation rather than an *integrity* operation. If the CSP deviates from the computation protocol or returns an incorrect answer (violating *system integrity* as defined above), then the client detects this and discards the result. If this occurs with a high enough frequency, the CSP is viewed as not being available enough for computation purposes. Search is another subtle operation that requires availability trust. Here the trust is not in the correctness of the returned results (which can be verified by the client), but in that the returned set is complete. Removing this trust would require the client to verify that the CSP is returning all data (i.e., making it available) that matches the query. This could of course be done if all the data was also stored client side, but this may defeat the purpose of storing data with a CSP in the first place. It is difficult to provide this for a general data set and set of queries, therefore we make it part of the trusted CSP protocol.

With a more precise view of the CIA model, we now consider what type of trust we need to place in a CSP. We argue that the only trust we should put into a CSP is in the availability of F-SCRAWL operations. Outsourcing data and services to a CSP requires some level of availability trust. In a sense, we are narrowing the scope of the CSP to be HBC only with respect to F-SCRAWL operations. In other words, the user trusts that the CSP will not recklessly destroy, corrupt, or otherwise make the data unavailable. Some data loss or service failure will occur naturally due to disasters, hardware failures or denial-of-service attacks. The rate at which this is expected to occur will generally be outlined in the service level agreement (SLA) of the CSP. Also, conformance to the SLA is something that can be monitored by the client assuming the client is checking the integrity of the data returned by the CSP. If the risk as stated in the SLA is too high or if the CSP is not abiding

by the SLA, one mitigation approach would be to replicate the data or services amongst multiple CSPs.

With the F-SCRAWL model in place, we can now more precisely talk about what types of trusted systems play a role in a secure cloud computing scenario. We divide the trust levels into three paradigms: fully trusted, A-trusted (with ACE-trusted as a special case), and malicious. Each paradigm is listed below along with a short description:

Fully trusted (or CIA-trusted) A fully trusted system is one where sensitive plaintext data such as cryptographic keys, personally identifiable information (PII), or sensitive business data may be stored in plaintext (either on disk or in memory).

Examples:

- Client systems of authenticated users
- Servers deployed in a private cloud

Availability-trusted (or A-trusted) An availability-trusted system will faithfully follow data availability protocols (F-SCRAWL) and fulfill availability SLAs. However, it will exploit sensitive data to which it has access but does not require in order to perform its duties (i.e., violates confidentiality). It will manipulate data available to it in order to accomplish its goals if the data manipulation cannot be easily detected by the data owner (i.e., violates integrity). If the system is **Availability-Computability-Enforceability-trusted** or **ACE-trusted**, the system is trusted to maintain system integrity with respect to computations it is asked to perform as well as enforce access control rules it receives with respect to third parties. The CSP will still attempt to exploit the data it receives if it can, but will accept and enforce rules regarding how third parties may use the F-SCRAWL protocols. ACE-trusted is a special case of

A-trusted.

Examples:

- Servers deployed in a public or community cloud
- Public CSPs may be considered ACE-trusted while a peer-to-peer network may be considered only A-trusted
- Authenticated users who wish to collude to access information to which they would not otherwise have access

Malicious Malicious entities are willing to break protocols and subvert systems in order to gain access to information or systems. Malicious users may be internal or external. Internal malicious users are those who abuse legitimate access to systems or cryptographic keys. External malicious users do not have legitimate access to systems or cryptographic keys and use a variety of techniques to bypass security mechanisms.

Examples:

- Everyone else
- Evil hackers
- Insider threat

5.4 Axon: A Data Structure Approach to Security, Scalability and Dependability

In order to bridge the gap between the ACE-trusted security model and applications, we built a framework to facilitate the communication between an application and a CSP. This framework is called Axon[3]. This section discusses the design decisions behind Axon and how it supports the ACE-trusted security model.

[3]The name Axon comes from the nerve fibers that connect neurons inside the brain and act as information transmission pathways

5.4.1 Architecture.

By design, Axon is a very small interface from the perspective of an application. In fact, in order to support the applications described in the next section, only three data structures are used. Each data structure has a small number of functions that can be called. These data structures become the only interface between the application code and the CSP. The data structures used for the applications in described in the next section are shown in Table 5.1.

Table 5.1: Axon Data Structures

Data Structure	**Example Functions**
Map	insert, update, remove, lookup, enumerate-keys
Queue	enqueue, dequeue
Topic	publish, subscribe, lookup

Focusing on a small set of data structures as the key interface between the application and the cloud provides a number of key benefits. First, it removes the responsibility of scaling and durability away from the application code. From the application's perspective, any data stored into the Axon data structures is expected to be highly available. Secondly, since there are only a few ways to store data, applications have fewer places to check to make sure that data is properly protected. However, the restriction to these data sets along with the condition that any information placed inside them must be encrypted puts some additional complexity burden on the applications. Techniques for dealing with this complexity are discussed in the next section.

The overall information flow for an Axon enabled application is shown in Figure 5.3. Cryptographic keys begin their life within a private cloud infrastructure that supports a hierarchy of private key generators. These private key generators distribute keys

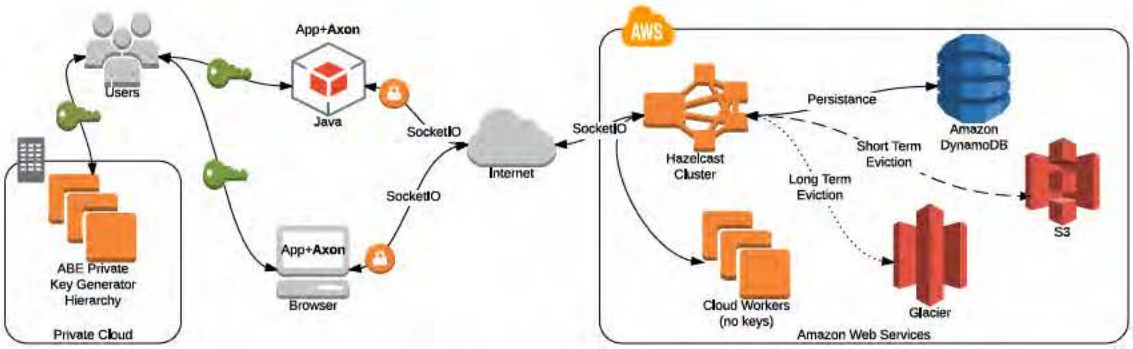


Figure 5.3: Axon Information Flow

representing user attributes to each of the users in the system. Users provide their keys to Axon enabled applications where any information that is stored in Axon data structures is required to be encrypted. The client side of the Axon framework then takes the request and sends it across a public channel (e.g., the Internet) to a Hazelcast server over a SocketIO connection. These requests consist of a data type (e.g., topic, map, queue), the data structure's name as a string (e.g., "parameters", "users"), the name of the function (e.g,. add, remove), and any required function parameters. When the cluster receives the request, it must determine if the client is authorized to perform that operation on that specific data structure.

In the current prototype implementation of Axon, all users are allowed to create data structures as long as the name for the data structure doesn't already exist. The creator of the data structure specifies the permissions for each operation that the data structure allows. The creator does this by generating a unique KP-PKE key pair for each function. In the request, the name of the function is bound to the public portion of the key pair. The private key is encrypted using attribute-based encryption and is also included in the request. When the creator sends the request, it includes for each function the public key, the protected private key, along with a authorization session timeout value. When the cluster receives the request, it creates the requested data structure along with a permissions map. The public

key, protected private key pair and timeout value are stored in this permission maps. When requests come into the cluster to perform operations on the data structure, the cluster first looks to see if the client has an active authorization session. If this is the first time this client has requested to do an operation on a specific data structure or the clients previous session has reached its timeout value, then it will not have an active authorization session. The server will look up the public key for the operation and encrypt a random value (also called a nonce, or number used once) with that public key. It will then send this encrypted nonce along with the protected private key to the client. If the client is able to satisfy the policy associated with the protected private key, it is able to recover the private portion of the KP-PKE. It can then use this private key to recover the random nonce sent by the server. It returns this value to the server, and if the value matches the random nonce that the server originally sent the client, an authorization session is created and timestamped. The client is then allowed to perform the associated function until the session reaches the timeout specified in the creation request. An example of the message exchanges involved in the creation of a data structure as well as a client gaining authorization to perform a function is shown in Figure 5.4.

This protocol serves as a good example of the difference between the A-trusted and ACE-trusted security models. Under the A-trusted security model, the CSP would not be bound by this authorization protocol and would be free to fulfill data requests from any client. Therefore, under an A-trusted model, additional measures must be taken to enforce these types of permissions. For example, to enforce a "write" type permission clients could include a signature on all stored data that represented a "write" permission attribute using an attribute-based signature scheme. Clients accessing the data structure would only consider the data valid if it contains a proper signature. The choice of security model is of course application and client specific, but we believe that the ACE-trusted security model provides the right balance for many situations.

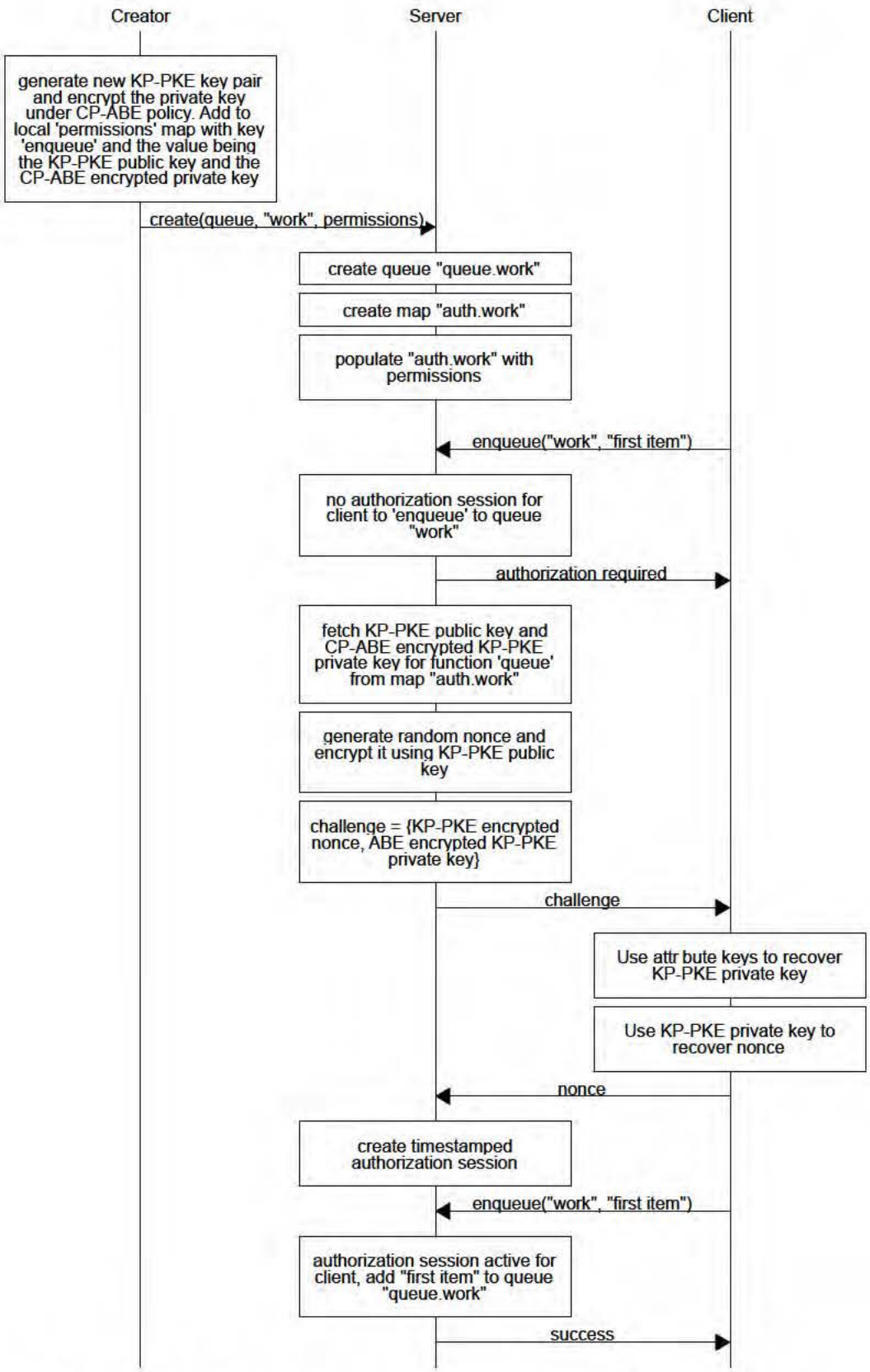


Figure 5.4: Example Session for Data Structure Creation and Authorization Protocols

The current implementation of Axon is built on top of Amazon's Web Services (AWS) public cloud infrastructure. Many cloud providers, including Amazon, provide services that support the data structures selected in Axon. For example, Amazon provides a service called Simple Notification System (SNS) that would support some aspects of the topic data structure. However, there are some drawbacks with using this service. First, users are limited to only 3,000 topics without additional authorization from Amazon. In our microblogging application, a unique topic is required for each tag and user. In order to provide this capability, Axon uses the Hazelcast project to provide distributed versions of each structure (map, queue, topic). Changes to these data structures are communicated to and from any connected clients via the SocketIO protocol as shown in Figure 5.3.

Since the Axon framework does not rely on the built-in cloud provider services for the data structures, it must still remain dependable even in when the entire processing cluster fails. Since Hazelcast is an in-memory distributed system, failure of the cluster would result in complete data loss. Fortunately, Hazelcast provides a persistance API for maps and queues. In Axon, this persistance API is connected to the database and storage services provided by Amazon. DynamoDB is a NoSQL database, and this is the primary destination for items inserted into maps and queues.

5.5 Applications

This section describes how the basic data structures in the Axon framework are layered to create a secure microblogging application. The first example demonstrates how to take the map data structure provided by Axon and enhance it to be compatible with the ACE-trusted security model. The next example demonstrates how to build a ACE-trusted remote procedure call protocol using Axon data structures. Finally, these relatively simple building blocks are used to build a more complex messaging application called Critter. This messaging application is compatible with the ACE-trusted security model, and could be used as a building block in even more complex applications.

5.5.1 Secure Map.

The secure map data structure interfaces with the regular map data structure provided by Axon and provides all of the same functions. The difference is that the secure map is initialized with a symmetric cryptographic key. The secure map uses this key in order to provide confidentiality and integrity guarantees to the data that passes through it. As a result, the secure map supports the ACE-trusted security model.

The secure map protects all three portions of a map: the map name, index keys[4], and values. When performing map functions, the secure map will first compute a MAC (using the cryptographic key provided to the secure map) of the plaintext map name to create a secure map name. This secure map name is then used to perform operations on a regular map data structure. When performing a *put* operation, the secure map must protect both an index key and a value. To protect the index key, a MAC is computed for the plaintext index key and used as the secure index key. Since the MAC function is deterministic, plaintext index keys will always map the same secure index keys as long as the cryptographic key is the same. Then both the index key and the value are each encrypted using AEAD, concatenated with a separating delimiter, and stored as the secure value. The reason that the index key is encrypted and stored alongside the encrypted value is to support index key enumeration operations. Since a MAC is used as the secure key, there is no way to "undo" the MAC operation to recover the plaintext key. Instead, the secure map performs a key enumeration on underlying map to get a set of secure key values. The secure map can than get each secure index key's respective secure value. Finally, the secure map can decrypt the secure index key portion of the secure value, thereby recovering the plaintext index keys. To perform *get* operations, the secure key is computed like before which returns the secure value. The secure map can decrypt the encrypted value portion of the secure value.

[4]In order to avoid confusion, we use the terms *cryptographic key* and *index key* to refer to keys used for encryption operations and map operations respectively

Through the use of a cryptographic key, the secure map is able to provide confidentiality and integrity guarantees. The availability of the map contents is supported through the Axon interface to the cloud infrastructure. This demonstrates a simple example of building an ACE-trusted data structure from one of the basic data structures provided by Axon. For an application using a secure map, the focus is shifted to acquiring the right set of cryptographic keys and not the mechanics of how the data is protected or stored.

5.5.2 Secure Remote Procedure Call.

In addition to creating secure data structures from the core data structures, it is also possible to build secure protocols. One example is a simple remote procedure call (RPC) protocol that leverages the queue data structure provided by Axon. The message sequence for this protocol is shown in Figure 5.5. Just like the secure map, the secure RPC protocol is initialized with a set of cryptographic keys. The keys allow the protocol to protect the data when sending and receiving data from the Axon queues. The queues then provide a layer of availability and durability. The current use case for the secure RPC mechanism is for a pool of remote servers that share a common KP-PKE key pair. Any of these remote servers is capable of fulfilling a request. Each local instance generates its own unique KP-PKE pair. The protocol supports *at least once* semantics and resubmits requests after a specified timeout period. In addition to the execute function, a broadcast function is available simply by replacing queues with topics. Permission to execute functions can leverage the same authentication protocol as the one described for data structure creation. Note that the remote server does not require private keys in order to authenticate a request, so instances of the remote server may run on the cloud infrastructure as long as the requested procedure call does not require access to plaintext data.

5.5.3 Micro-Messaging.

Now that we have described some of the building blocks, we now describe the structure of a relatively simple messaging application called Critter. Critter supports the

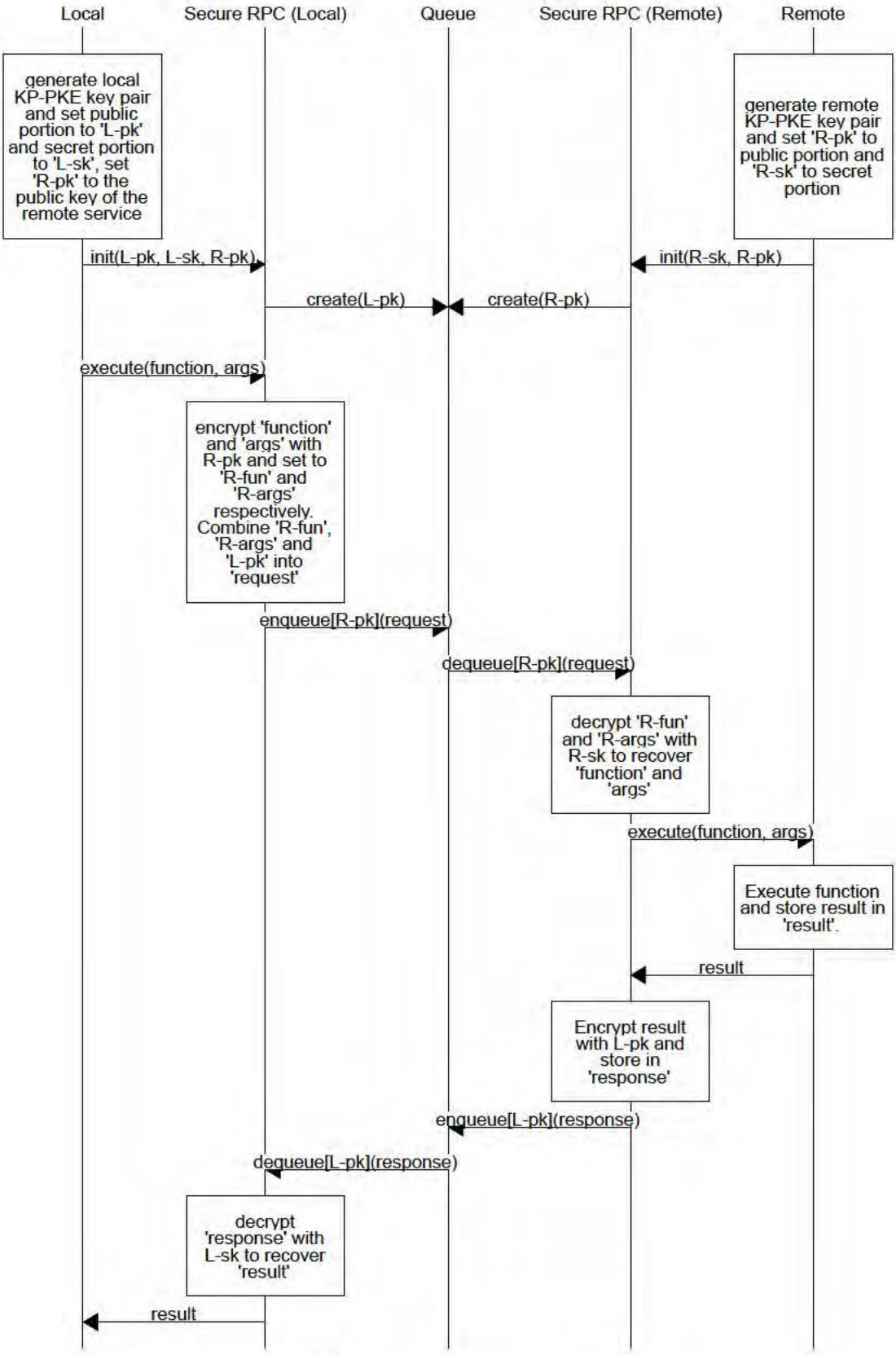


Figure 5.5: Secure RPC Message Sequence

basic messaging capabilities of popular microblogging applications such as Twitter and Instagram. Each user has a public feed to which followers can subscribe. Users may tag messages by including the hashtag symbol # followed by a keyword of the user's choice. We refer to these hashtag keywords as *topics*. Users may then follow these tags for topics which interest them. Users may also direct messages to only be sent to specific users rather than their public feed by including the target users' handle preceded by the @ symbol. We'll re

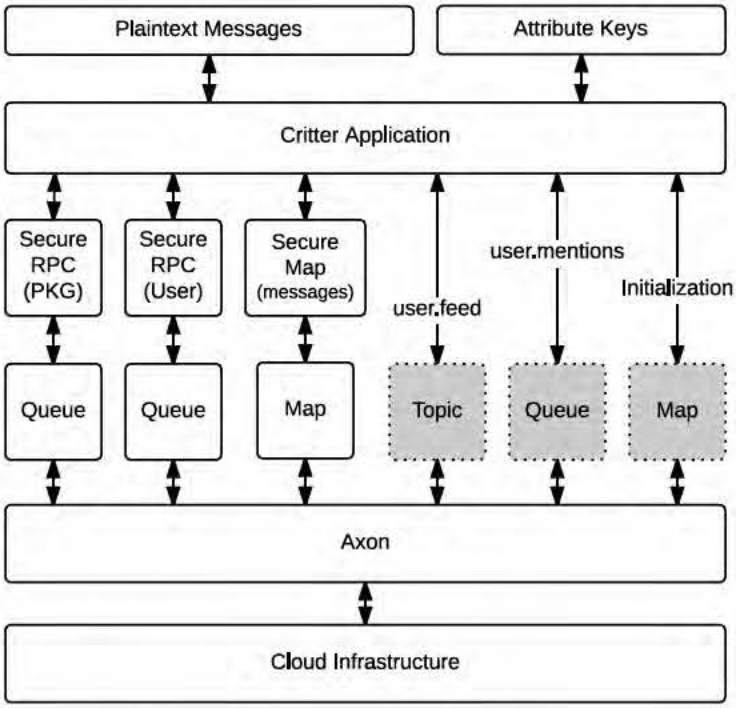


Figure 5.6: Critter Information Flow

The structure of the Critter application is shown in Figure 5.6. The highlighted portions indicate where Critter uses Axon data structures directly rather than relying on secure components. When the critter application is first started and before any messages are sent, a small set of public data is stored in an unsecure map. This map contains the

information needed to bootstrap the security for the rest of the system and consists of the following:

- A public random value used when creating private keys from passwords (also known as a salt)
- The public parameters for the CP-ABE
- A private key that has been encrypted using a CP-ABE policy that requires the "CritterUser" attribute

Whenever a new user is created, that user receives an the "CritterUser" attribute from the private key generator. This process is assumed to happen out-of-band. When a user logs into the system, the user can use this attribute to recover the CP-ABE protected private key stored in the public map. This key is used in a few places within the application. Specifically it is used to access the secure map where the messages are stored.

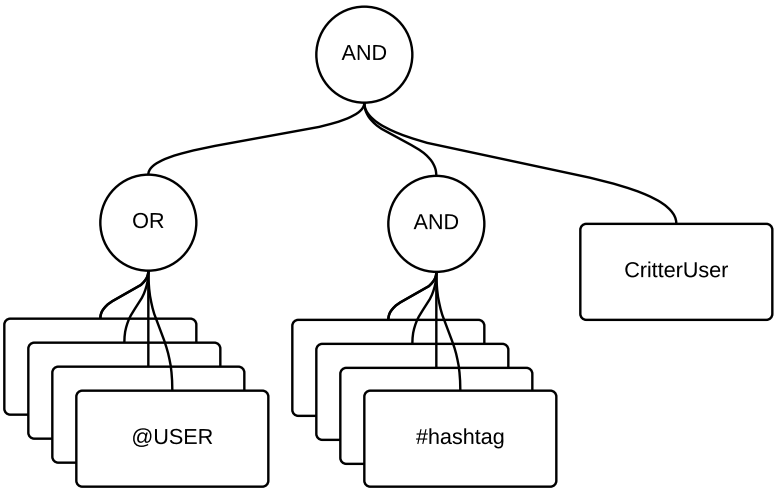


Figure 5.7: General Critter Message Policy Structure

(messa

======

@bob \

climat

#huma

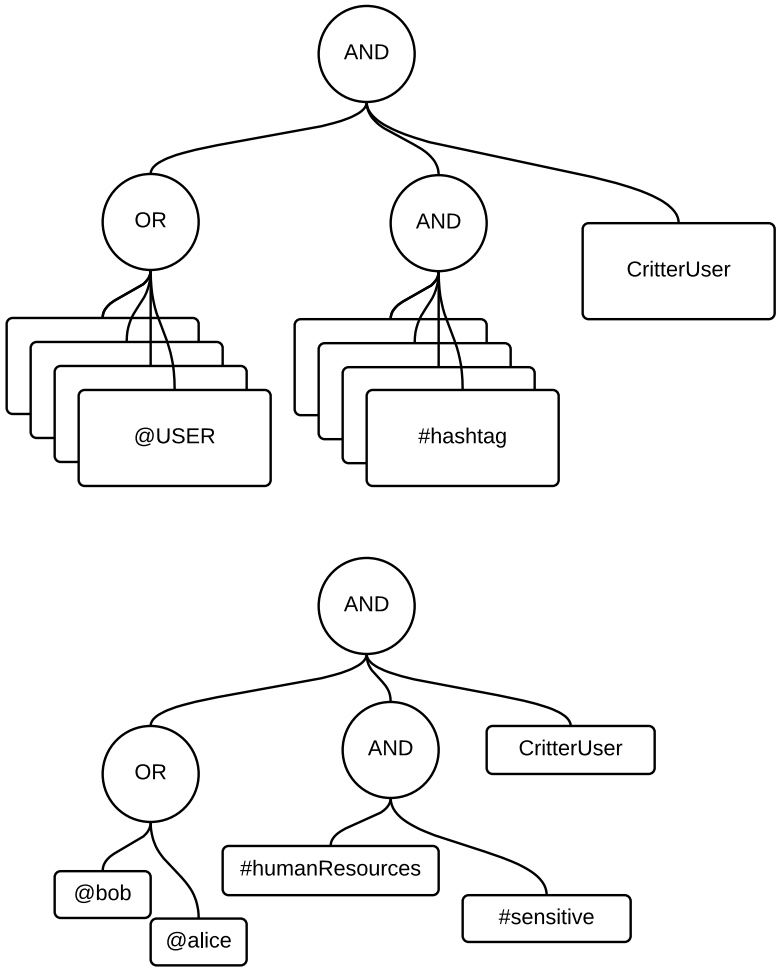


Figure 5.8: Example message mapped to security policy

When a user sends a message, the application first parses the message to determine if it contains any topics or if the message is directed to specific users. If the message has any mentions, the message is considered to be *private* and the relevant users are considered *mentioned* in the message. If there are no mentions, the message is *public*. If the message contains topics, then the message is *restricted.* Otherwise the message is *unrestricted*. Critter uses this information to build the MA-AHASBE policy based on the general structure shown in Figure 5.7. If the message is private, all of the user handles, including that of the sender, are combined in under an *OR* node in the policy. If the message is restricted, then the topics are combined under an *AND* node. Finally, all messages include the "CritterUser" attribute. This attribute is particularly useful for unrestricted,

public messages. In this case, the "CritterUser" attribute is the only one in the policy and is used to provide a baseline policy to protect the information from the CSP. The policy enforces the condition that in order to decrypt, a user must possess as attributes:

- at least one of the handles used in the message (if any are used)
- all of the topics in the message (if any are used)
- the "CritterUser" attribute

This policy forces users to have all of the tagged topics as attributes in order to decrypt a message. In Critter, this is a way to control who has access to certain topics of conversation.

Once the policy is built, Critter uses MA-AHASBE to create a corresponding ciphertext. A universally unique identifier (UUID) is generated for the ciphertext and the encrypted message is stored in a secure message map with the UUID as the index key. If the message is public, the message UUID is then posted to the user's feed, which is an Axon topic. Any followers of this user will then receive the UUID and can then retrieve the message from the secure map. If the message is private, the UUID is instead sent to an Axon queue for each user mentioned in the message.

Critter uses the secure RPC mechanism described in the previous section in order to request and receive attribute keys from the private key generator. Whenever a user receives a message with topics it does not have attributes for, it issues a key request to the private key generator. The private key generator determines if the user is allowed to access messages with the request topic. If so, the PKG generates a key and sends it back to the user. If not, a message is sent denying the request.

Consider the example shown in Figure 5.8. In this example, a user Alice with the handle `@alice` publishes the message "`@bob What's the status of today's climate assessment? #sensitive #humanResources`". This is a private, restricted message. Critter builds the policy shown in Figure 5.8, generates and stores the ciphertext

in the secure message map with a fresh UUID as the key, and finally adds the message to a queue named "`@bob.mentions`". Whenever Bob comes online, he can check his queue and retrieve the message. If he can satisfy the policy, he can then recover the message. If not, he will send a request to the PKG for any missing attributes. Suppose Alice instead wanted to send the message to anyone following her feed. In this case, Alice would remove the mention "`@bob`" from the message. The resulting policy would then not contain any mentions. The UUID would be sent to Alice's feed topic "`@alice.feed`". Any followers would then receive the UUID and attempt the same decryption process as previously described with Bob.

5.5.4 Experimental Results.

In this section, we examine the performance characteristics of Axon and Critter. The performance can be be viewed from two perspectives. First, we look at the performance impact to the user in terms of encryption and decryption time. Second we look at the impact from the server perspective, specifically in the requirement to support the larger payloads associated with encrypting the data using MA-AHASBE. In order to do this, we analyzed a sample set of 450,000 messages from the Twitter message service. All experiments were performed on machines running Windows 7 with Intel Xeon X5675 6-core processors and 32 GB of RAM connected on a local area network.

As described in the previous section, the number of topic tags and mention tags determines the size of the access policy for a particular message. The size of the access policy, in turn, influences the size of the ciphertext created by MA-AHASBE. The sample database was analyzed to determine how many topic and mention tags occur in a message. Some summary information is presented in Table 5.2. The table shows the number of topics and mentions in terms of percentiles. For example, 99% of all the messages in the sample contain five topic tags or less. The largest number of topics in any single tweet was 17. For

mention tags, 99% of messages have six or less while the largest for a single message was 14. The final row shows the counts when both types of tags are combined.

Table 5.2: Summary Statistics

Category	*50th percentile*	*95th percentile*	*99th percentile*	*Max*
Topics	0	2	5	17
Mentions	1	2	6	14
Combined	1	4	7	17

Another way to look at how the tags occur within a message is shown in Figures 5.9 and 5.10. These figures shown the percentiles of how many tags occur with respect to one another. In both figures, we see that there is a substantial drop between the maximum count relative to the 99% percentile case. This is particularly true when there are either zero mention or zero topic tags. When there are zero topic tags, the largest observed number of mention tags was 14, but 99% of messages had five mentions or less. When there are zero mention tags, the largest observed number of topic tags was 17, but 99% of messages had six mentions or less. The data also reflects the fixed 140 character limit of Twitter messages. As the number of topic tags increase in a message, the number of mentions decrease.

The number and type of tags in a message influence the amount of time to perform encryption and decryption operations. For encryption operations, the performance scales with the total number of tags. The performance of the encryption operation is shown in Figure 5.11. The format of the figure follows the format of Figure 5.9 where each line represents the percentile of mentions per topic. As shown in Table 5.2, 99% of messages have five topics or less. In Figure 5.11 we see that for messages with five topics, encryption

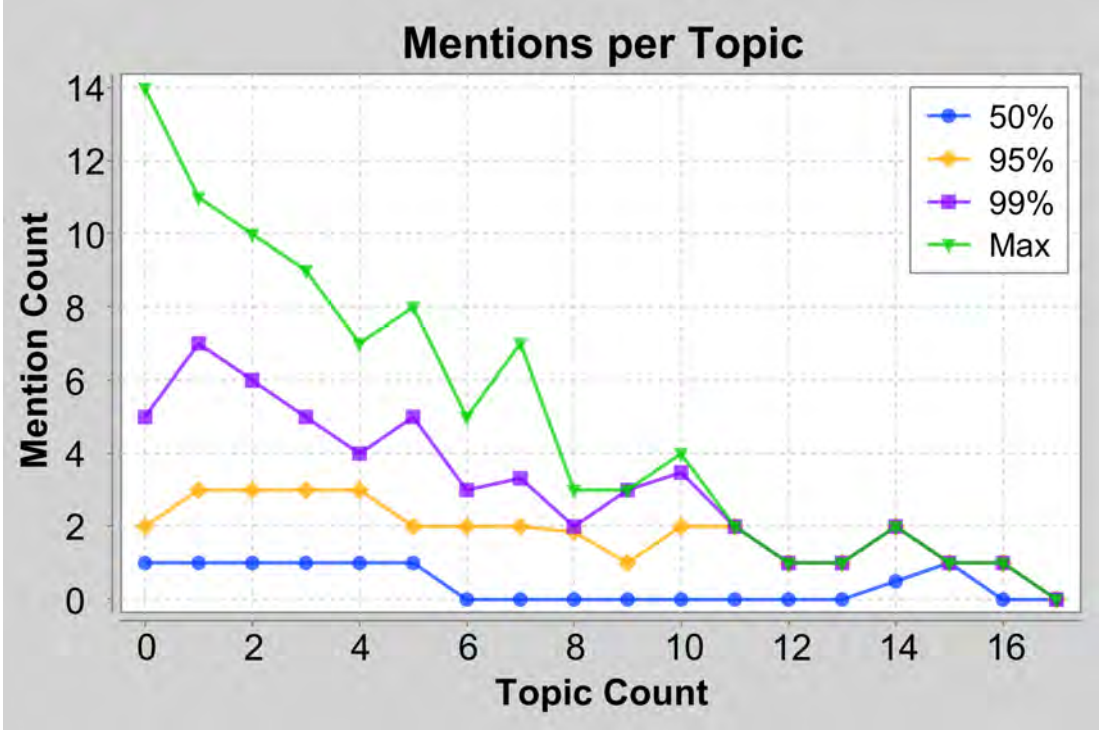


Figure 5.9: The number of mentions in a tweet per the number of tags based on percentiles. For example, of all the sample tweets with exactly two topic tags, 95% have three mention tags or less and 99% have six mention tags or less. All sample tweets with two topic tags had ten mentions or less.

time time is typically at most around 85 milliseconds. As the number of topics increase towards the maximum, the lines converge since there are fewer mention tags contributing to the total. The worst case encryption times are under 140 milliseconds. The decryption timings are shown in Figure 5.12. There is a clear linear scaling with the number of topics, but there is not much variation with the number of mentions. Recall that the security policy is built by combining the mention tags under an OR gate and the topic tags under an AND gate (see Figure 5.7). The decryption time scales with the number of nodes required for decryption, so even though there may be several mention tags only one is required for decryption. It is also of note that the decryption time is an order of magnitude larger for

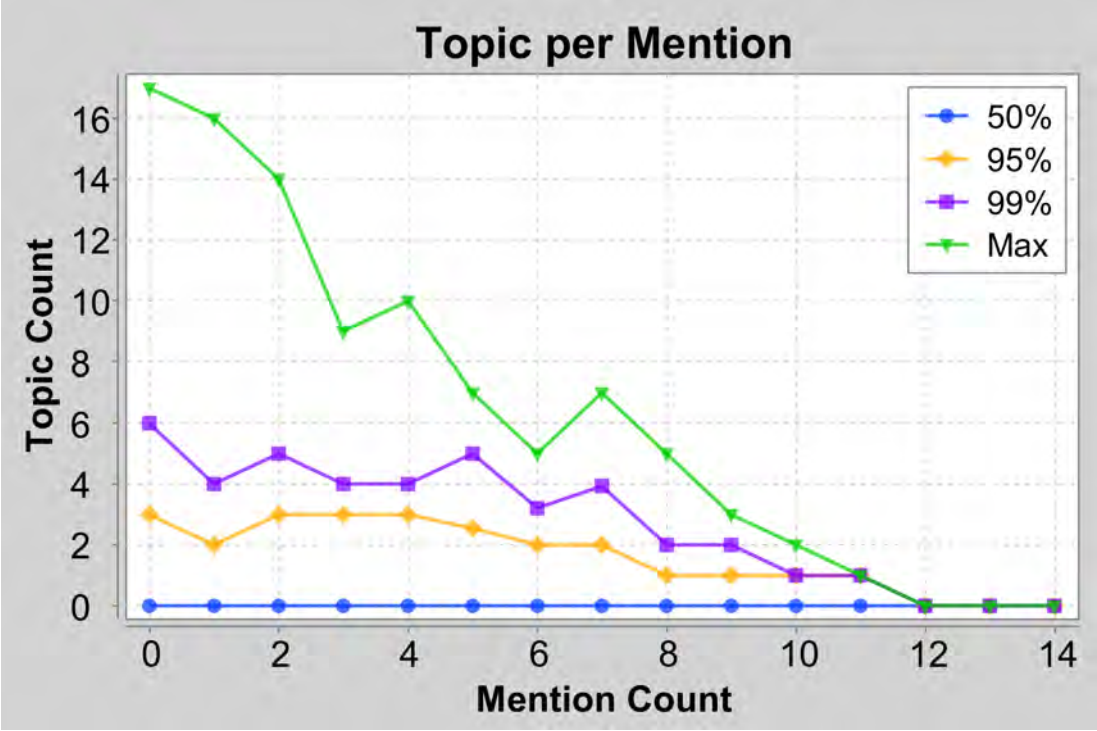


Figure 5.10: The number of topic tags in a tweet per the number of mentions based on percentiles. For example, of all the sample tweets with exactly two mention tags, 95% have three mention tags or less and 99% have five mentions or less. All sample tweets with two mentions had 14 topic tags or less.

decryption versus encryption. This is due to an expensive mathematical operation called a bilinear pairing that is required for decryption. In the worst case, it takes just over 1.2 seconds to decrypt a policy with 17 topic tags. However, since 99% of messages have five topics or less, most messages can be decrypted in around 500 milliseconds.

From the server perspective, the biggest impact on performance is the size of the payload. A Twitter message represents a type of worst case in terms of MA-AHASBE overhead. First, the access policy is potentially different for each message and so it must be

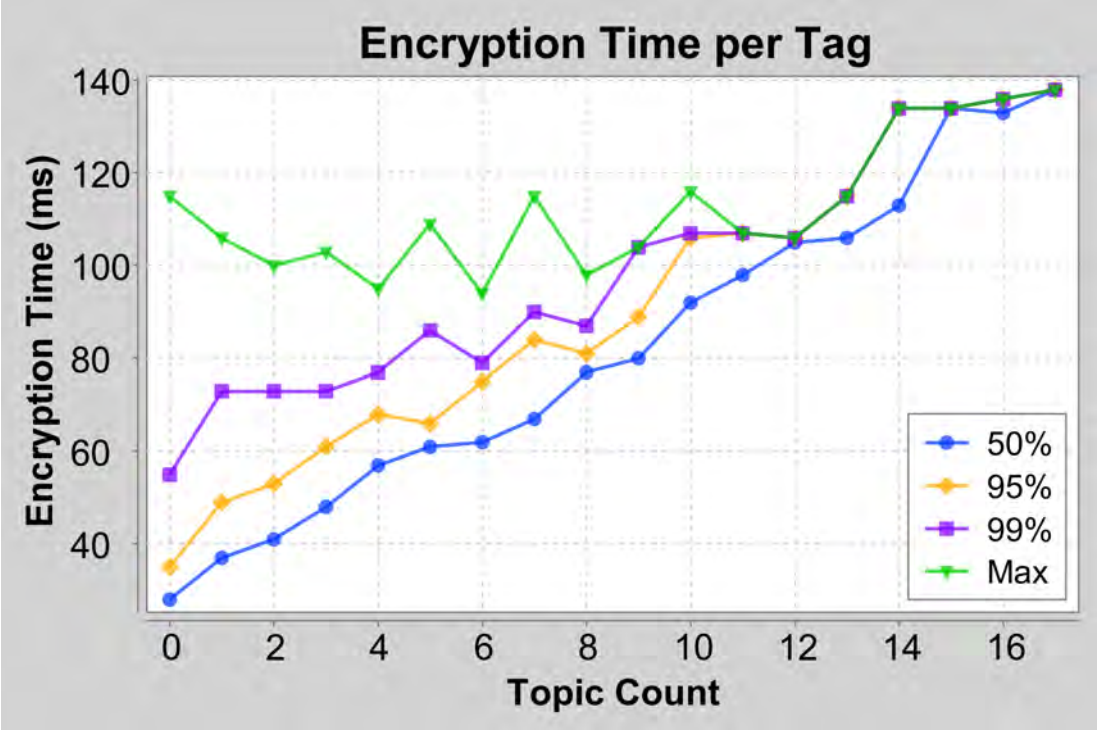


Figure 5.11: The amount of time to encrypt a tweet for a given number of topics and the percentiles of mentions.

included. Second, since the size of the plaintext message is so small, the relative overhead of MA-AHASBE is significant. The ciphertext sizes are shown in Figure 5.13. Again, for 99% of messages the ciphertext size is under 3,500 bytes, but can be as large as 5,200 bytes in the worst case. To see the effect on server performance, several client machines are used to saturate the system by sending as many messages of a fixed size as possible in a 60 second window. Figure 5.14 shows the effects of increasing the number of servers as well as increasing the size of the payload. Each box plot is the result of five samples. As expected, increasing the number of servers increases the throughput while increasing payload size decreases throughput. Based on the data from Figure 5.13, we see that most messages will fall under 3,500 bytes. While this has a modest impact on performance,

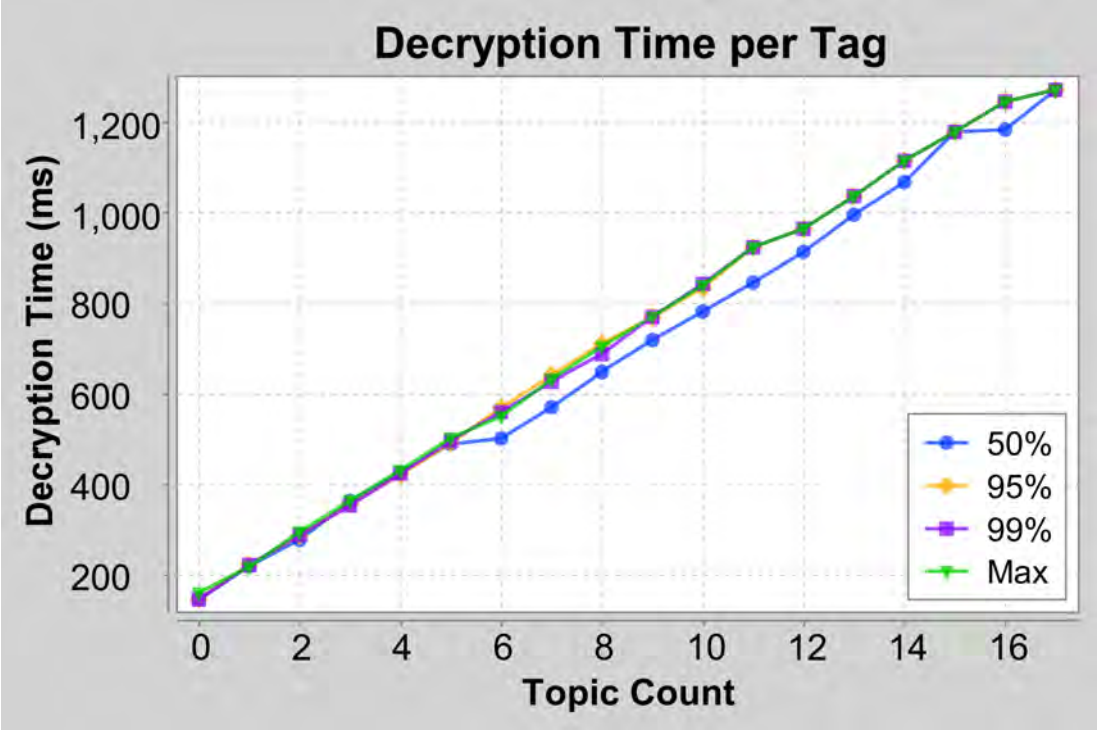


Figure 5.12: The amount of time to decrypt a tweet for a given number of topics and the percentiles of mentions.

we believe there are situations where the security capabilities of a system such as Critter outweigh the performance impact.

5.6 Conclusions and Future Work

In this chapter, we have proposed a security model that reduces the trust required in a CSP to availability operations. To make development of applications easier in this restricted model, we developed a framework called Axon that uses a small set of data structures: maps, queues, topics. Axon uses the cloud to provided availability and durability for these data structures. On the client, additional secure data structures and protocols are created using a layered approach. Finally, a secure messaging application Critter was built using the Axon framework.

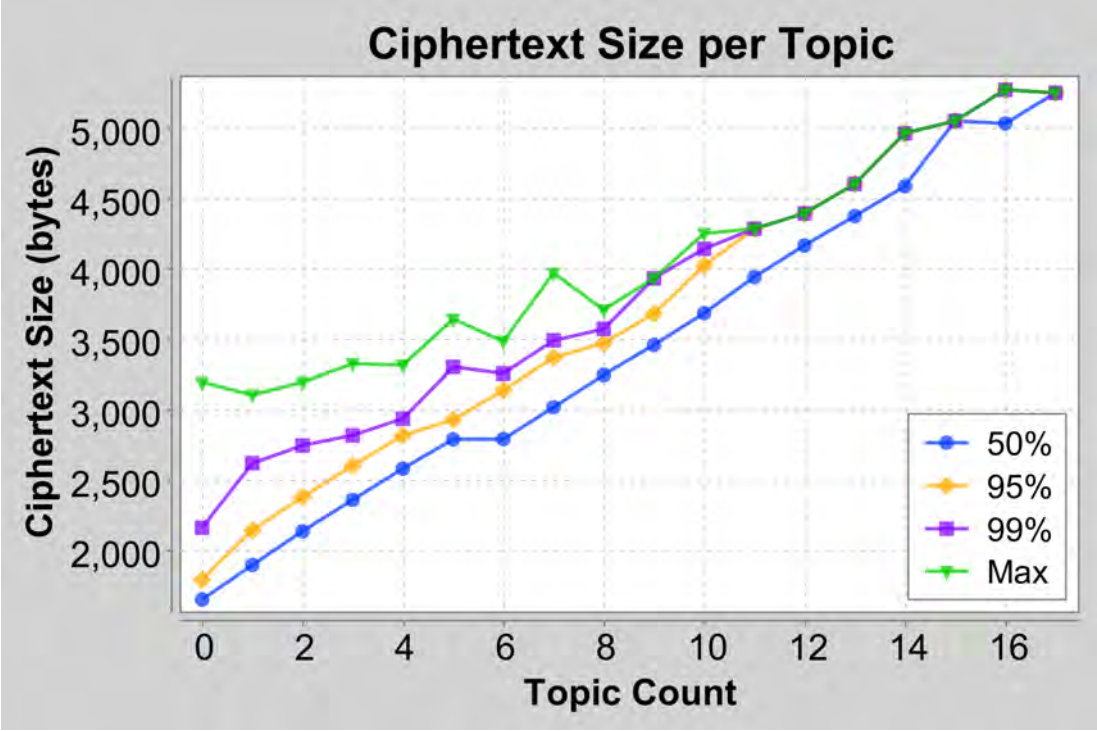


Figure 5.13: The size of ciphertext for a given number of topics and the percentiles of mentions.

An aspect of dependability that isn't currently addressed by the Axon framework is being able to function even when the connection to the CSP is temporarily lost. One way to improve this aspect of dependability in this area is to cache as much data locally on the clients as possible. This benefits performance in many ways. First, if the connection to the CSP is lost, access to at least a portion of the relevant data may still be accessible. Meaningful computations may still be accomplished while the connection is down. Secondly, caching data on the client increases the availability of the data if the cloud version becomes corrupted. Finally, there is a performance advantage as the data is quicker to access since the client does not need to make costly network calls to the cloud in order to access the data it needs. An important negative consequence of this approach is it increases the storage burden on the client. However, many client machines are desktops

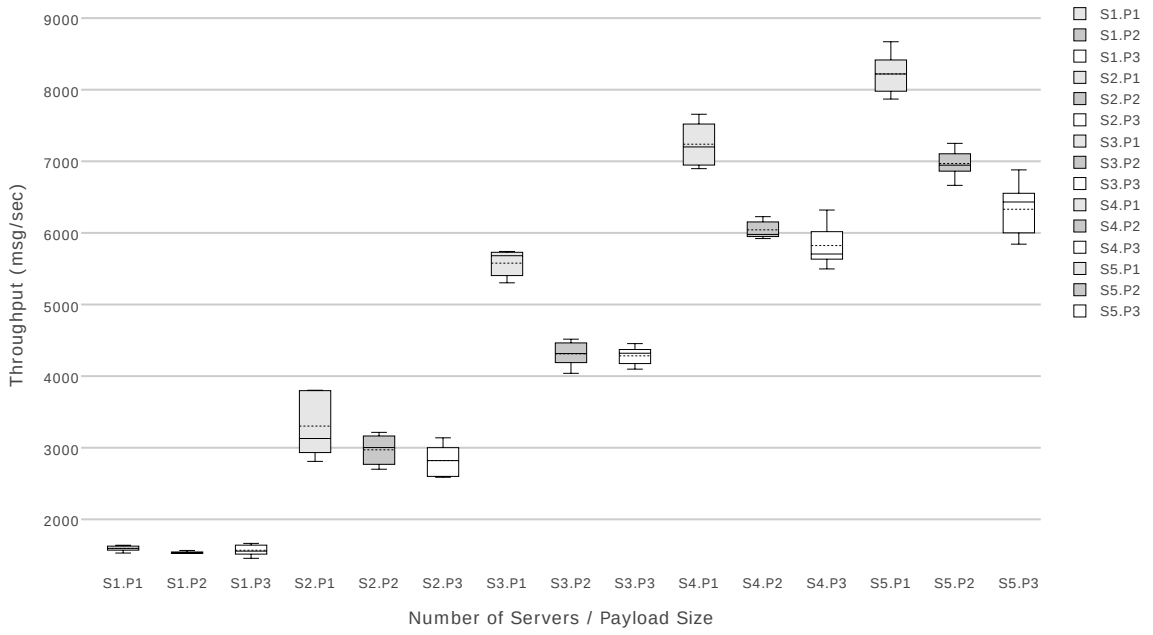


Figure 5.14: Throughput Performance per Number of Servers and Payload Size. The dashed lines represent the sample mean and the solid line indicates the median. The *S* label indicates the number of servers and the *P* label indicates the payload size. P1=140 bytes, P2=3,500 bytes, P3=5,000 bytes.

where local storage is cheap. Even mobile clients can have significant amounts of locally available storage. For mobile clients, calls to local storage will also typically be more power efficient than requests that must go over the network. One approach would be to incorporate conflict-free replicated data types (CDRT [34]) within Axon.

Another improvement we wish to make is to model the protocols in Axon using a formal modeling language such as SPIN [57] or TLA+[68]. For small applications like Critter, a formal model may not be necessary. However, as applications increase in size and complexity, a formal model can help ensure that all data is processed through a security mechanism (such as a secure data structure or protocol) before being sent to a core, unprotected data structure.

Finally, we would like to extend the Axon prototype to include different cloud implementation designs as well as to build a testing suite in order to evaluate the performance of each implementation. The current design relies on Hazelcast, but we would like to use some of the other popular messaging frameworks. Since the client side portion of Axon is insulated from the actual cloud side implementation, the same application code could be used for each cloud implementation.

VI. Summary of Contributions and Further Research

This chapter concludes this document by presenting the contributions of this research as well as providing some thoughts on the direction of future work. The cornerstone contribution of this research is the encryption scheme MA-AHASBE. Building on this cornerstone, the other contribution of this research is the demonstration of how MA-AHASBE can be used to build applications that can leverage public cloud computing resources. Future research efforts can build on this framework, expanding the types of services that can be provided and improving the computational cost for doing so.

6.1 Contributions

The first significant contribution of this research is the development and implementation of MA-AHASBE. The MA-AHASBE system presents several significant developments in attribute-based encryption research. There are many ABE systems that individually have each of the characteristics available with MA-AHASBE. However, to the best of my knowledge, MA-AHASBE is the only system that incorporates all of these in a single system. Attribute-sets have been shown to be a useful and practical extension of ABE [17]. MA-AHASBE is the only known multi-authority extension of ASBE. In fact, the only known published work to extend ASBE is the HASBE system which provides a type of hierarchical extension. The hierarchical mechanism in HASBE does not provide for hierarchical autonomy, which as argued in Chapter 4 is a requirement for an efficient implementation of ABAC with an ABE. Aside from the theoretical contributions of MA-AHASBE, the implementation of MA-AHASBE (described in Appendix B) stands on its own as a novel contribution. The implementation shows that implementing MA-AHASBE can be efficient and provides insight into the cost of policy size, number of users in the system, and the penalty involved with protecting attributes.

The second significant contribution of this research is the application of MA-AHASBE to the area of microblogging. Applying MA-AHASBE in this way brought about other contributions. Two new security models were developed, A-trusted and ACE-trusted, with ACE-trusted being the model most applicable to public cloud computing. These models focus on availability as the key trust requirement between a client and a cloud service provider. The availability requirements are defined in detail through the F-SCRAWL operations, a novel formulation. The A-trusted model only has the availability trust, while the ACE-trusted model adds certain computability and access control enforcement requirements. While this does increase the trust required in a CSP, the efficiency gains can be significant and the tradeoff in security seems reasonable for reputable CSPs. Neither model requires confidentiality trust, so CSPs cannot learn the contents of data. In order to support this model, the Axon framework is presented. This framework demonstrates a layered, data structure oriented approach to building complex, secure data structures and protocols. This method is used to build Critter, a secure microblogging application. Performance results indicate that while there is a cost associated with enforcing an ACE-trusted model in Critter versus processing the data in plaintext, the cost is not unreasonable and the benefits in security can be significant. This is particularly important if the messages being shared in Critter are sensitive or require complex access control policies.

6.2 Future Work

This work will hopefully new doors and new ways of thinking about how cloud services can be secured. Already there are some tangential efforts at AFIT that build on the work presented in this research. One effort is looking at techniques to speed up pairing computations using graphics hardware. Research already exists that accomplishes this with composite order groups [110], but in order to work with MA-AHASBE the techniques would need to be adopted for prime order groups and Type-3 pairings. If

successful, this could improve the efficiency and scale of a MA-AHASBE implementation. Also, similar techniques used in MA-AHASBE for combining ASBE and AHIBE could be used with the system described in [90] which uses only standard assumptions, instead of the non-standard assumptions used in MA-AHASBE. This extra boost in security confidence comes at a potentially significant computational cost. This is because what where single term multiplications in MA-AHASBE, now become vector multiplications in [90]. The capability to speed up pairing computations using graphics hardware could help mitigate this extra computational cost. Cloud resources that include very powerful graphics hardware are becoming more commonplace from their use in scientific computing applications. So the improvement of pairing computations using graphics hardware fits seamlessly in the secure cloud computing paradigm.

Another interesting application of this approach is in software version control. Current research at AFIT is looking at ways to secure the popular version control software *git*. Similar to the application in Chapter 5, this approach would mesh well with MA-AHASBE. Specifically, MA-AHASBE could be used to provide fine-grained attribute based control to particular files or folders for both read and write access. Also, it is possible that the concepts used in securing git could be extended and applied to a fully version controlled, secure file system. Finally, MA-AHASBE can be used with many cryptographic systems which are based on symmetric keys such as encrypted search and group collaboration (additional notes on these application areas are provided in Appendix C).

6.3 Conclusion

The end result of the contributions presented in this research is to support the thesis statement stated in Chapter 1:

> *Efficient techniques exist to support the secure use of public cloud computing resources by a large, federated enterprise.*

MA-AHASBE combines with a hybrid cloud computing model where the public cloud service providers operated under an ACE-trusted model of security. This forms the backbone of key management that is designed to support an ABAC paradigm in a large, federated enterprise. The application focused contributions support the claim that this approach can be successfully applied to tasks such as microblogging. The future work highlights areas where additional services such as search, real-time collaboration and version control can be added. It also may be possible to improve the efficiency of the current approach through the use of graphics hardware. Overall, the results of this type of research will hopefully contribute to a new approach to securing sensitive information while being able to take advantage of the ever decreasing costs of cloud computing.

Appendix A: MA-AHASBE Security Proof Details

A.1 Notation and Preliminaries

This proof describes group operations assuming they are written multiplicatively, even though we write some groups such as $\mathbb{G}_1$ additively. The concepts apply regardless of the notation for the groups. For example, the variable s in the term sP_1 is referred to as being "in the exponent" of P_1. The proof demonstrates security under chosen plaintext attack or CPA. The CPA construction is made by reversing the CCA transform used in the construction given in §4.4 by applying the following changes:

- Remove $CT_{\mathbf{com}}$ and $CT_{\mathbf{tag}}$ from the ciphertext $\mathcal{CT}$
- Remove C_{σ} from $\mathcal{CT}$
- Replace k' in the global leaf nodes with the message $m \in \mathbb{G}_T$

This modified encryption algorithm is referred to as $\mathbf{Encrypt}_{\mathsf{CPA}}$. Since $CT_{\mathbf{com}}$ and $CT_{\mathbf{tag}}$ are no longer in the ciphertext, there is no need to perform the checks to make sure they are consistent. Also, the new message recovery algorithm for m becomes the old recovery algorithm for k'. Ciphertexts created using $\mathbf{Encrypt}_{\mathsf{CPA}}$ are labeled $\mathcal{CT}_{\mathsf{CPA}}$. For this proof, the ciphertexts do not include the plaintext policies as this would make it trivial for the adversary to distinguish the message-policy pairs. The adversary has access to the policies since they are included in the adversary's submission, they are just not returned in the encryption.

For ease of presentation, the instantations below show the simulator choosing the global parameters. In particular, the simulator knows the global secret key GSK. This allows the simulator to execute the user key generation algorithm that requires the user identifier requested by the adversary. However, this also allows the simulator to facilitate collusion as described in §4.5.2.3, since it is in effect acting as the trusted central authority.

In order to achieve the security required by the security game in §4.5.1, the simulator cannot know these values. This can be achieved using a secure computation protocol, and the method chosen affects the overall set of security assumptions. The simulator and adversary can participate in a two-party secure computation protocol (as described in §4.3.2.1) to compute the values according to the algorithms in global setup. Note that this can be done adaptively as the adversary is requesting keys. In this approach, neither the simulator nor the adversary know the components of GSK, but merely have access to the global parameters which includes the user identifier parameters. However, any cryptographic assumptions made by the two-party secure computation protocol (for which there are standard model schemes such as [23]) must be added to the assumptions presented in §4.5.3.

Each lemma presented here follows the same pattern. First the cryptographic assumption is presented. Then the simulator instantiates the global and local authority parameters using the terms in the assumption. The setup is followed by the key extraction phases. Since there is no computational difference between Phase 1 and Phase 2 as described in MA-AHASBE security game, these phases are presented together. Also, the computations are only shown for domain key extractions and not for user key extractions. The adversary may request a key of either type. If the adversary requests a domain key, the simulator follows the procedures described in the proof and returns the result. If the adversary requests a user key, the simulator first creates the corresponding domain key using the proof procedure, and then uses the unmodified algorithm **UserKeyGeneration** to create the user key. Recall that a user key inherits the semi-functional properties (normal, full, partial) of the domain key that creates it. The unique portion of either type of key extraction query is the domain key generation and so that is all that is presented.

Next, the adversary submits two message-policy pairs for the challenge phase. Recall that the two policies must be structurally equivalent. In MA-AHASBE, the q terms in the

ciphertext represent the splitting of the secret s according to the policy. Since s is not typically directly available to the simulator, the computation for each q term must now be done in the exponent. The terms in the polynomial can be represented by $g^{a_n x^p}$ (where g is a generator and a_n is the randomly chosen coefficient) with the addition of terms by $g^{a_0} g^{a_1 x} g^{a_2 x^2} g^{a_3 x^3} \ldots = g^{a_0 + a_1 x + a_2 x^2 + a_3 x^3 \ldots}$. If the generator g and the initial term g^s are given, then the remainder of the $g^{q_n(0)}$ terms can be calculated using the same method described in the $\textbf{Encrypt}_{\mathsf{CPA}}$ algorithm in the exponent rather than directly. Therefore, in the proofs we give the computations for the initial values containing g^s. The computation for the relevant generator g is nearly identical to what is shown for g^s, only dropping the s in each term where it appears. In each lemma, the relevant terms without s values are present in the assumption. The simulator can then calculate the relevant $g^{q_n(0)}$ values according to the policy. The result remains in the exponent since this is what the ciphertext requires and since the simulator cannot compute the $q_n(0)$ values directly.

In each computational section, the actions of the simulator are presented in the same order. First the simulator chooses some set of parameters randomly. The simulator then explicitly sets some set of parameters. These are the parameters that the simulator can explicitly compute and usually represent components that will be returned to the adversary. Because of the explicit computations, some parameters are implicitly set. Usually the simulator cannot directly compute these parameters, but they are useful in understanding the limitations of the simulator for the given lemma. Finally, each lemma concludes with the adversary submitting a guess for which message-policy was chosen. A short description is then given that explains how this guess corresponds to the security games presented in Section 5. Note that at any time during the key extraction phases, the simulator tracks the requests and can determine if the adversary has requested keys that allow for a trivial decryption. If this is the case, the simulator aborts. The simulator can also detect if the

policies submitted for the challenge are not structurally equivalent and also aborts in this case.

The following shorthand will be used throughout the proofs where **id** represents a domain, **attr** represents an attribute, and $y_{i,j}, u_i, \lambda_{i,j}, \nu_i$ represent parameters chosen by the simulator during the proof:

$\mathbf{id} = \langle \mathsf{id}_1, ..., \mathsf{id}_\ell \rangle$

$\mathbf{attr} = \langle \mathsf{attr}_1, ..., \mathsf{attr}_\ell \rangle$

$h_i(\mathbf{id}) = \sum_{j=1}^{\ell} y_{i,j} \mathsf{id}_j + u_i$

$g_i(\mathbf{id}) = \sum_{j=1}^{\ell} \lambda_{i,j} \mathsf{id}_j + \nu_i$

A.2 Lemma 4.1.

$|Pr[X_{real}] - Pr[X_0]| \leq \epsilon_{LW1}$

A.2.1 Assumption.

LW1 $= (F_1, bsF_1, sF_1, aF_1, ab^2F_1, bF_1, b^2F_1, asF_1, b^2sF_1, b^3F_1, b^3sF_1, F_2, bF_2, Z_1)$

Let $Z_1 = (ab^2s + \gamma)F_1$. It is hard to decide whether $\gamma = 0$ (Z_1 is real) or $\gamma \in_U \mathbb{Z}_p$ (Z_1 is random).

A.2.2 Setup(λ, n_{auth}).

$\mathscr{B}$ receives an instance of LW1 and chooses:

$p, e, \mathcal{H}, h$ based on λ

$\alpha_g, \beta_{id}, \{\beta_j\}_{j\in[0,r_{max}]}, y, \{\alpha_i, k_i, o_i, v'_i, \{y_{i,j}, \lambda_{i,j}\}_{j\in[1,h_{max}]}, u_i, \nu_i\}_{i\in[1,n_{auth}]} \xleftarrow{U} \mathbb{Z}_p^*$

$Q_{id}, U_{id} \xleftarrow{U} \mathbb{G}_2$

$\mathscr{B}$ explicitly sets $\forall i \in [1, n_{auth}]$:

$P_1 = b^2F_1 + yF_1$

$a_iP_1 = k_i((ab^2F_1) + y(aF_1))$

$\tau_iP_1 = o_i(b^3F_1) + k_iv'_i(ab^2F_1) + o_iy(bF_1) + k_iv'_iy(aF_1)$

$Q_{i,1,j} = \{\lambda_{i,j} o_i^2 (b^2 F_1) + y_{i,j} F_1\}_{j \in [1, h_{max}]}$

$a_i Q_{i,1,j} = \{k_i (\lambda_{i,j} o_i^2 (ab^2 F_1) + y_{i,j} (aF_1))\}_{j \in [1, h_{max}]}$

$\tau_i Q_{i,1,j} = \{\lambda_{i,j} o_i^3 (b^3 F_1) + k_i v_i' \lambda_{i,j} o_i^2 (ab^2 F_1) + o_i y_{i,j} (bF_1) + k_i v_i' y_{i,j} (aF_1))\}_{j \in [1, h_{max}]}$

$U_{i,1} = \nu_i o_i^2 (b^2 F_1) + u_i F_1$

$a_i U_{i,1} = k_i (\nu_i o_i^2 (ab^2 F_1) + u_i (aF_1))$

$\tau_i U_{i,1} = o_i \nu_i o_i^3 (b^3 F_1) + k_i v_i' \nu_i o_i^2 (ab^2 F_1) + o_i u_i (bF_1) + k_i v_i' u_i (aF_1)$

$V_{i,2} = o_i (bF_2)$

$V_{i,2}' = v_i' F_2$

$e(P_1, P_{i,2})^{\alpha_i} = (e(o_i^2 (b^2 F_1) + b^2 F_1 + yF_1, yF_2) e(o_i^2 (b^3 F_1), bF_2))^{\alpha_i}$

GSK: $(\frac{1}{\beta 0}, \frac{-\alpha_g}{\beta 0}, \frac{\alpha_g}{\beta_{id}}, Q_{id}, U_{id})$

$\mathscr{B}$ implicitly sets $\forall i \in [1, n_{auth}]$:

$a_i = k_i a$

$v_i = b_i = o_i b$

$\tau_i = o_i b + k_i a v_i' = b_i + a_i v_i' = v_i + a_i v_i'$

$P_{i,2} = (b_i^2 + y) F_2$

$Q_{i,2,j} = \{\lambda_{i,j} o_i^2 (b^2 F_2) + y_{i,j} F_2\}_{j \in [1, h_{max}]}$

$U_{i,2} = \nu_i o_i^2 (b^2 F_2) + u_i F_2$

$\mathscr{B}$ publishes for all players including $\mathscr{A}$:

GP: $(p, e, \mathcal{H}, h, P_1, F_2, \beta_{id} P_1, \frac{-\alpha_g}{\beta_0} F_2, \{\beta_j P_1, \frac{1}{\beta_j} F_2\}_{j \in [0, r_{max}]}, e(P_1, F_2)^{\alpha_g})$

AP_i: $\{a_i P_1, \tau_i P_1, U_{i,1}, a_i U_{i,1}, \tau_i U_{i,1}, \{Q_{i,1,j}, a_i Q_{i,1,j}, \tau_i Q_{i,1,j}\}_{j \in [1, h_{max}]}, e(P_1, P_{i,2})^{\alpha_i}\}_{i \in [1, n_{auth}]}$

A.2.3 Phases 1 and 2.

$\mathscr{A}$ makes q key extract queries. $\mathscr{B}$ does not know $\{P_{i,2}, Q_{i,2,j}, U_{i,2}\}_{i \in [1, n_{auth}]; j \in [1, h_{max}]}$ which are the master secrets for each authority.

$\mathscr{B}$ receives a key query for a domain with $\mathbf{id} = (\mathsf{id}_1, ..., \mathsf{id}_\ell)$ under authority i where $i \in [1, n_{auth}]$ and chooses:

$r'_1, r'_2, r'_3, r'_4, (z'_{1,j}, z'_{2,j})_{j \in [\ell+1, h']}, w_1, w_2 \xleftarrow{U} \mathbb{Z}_p^*$

$\mathscr{B}$ explicitly sets $\forall\, j \in [\ell + 1, h']$:

$K_{1,1} = w_1 y F_2 + r'_1 o_i (bF_2)$

$K_{1,2} = v'_i K_{1,3}$

$K_{1,3} = r'_1 F_2 - w_1 o_i (bF_2)$

$K_{2,1} = \alpha_i y F_2 + r'_2 o_i (bF_2) + w_1 h_i(\mathbf{id}) F_2$

$K_{2,2} = v'_i K_{2,3}$

$K_{2,3} = r'_2 F_2 - o_i (w_1 g_i(\mathbf{id}) + \alpha_i)(bF_2)$

$D_{j,1} = w_1 y_{i,j} F_2 + z'_{1,j} o_i (bF_2)$

$D_{j,2} = v'_i D_{j,3}$

$D_{j,3} = z'_{1,j} F_2 - w_1 \lambda_{i,j} o_i (bF_2)$

$J_{1,1} = w_2 y F_2 + r'_3 o_i (bF_2)$

$J_{1,2} = v'_i J_{1,3}$

$J_{1,3} = r'_3 F_2 - w_2 o_i (bF_2)$

$J_{2,1} = r'_4 o_i (bF_2) + w_2 h_i(\mathbf{id}) F_2$

$J_{2,2} = v'_i J_{2,3}$

$J_{2,3} = r'_4 F_2 - w_2 o_i g_i(\mathbf{id})(bF_2)$

$E_{j,1} = w_2 y_{i,j} F_2 + z'_{2,j} o_i (bF_2)$

$E_{j,2} = v'_i J_{2,3}$

$E_{j,3} = z'_{2,j} F_2 - w_2 \lambda_{i,j} o_i (bF_2)$

$\mathscr{B}$ implicitly sets:

$r_1 = r'_1 - w_1 o_i b$

$r_2 = r'_2 - o_i b(w_1 g_i(\mathbf{id}) + \alpha_i)$

$r_3 = r'_3 - w_2 o_i b$

$r_4 = r'_4 - w_2 g_i o_i b(\mathbf{id})$

$z_{1,j} = z'_{1,j} - w_1 \lambda_{i,j} o_i b$

$z_{2,j} = z'_{2,j} - w_2 \lambda_{i,j} o_i b$

If the request was for a domain key, $\mathscr{B}$ returns the components computed above to $\mathscr{A}$ as the domain key. Otherwise $\mathscr{B}$ computes the user key according to the algorithm **UserKeyGeneration** and returns the result to $\mathscr{A}$ as the user key.

A.2.4 Challenge.

$\mathscr{B}$ receives two message-policy pairs (M_0, P_0^*) and (M_1, P_1^*).

$\mathscr{B}$ chooses:

$\beta^* \overset{U}{\leftarrow} \{0, 1\}$

$\mathscr{B}$ explicitly sets (where $\mathbf{attr}_c$ ranges over each leaf node attribute in the policy):

$s\beta_d P_1 = \beta_d(b^2 sF_1 + y(sF_1))$

$e(P_1, P_{i,2})^{s\alpha_i} = (e(o_i^2(b^2 sF_1) + b^2 sF_1 + y(sF_1), yF_2)e(o_i^2(b^3 sF_1), bF_2))^{\alpha_i}$

$e(P_1, F_2)^{s\alpha_g} = e(b^2 sF_1 + y(sF_1), F_2)^{\alpha_g}$

$C_{1,1} = s\mathcal{H}_1(\mathbf{attr}_c) = o_i^2 g_i(\mathbf{attr}_c)(b^2 sF_1) + h_i(\mathbf{attr}_c)(sF_1)$

$C_{1,2} = as\mathcal{H}_1(\mathbf{attr}_c) + \mu\sigma F_1 = o_i^2 k_i g_i(\mathbf{attr}_c)Z_1 + k_i h_i(\mathbf{attr}_c)(asF_1)$

$C_{1,3} = -\tau s\mathcal{H}_1(\mathbf{attr}_c) - \mu\sigma V'_1 = -o_i^3 g_i(\mathbf{attr}_c)(b^3 sF_1) - o_i h_i(\mathbf{attr}_c)(bsF_1) - k_i o_i^2 v'_i g_i(\mathbf{attr}_c)Z_1 - k_i v'_i h_i(\mathbf{attr}_c)(asF_1)$

$C_{2,1} = sP_1 = b^2 sF_1 + y(sF_1)$

$C_{2,2} = asP_1 + \mu F_1 = k_i Z_1 + yk_i(asF_1)$

$C_{2,3} = -\tau s P_1 - \mu V_1' = -o_i(b^3 s F_1) - y o_i(bs F_1) - k_i v_i' Z_1 - k_i v_i'(as F_1)$

$\mathscr{B}$ implicitly sets:

$\mu = 0$ or *random*

$\sigma = g_i(\mathbf{attr}_c)$

$\mathscr{B}$ encrypts M_{β^*} under the policy $P^*_{\beta^*}$ using the values above and the algorithm $\mathbf{Encrypt}_{\mathsf{CPA}}$ and returns the result $\widehat{CT}_{\mathsf{CPA}}$. Note that if $Z_1 = ab^2 s F_1$ then the ciphertext is normal. Otherwise, $Z_1 = (ab^2 s + \mu)F_1$ for some $\mu \in_U \mathbb{Z}_p$ and the ciphertext contains semi-functional leaf nodes with $\mu = \mu$ and $\sigma = g_i(\mathbf{attr}_c)$. Finally, note that since $\mathscr{B}$ cannot compute aF_2, $\mathscr{B}$ cannot compute a semi-functional key in order to determine if CT_{CPA} is semi-functional.

A.2.5 Guess.

$\mathscr{A}$ returns its guess β' to $\mathscr{B}$

If Z_1 is real, $\widehat{CT}_{\mathsf{CPA}}$ is normal and therefore $\mathscr{B}$ is simulating $Game_{real}$. Otherwise, Z_1 is random and $\widehat{CT}_{\mathsf{CPA}}$ is semi-functional and therefore $\mathscr{B}$ is simulating $Game_0$. If $\mathscr{B}$ returns 1 if $\beta^* = \beta'$ and 0 otherwise, then it can solve the LW1 problem with advantage:

$$\mathrm{Adv}^{LW1}_{\mathscr{B}} = |Pr[\beta^* = \beta' | Z_1 \text{ is real}] - Pr[\beta^* = \beta' | Z_1 \text{ is random}]| = |Pr[X_{real}] - Pr[X_0]|$$

A.3 Lemma 4.2.

$|Pr[X_{k-1,1}] - Pr[X_{k,0}]| \leq \epsilon_{LW2}$ for $1 \leq k \leq q$

A.3.1 Assumption.

$\mathrm{LW2} = (F_1, dF_1, d^2F_1, bxF_1, dbxF_1, d^2xF_1, F_2, dF_2, bF_2, cF_2, Z_2)$

Let $Z_2 = (bc + \gamma)F_2$. It is hard to decide whether $\gamma = 0$ (Z_2 is real) or $\gamma \in_U \mathbb{Z}_p$ (Z_2 is random).

A.3.2 Setup(λ, n_{auth}).

$\mathscr{B}$ receives an instance of LW2 and chooses:

$p, e, \mathcal{H}, h$ based on λ

$\alpha_g, \beta_{id}, \{\beta_j\}_{j\in[0,r_{max}]}, \{\alpha_i, a_i, k_i, o_i, y_{v,i}, v'_i, \{y_{i,j}, \lambda_{i,j}\}_{j\in[1,h_{max}]}, u_i, v_i\}_{i\in[1,n_{auth}]} \xleftarrow{U} \mathbb{Z}_p^*$

$Q_{id}, U_{id} \xleftarrow{U} \mathbb{G}_2$

$\mathscr{B}$ explicitly sets $\forall i \in [1, n_{auth}]$:

$P_1 = dF_1$

$a_i P_1 = a_i(dF_1)$

$\tau_i P_1 = d^2F_1 + y_{v,i}(dF_1)$

$Q_{i,1,j} = \{\lambda_{i,j}(dF_1) + y_{i,j}F_1\}_{j\in[1,h_{max}]}$

$a_i Q_{i,1,j} = \{a_i(\lambda_{i,j}(dF_1) + y_{i,j}F_1)\}_{j\in[1,h_{max}]}$

$\tau_i Q_{i,1,j} = \{\lambda_{i,j}(d^2F_1) + \lambda_{i,j}y_{v,i}(dF_1) + y_{i,j}(dF_1) + y_{i,j}y_{v,i}F_1\}_{j\in[1,h_{max}]}$

$U_{i,1} = v_i(dF_1) + u_iF_1$

$a_i U_{i,1} = a_i(v_i(dF_1) + u_iF_1)$

$\tau_i U_{i,1} = v_i(d^2F_1) + v_i y_{v,i}(dF_1) + u_i(dF_1) + u_i y_{v,i}F_1$

$V_{i,2} = -a_i o_i(bF_2) + dF_2 + y_{v,i}F_2$

$V'_{i,2} = o_i(bF_2)$

$P_{i,2} = k_i(dF_2)$

$Q_{i,2,j} = \{\lambda_{i,j}(dF_2) + y_{i,j}F_2\}_{j\in[1,h_{max}]}$

$U_{i,2} = v_i(dF_2) + u_iF_2$

$e(P_1, P_{i,2})^{\alpha_i} = e(dF_1, k_i(dF_2))^{\alpha_i}$

GSK: $(\frac{1}{\beta 0}, \frac{-\alpha_g}{\beta 0}, \frac{\alpha_g}{\beta_{id}}, Q_{id}, U_{id})$

$\mathscr{B}$ implicitly sets $\forall i \in [1, n_{auth}]$:

$b_i = o_i b$

$v_i = -a_i b_i + d + y_{v,i} = -a_i o_i b + d + y_{v,i}$

$v'_i = b_i = o_i b$

$\tau_i = d + y_{v,i}$

$\mathscr{B}$ publishes for all players including $\mathscr{A}$:

GP: $(p, e, \mathcal{H}, h, P_1, F_2, \beta_{id} P_1, \frac{-\alpha_g}{\beta_0} F_2, \{\beta_j P_1, \frac{1}{\beta_j} F_2\}_{j \in [0, r_{max}]}, e(P_1, F_2)^{\alpha_g})$

AP_i: $\{a_i P_1, \tau_i P_1, U_{i,1}, a_i U_{i,1}, \tau_i U_{i,1}, \{Q_{i,1,j}, a_i Q_{i,1,j}, \tau_i Q_{i,1,j}\}_{j \in [1, h_{max}]}, e(P_1, P_{i,2})^{\alpha_i}\}_{i \in [1, n_{auth}]}$

A.3.3 Phases 1 and 2.

$\mathscr{A}$ makes q key extract queries labeled q_θ where $\theta \in [1, q]$. If $\theta < k$ a semi-functional key is returned and if $\theta > k$ a normal key is returned. $\mathscr{B}$ creates semi-functional keys using the master secret, a_i and F_2. From Z_2, if $\gamma = 0$ then the key is normal, otherwise if $\gamma \in_U \mathbb{Z}_p$ then the key is partially semi-functional with $\gamma_1 = \gamma$, $\pi = g_i(\mathbf{attr}_{c,k})$ and $\pi_{i,j} = \lambda_{i,j}$.

$\mathscr{B}$ chooses:

$w'_1, r'_2, (z'_{1,j})_{j \in [\ell+1, h']} \xleftarrow{U} \mathbb{Z}_p^*$

For $\theta = k$, $\mathscr{B}$ computes $\mathcal{S}_1$ $\forall j \in [\ell + 1, h']$:

$K_{1,1} = w'_1 P_2 - a_i Z_2 + y_{v,i}(cF_2)$

$K_{1,2} = Z_2$

$K_{1,3} = cF_2$

$K_{2,1} = \alpha_i P_2 + w'_1(g_i(\mathbf{attr}_{c,k})(dF_2) + h_i(\mathbf{attr}_{c,k})F_2) + r'_2 V_2 - a_i g_i(\mathbf{attr}_{c,k}) Z_2 + y_{v,i} g_i(\mathbf{attr}_{c,k})(cF_2) - h_i(\mathbf{attr}_{c,k})(cF_2)$

$K_{2,2} = r'_2 V'_2 + g_i(\mathbf{attr}_{c,k}) Z_2$

$K_{2,3} = r'_2 F_2 + g_i(\mathbf{attr}_{c,k})(cF_2)$

$D_{j,1} = w'_1 Q_{2,j} + z'_{1,j} V_2 - y_{i,j}(cF_2) - a_i \lambda_{i,j} Z_2 + y_{v,i} \lambda_{i,j}(cF_2)$

$D_{j,2} = z'_{1,j} V'_2 + \lambda_{i,j} Z_2$

$D_{j,3} = z'_{1,j} F_2 + \lambda_{i,j}(cF_2)$

$\mathscr{B}$ computes $\mathcal{S}_2$ normally.

$\mathscr{B}$ implicitly sets $\forall j \in [\ell + 1, h']$:

$w_1 = w'_1 - c$

$r_1 = c$

$r_2 = r'_2 + g_i(\mathbf{attr}_{c,k})c$

$z_{1,j} = z'_{1,j} - \lambda_{i,j} c$

A.3.4 Challenge.

$\mathscr{B}$ receives two message policy pairs (M_0, P^*_0) and (M_1, P^*_1).

$\mathscr{B}$ chooses:

$\beta^* \overset{U}{\leftarrow} \{0, 1\}$

$\mathscr{B}$ explicitly sets (where $\mathbf{attr}_c$ ranges over each leaf node attribute in the policy):

$s\beta_d P_1 = \beta_d(dbxF_1)$

$e(P_1, P_{i,2})^{s\alpha_i} = e(dbxF_1, k_i dF_2)^{\alpha_i}$

$e(P_1, F_2)^{s\alpha_g} = e(dbxF_1, F_2)^{\alpha_g}$

$C_{1,1} = s\mathcal{H}_1(\mathbf{attr}_c) = g_i(\mathbf{attr}_c)(dbxF_1) + h_i(\mathbf{attr}_c)(bxF_1)$

$C_{1,2} = as\mathcal{H}_1(\mathbf{attr}_c) + \mu\sigma F_1 = a_i g_i(\mathbf{attr}_c)(dbxF_1) + a_i h_i(\mathbf{attr}_c)(bxF_1) - g_i(\mathbf{attr}_c)(d^2 xF_1)$

$C_{1,3} = -\tau s\mathcal{H}_1(\mathbf{attr}_c) - \mu\sigma V'_1 = -y_{v,i} g_i(\mathbf{attr}_c)(dbxF_1) - h_i(\mathbf{attr}_c)(dbxF_1) - y_{v,i} h_i(\mathbf{attr}_c)(bxF1)$

$C_{2,1} = sP_1 = dbxF_1$

$C_{2,2} = asP_1 + \mu F_1 = a_i(dbxF_1) - d^2 xF_1$

$C_{2,3} = -\tau sP_1 - \mu V'_1 = -y_{v,i}(dbxF_1)$

$\mathscr{B}$ implicitly sets:

$s = bx$

$\mu = -d^2 x$

$\sigma = g_i(\mathbf{attr}_c)$

$\mathscr{B}$ encrypts M_{β^*} under the policy $P^*_{\beta^*}$ using the values above and the algorithm $\mathbf{Encrypt}_{\mathsf{CPA}}$ and returns the result $\widetilde{CT}_{\mathsf{CPA}}$.

A.3.5 Guess.

$\mathscr{A}$ returns its guess β' to $\mathscr{B}$

If Z_2 is real, $\mathscr{B}$ is simulating $Game_{k-1,1}$. Otherwise, Z_2 is random and $\mathscr{B}$ is simulating $Game_{k,0}$. If $\mathscr{B}$ returns 1 if $\beta^* = \beta'$ and 0 otherwise, then it can solve the LW2 problem with advantage:

$$\mathrm{Adv}^{LW2}_{\mathscr{B}} = |Pr[\beta^* = \beta' | Z_2 \text{ is real}] - Pr[\beta^* = \beta' | Z_2 \text{ is random}]| = |Pr[X_{k-1,1}] - Pr[X_{k,0}]|$$

A.4 Lemma 4.3.

$|Pr[X_{k,0}] - Pr[X_{k,1}]| \leq \epsilon_{LW2}$ for $1 \leq k \leq q$

This is the same procedure as **Lemma 2** except that $\mathcal{S}_2$ is used rather than $\mathcal{S}_1$. This is easily done since the only difference between $\mathcal{S}_1$ and $\mathcal{S}_2$ is that the latter does not include $\alpha_i P_2$. Since $\alpha_i P_2$ is explicity calculated, it is straightfoward to remove. $\mathcal{S}_1$ is made semi-functional for every key extraction query.

A.5 Lemma 4.4.

$|Pr[X_{q,1}] - Pr[X_{M-random}]| \leq \epsilon_{DBDH-3}$

A.5.1 Assumption.

DBDH-3 = $(F_1, aF_1, bF_1, sF_1, F_2, aF_2, bF_2, sF_2, Z_T)$

Let $Z_T = e(F_1, F_2)^{abs+\gamma}$. It is hard to decide whether $\gamma = 0$ (Z_T is real) or $\gamma \in_U \mathbb{Z}_p$ (Z_T is random).

A.5.2 Setup(λ, n_{auth}).

$\mathscr{B}$ receives an instance of DBDH-3 and chooses:

$p, e, \mathcal{H}, h$ based on λ

$\alpha_g, \beta_{id}, \{\beta_j\}_{j\in[0,r_{max}]}, y, \{k_i, v_i, v'_i, \{y_{i,j}\}_{j\in[1,h_{max}]}, u_i\}_{i\in[1,n_{auth}]} \overset{U}{\leftarrow} \mathbb{Z}_p^*$

$Q_{id}, U_{id} \overset{U}{\leftarrow} \mathbb{G}_2$

$\mathscr{B}$ explicitly sets $\forall i \in [1, n_{auth}]$:

$P_1 = yF_1$

$a_iP_1 = yk_i(aF_1)$

$\tau_iP_1 = v_iy(F_1) + k_iv'_iy(aF_1)$

$Q_{i,1,j} = \{y_{i,j}P_1\}_{j\in[1,h_{max}]} = \{y_{i,j}y(F_1)\}_{j\in[1,h_{max}]}$

$a_iQ_{i,1,j} = \{y_{i,j}yk_i(aF_1)\}_{j\in[1,h_{max}]}$

$\tau_iQ_{i,1,j} = \{v_iy_{i,j}y(F_1) + k_iv'_iy_{i,j}y(aF_1)\}_{j\in[1,h_{max}]}$

$U_{i,1} = u_iP_1 = u_iy(F_1)$

$a_iU_{i,1} = k_iu_iy(aF_1)$

$\tau_iU_{i,1} = v_iu_iy(F_1) + k_iv'_iu_iy(aF_1)$

$V_{i,2} = v_iF_2$

$V'_{i,2} = v'_iF_2$

$P_{i,2} = k_iy(F_2)$

$Q_{i,2,j} = \{y_{i,j}y(F_2)\}_{j\in[1,h_{max}]}$

$U_{i,2} = u_i y(F_2)$

$e(P_1, P_{i,2})^{\alpha_i} = e(aF_1, bF_2)^{k_i^2 y^2}$

GSK: $(\frac{1}{\beta 0}, \frac{-\alpha_g}{\beta 0}, \frac{\alpha_g}{\beta_{id}}, Q_{id}, U_{id})$

$\mathscr{B}$ implicitly sets $\forall i \in [1, n_{auth}]$:

$a_i = k_i a$

$\alpha_i = a_i b$

$\tau_i = v_i + a_i v_i'$

$\mathscr{B}$ publishes for all players including $\mathscr{A}$:

GP: $(p, e, \mathcal{H}, h, P_1, F_2, \beta_{id} P_1, \frac{-\alpha_g}{\beta_0} F_2, \{\beta_j P_1, \frac{1}{\beta_j} F_2\}_{j \in [0, r_{max}]}, e(P_1, F_2)^{\alpha_g})$

AP_i: $\{a_i P_1, \tau_i P_1, U_{i,1}, a_i U_{i,1}, \tau_i U_{i,1}, \{Q_{i,1,j}, a_i Q_{i,1,j}, \tau_i Q_{i,1,j}\}_{j \in [1, h_{max}]}, e(P_1, P_{i,2})^{\alpha_i}\}_{i \in [1, n_{auth}]}$

A.5.3 Phases 1 and 2.

$\mathscr{A}$ makes a number of key extract queries. Note that $\mathscr{B}$ does not know α_i or $\alpha_i F_2$ and therefore cannot create a normal key.

$\mathscr{B}$ receives a key query for a domain with $\mathbf{id} = (\mathsf{id}_1, ..., \mathsf{id}_\ell)$ under authority i where $i \in [1, n_{auth}]$ and chooses:

$r_1', r_2', r_3', r_4', (z_{1,j}', z_{2,j}', (\pi_j), \eta_j)_{j \in [\ell+1, h']}, w_1, w_2, \gamma_1', \gamma_1, \gamma_2, \eta \xleftarrow{U} \mathbb{Z}_p^*$

$\mathscr{B}$ explicitly sets $\forall j \in [\ell + 1, h']$:

$K_{1,1} = w_1 P_{i,2} + r_1 V_{i,2} - \gamma_1 k_i (aF_2)$

$K_{1,2} = r_1 V_{i,2}' + \gamma_1 F_2$

$K_{1,3} = r_1 F_2$

$K_{2,1} = \gamma_1' k_i (aF_2) + w_1 h_i(\mathbf{attr}) P_{i,2} + r_2 V_{i,2}$

$K_{2,2} = r_2 V_{i,2}' + y(bF_2) - \gamma_1' F_2$

$K_{2,3} = r_2 F_2$

$D_{j,1} = w_1 Q_{i,2,j} + z_{1,j} V_{i,2} - \gamma_1 \pi_j k_i (aF_2)$

$D_{j,2} = z_{1,j} V'_{i,2} + \gamma_1 \pi_j F_2$

$D_{j,3} = z_{1,j} F_2$

$J_{1,1} = w_2 P_{i,2} + r_3 V_{i,2} - \gamma_2 k_i (aF_2)$

$J_{1,2} = r_3 V'_{i,2} + \gamma_2 F_2$

$J_{1,3} = r_3 F_2$

$J_{2,1} = -\gamma'_2 \eta k_i (aF_2) + w_2 h_i(\mathbf{attr}) P_{i,2} + r_4 V_{i,2}$

$J_{2,2} = r_4 V'_{i,2} + \gamma_2 \eta F_2$

$J_{2,3} = r_4 F_2$

$E_{j,1} = w_2 Q_{i,2,j} + z_{2,j} V_{i,2} - \gamma_2 \eta_j k_i (aF_2)$

$E_{j,2} = z_{2,j} V'_{i,2} + \gamma_2 \eta_j F_2$

$E_{j,3} = z_{2,j} F_2$

$\mathscr{B}$ implicitly sets:

$a_i \gamma'_1 = by - \gamma_1 \pi$

A.5.4 Challenge.

$\mathscr{B}$ receives two message policy pairs (M_0, P_0^*) and (M_1, P_1^*).

$\mathscr{B}$ chooses:

$\beta^* \overset{U}{\leftarrow} \{0, 1\}\ \mu' \overset{U}{\leftarrow} \mathbb{Z}_p^*$

$\mathscr{B}$ explicitly sets (where $\mathbf{attr}_c$ ranges over each leaf node attribute in the policy):

$s\beta_d P_1 = \beta_d y(sF_1)$

$e(P_1, P_{i,2})^{s\alpha_i + k_i \gamma} = Z_T^{k_i}$

$e(P_1, F_2)^{s\alpha_g} = e(y(sF_1), F_2)^{\alpha_g}$

$C_{1,1} = s\mathcal{H}_1(\mathbf{attr}_c) = yh_i(\mathbf{attr}_c)(sF_1)$

$C_{1,2} = as\mathcal{H}_1(\mathbf{attr}_c) + \mu\sigma F_1 = h_i(\mathbf{attr}_c)\mu' F_1$

$C_{1,3} = -\tau s\mathcal{H}_1(\mathbf{attr}_c) - \mu\sigma V_1' = -v_i yh_i(\mathbf{attr}_c)(sF_1) - v_i' h_i(\mathbf{attr}_c)\mu' F_1$

$C_{2,1} = sP_1 = y(sF_1)$

$C_{2,2} = asP_1 + \mu F_1 = \mu' F_1$

$C_{2,3} = -\tau sP_1 - \mu V_1' = -yv_i(sF_1) - v_i'\mu' F_1$

$\mathscr{B}$ implicitly sets:

$s = s$

$\mu = \mu' - a_i sy$

$\sigma = h_i(\mathbf{attr}_c)$

$\mathscr{B}$ encrypts M_{β^*} under the policy $P^*_{\beta^*}$ using the values above and the algorithm $\mathbf{Encrypt}_{\mathsf{CPA}}$ and returns the result $\widehat{CT}_{\mathsf{CPA}}$.

A.5.5 Guess.

$\mathscr{A}$ returns its guess β' to $\mathscr{B}$

If Z_T is real, $\mathscr{B}$ is simulating $Game_{q,1}$. Otherwise, Z_T is random and $\mathscr{B}$ is simulating $Game_{final}$. If $\mathscr{B}$ returns 1 if $\beta^* = \beta'$ and 0 otherwise, then it can solve the DBDH-3 problem with advantage:

$$\mathrm{Adv}^{DBDH-3}_{\mathscr{B}} = |Pr[\beta^* = \beta' | Z_T \text{ is real}] - Pr[\beta^* = \beta' | Z_T \text{ is random}]| = |Pr[X_{q,1}] - Pr[X_{M-random}]|$$

A.6 Lemma 4.5.

$|Pr[X_{M-random}] - Pr[X_{final}]| \leq \epsilon_{A1}$

A.6.1 Assumption.

$\mathrm{A1} = (F_1, zF_1, dzF_1, azF_1, adzF_1, szF_1, F_2, zF_2, aF_2, xF_2, (dz - ax)F_2, Z_1)$

Let $Z_1 = csdzF_1$. It is hard to decide whether $c = 1$ (Z_1 is real) or $c \in_U \mathbb{Z}_p$ (Z_1 is random).

A.6.2 Setup(λ, n_{auth}).

$\mathscr{B}$ receives an instance of A1 and chooses:

$p, e, \mathcal{H}, h$ based on λ

$\alpha_g, \beta_{id}, \{\beta_j\}_{j\in[0,r_{max}]}, \{\alpha_i, k_i, o_i, v_i, v'_i, \{y_{i,j}\}_{j\in[1,h_{max}]}, u_i\}_{i\in[1,n_{auth}]} \xleftarrow{U} \mathbb{Z}_p^*$

$Q_{id}, U_{id} \xleftarrow{U} \mathbb{G}_2$

$\mathscr{B}$ explicitly sets $\forall i \in [1, n_{auth}]$:

$P_1 = zF_1$

$a_iP_1 = k_i(azF_1)$

$\tau_iP_1 = v_i(zF_1) + k_iv'_i(azF_1)$

$Q_{i,1,j} = \{y_{i,j}(dzF_1)\}_{j\in[1,h_{max}]}$

$a_iQ_{i,1,j} = \{k_iy_{i,j}(adzF_1)\}_{j\in[1,h_{max}]}$

$\tau_iQ_{i,1,j} = \{v_iy_{i,j}(dzF_1) + k_iv'_iy_{i,j}(adzF_1)\}_{j\in[1,h_{max}]}$

$U_{i,1} = u_i(dzF_1)$

$a_iU_{i,1} = k_iu_i(adzF_1)$

$\tau_iU_{i,1} = v_i(dzF_1) + k_iv'_i(adzF_1)$

$V_{i,2} = v_iF_2$

$V'_{i,2} = v'_iF_2$

$P_{i,2} = o_izF_2$

$e(P_1, P_{i,2})^{\alpha_i} = e(zF_1, o_i(zF_2))^{\alpha_i}$

GSK: $(\frac{1}{\beta 0}, \frac{-\alpha_g}{\beta 0}, \frac{\alpha_g}{\beta_{id}}, Q_{id}, U_{id})$

$\mathscr{B}$ implicitly sets $\forall i \in [1, n_{auth}]$:

$a_i = k_i a$

$z_i = o_i z$

$\tau_i = v_i + a_i v'_i$

$Q_{i,2,j} = \{y_{i,j}(dzF_2)\}_{j\in[1,h_{max}]}$

$U_{i,2} = u_i(dzF_2)$

$\mathscr{B}$ publishes for all players including $\mathscr{A}$:

GP: $(p, e, \mathcal{H}, h, P_1, F_2, \beta_{id}P_1, \frac{-\alpha_g}{\beta_0}F_2, \{\beta_j P_1, \frac{1}{\beta_j}F_2\}_{j\in[0,r_{max}]}, e(P_1, F_2)^{\alpha_g})$

AP_i: $\{a_i P_1, \tau_i P_1, U_{i,1}, a_i U_{i,1}, \tau_i U_{i,1}, \{Q_{i,1,j}, a_i Q_{i,1,j}, \tau_i Q_{i,1,j}\}_{j\in[1,h_{max}]}, e(P_1, P_{i,2})^{\alpha_i}\}_{i\in[1,n_{auth}]}$

A.6.3 Phases 1 and 2.

$\mathscr{A}$ makes q key extract queries.

$\mathscr{B}$ receives a key query for a domain with $\mathbf{id} = (\mathsf{id}_1, ..., \mathsf{id}_\ell)$ under authority i where $i \in [1, n_{auth}]$ and chooses:

$r'_1, r'_2, r'_3, r'_4, (z'_{1,j}, z'_{2,j}, (\pi_j), \eta_j)_{j\in[\ell+1,h']}, w_1, w_2, \pi', \gamma_1, \gamma_2, \eta' \xleftarrow{U} \mathbb{Z}_p^*$

$\mathscr{B}$ explicitly sets $\forall j \in [\ell + 1, h']$:

$K_{1,1} = w_1 P_{i,2} + r_1 V_{i,2} - \gamma_1 k_i (aF_2)$

$K_{1,2} = r_1 V'_{i,2} + \gamma_1 F_2$

$K_{1,3} = r_1 F_2$

$K_{2,1} = \alpha_i P_{i,2} + w_1 h_i(\mathbf{attr})(dz - ax)F_2 + r_2 V_{i,2} - \gamma_1 k_i \pi'(aF_2)$

$K_{2,2} = r_2 V'_{i,2} + w_1 h_i(\mathbf{attr})(xF_2) + \gamma_1 \pi' F_2$

$K_{2,3} = r_2 F_2$

$D_{j,1} = w_1 y_{i,j}(dz - ax)F_2 + z_{1,j} V_{i,2} - \gamma_1 \pi'_j k_i (aF_2)$

$D_{j,2} = z_{1,j} V'_{i,2} + w_1 y_{i,j}(xF_2) + \gamma_1 \pi'_j F_2$

$D_{j,3} = z_{1,j} F_2$

$J_{1,1} = w_2 P_{i,2} + r_3 V_{i,2} - \gamma_2 k_i (aF_2)$

$J_{1,2} = r_3 V'_{i,2} + \gamma_2 F_2$

$J_{1,3} = r_3 F_2$

$J_{2,1} = w_2 h_i(\mathbf{attr})(dz - ax)F_2 + r_4 V_{i,2} - \gamma_2 \eta' k_i (aF_2)$

$J_{2,2} = r_4 V'_{i,2} + w_2 h_i(\mathbf{attr})(xF_2) + \gamma_2 \eta' F_2$

$J_{2,3} = r_4 F_2$

$E_{j,1} = w_2 y_{i,j}(dz - ax)F_2 + z_{2,j} V_{i,2} - \gamma_2 \eta'_j k_i (aF_2)$

$E_{j,2} = z_{2,j} V'_{i,2} + w_1 y_{i,j}(xF_2) + \gamma_2 \eta'_j F_2$

$E_{j,3} = z_{2,j} F_2$

$\mathscr{B}$ implicitly sets:

$\pi = \pi' + \gamma_1^{-1} w_1 h_i(\mathbf{attr}) x$

$\pi_j = \pi' j + \gamma_1^{-1} w_1 y_{i,j} x$

$\eta = \eta' + \gamma_2^{-1} w_2 h_i(\mathbf{attr}) x$

$\eta_j = \eta'_j + \gamma_2^{-1} w_2 y_{i,j} x$

A.6.4 Challenge.

$\mathscr{B}$ receives two message policy pairs (M_0, P_0^*) and (M_1, P_1^*).

$\mathscr{B}$ chooses:

$\beta^* \xleftarrow{U} \{0, 1\}$

$a'_i, \xi \xleftarrow{U} \mathbb{Z}_p^*$

$\mathscr{B}$ explicitly sets (where $\mathbf{attr}_c$ ranges over each leaf node attribute in the policy):

$s\beta_d P_1 = \beta_d (szF_1)$

$e(P_1, P_{i,2})^{s\alpha_i} \xleftarrow{U} \mathbb{G}_T$

$e(P_1, F_2)^{s\alpha_g} \xleftarrow{U} \mathbb{G}_T$

$C_{1,1} = s\mathcal{H}_1(\mathbf{attr}_c) = h_i(\mathbf{attr}_c)Z_1$

$C_{1,2} = as\mathcal{H}_1(\mathbf{attr}_c) + \mu\sigma F_1 = a_i' h_i(\mathbf{attr}_c)Z_1 + \xi F_1$

$C_{1,3} = -\tau s\mathcal{H}_1(\mathbf{attr}_c) - \mu\sigma V_1' = -v_i h_i(\mathbf{attr}_c)Z_1 - v_i' a_i' h_i(\mathbf{attr}_c)Z_1 - v_i'\xi F_1$

$C_{2,1} = sP_1 = szF_1$

$C_{2,2} = asP_1 + \mu F_1 = a_i'(szF_1)$

$C_{2,3} = -\tau sP_1 - \mu V_1' = -v_i(szF_1) - v_i' a_i'(szF_1)$

$\mathscr{B}$ implicitly sets:

$a_i' = k_i a + \mu'$

$\mu = \mu' sz$

$\xi = \mu\sigma'$

$\sigma = \sigma' + cdh_i(\mathbf{attr}_c)$

$\mathscr{B}$ encrypts M_{β^*} under the policy $P^*_{\beta^*}$ using the values above and the algorithm $\mathbf{Encrypt}_{\mathsf{CPA}}$ and returns the result $\widehat{\mathcal{CT}}_{\mathsf{CPA}}$.

A.6.5 Guess.

$\mathscr{A}$ returns its guess β' to $\mathscr{B}$

If Z_1 is real then $\widehat{\mathcal{CT}}_{\mathsf{CPA}}$ encrypts a random message under policy $P^*_{\beta^*}$ and simulates $Game_{M-random}$. If Z_1 is random, then $\widehat{\mathcal{CT}}_{\mathsf{CPA}}$ encrypts a random message under a random, structurally equivalent policy and simulates $Game_{final}$. If $\mathscr{B}$ returns 1 if $\beta^* = \beta'$ and 0 otherwise, then it can solve the A1 problem with advantage:

$$\mathrm{Adv}^{A1}_{\mathscr{B}} = |Pr[\beta^* = \beta' | Z_1 \text{ is real}] - Pr[\beta^* = \beta' | Z_1 \text{ is random}]| = |Pr[X_{M-random}] - Pr[X_{final}]|$$

■

Appendix B: MA-AHASBE Implementation Details

The MA-AHASBE system is introduced in Chapter 4. This chapter presents the details of a prototype implementation of MA-AHASBE. The prototype implementation serves as a proof of concept as well as the primary means of evaluating MA-AHASBE performance. The chapter first introduces the tools and techniques used to build the implementation. The next section then describes issues that must be addressed in an implementation that may not be obvious from the construction given in Chapter 4. The performance of the MA-AHASBE implementation is then presented and analyzed. The data is presented for both size and time. The chapter concludes with a summary and some thoughts on future work.

B.1 Tools and Techniques

Java / C++ / Qt5 / MinGW The prototype for MA-AHASBE is primarily written in C++ and uses the Qt5 library. The code is compiled using the MinGW 4.8.1 compiler for Windows. The current codebase is designed to be cross-platform and could build on Windows, Linux and MacOS X. Java bindings to the native C++ have been written and the Java implementation of the Amazon Web Services Software Development Kit (AWS SDK) is used to interact with Amazon's cloud services.

Sodium Qt5 provides SHA based hashing and message authentication code generation, but does not contain many other cryptographic algorithms. Sodium is used for cryptographic algorithms not provided by the Qt5 library such as AES and RSA.

YAML YAML Aint Markup Langauge (YAML) is a data serialization langauge that is designed to be both human readable and relatively efficient to process. It provides similar functionality as other data serialization formats such as JSON and XML. Like JSON it provides methods for representing lists, associative arrays and scalars. It is useful for representing tree-based or hierarchical structures, especially when the

content may need to be read or edited manually. The MA-AHASBE implementation uses the YAML-CPP 0.5.1 and SnakeYAML 1.14 projects, which are C++ and Java (respectively) implementations of the YAML language for reading and writing the various data structures in MA-AHASBE.

RELIC MA-AHASBE uses RELIC Tookit 0.3.5 [4], which is a pairing and multiple precision arithmetic tookit. RELIC is used for all finite field arithmetic and to perform all pairing operations. It was chosen over another popular pairing library PBC due to is cross platform design, highly customizable build process, extensive unit testing, and support for generic finite field arithmetic. RELIC has even been used for embedded wireless sensor networks [84].

B.2 Implementation

This section addresses some of issues that must be addressed in an implementation of MA-AHASBE. The mathematical construction of MA-AHASBE leaves many choices up to the implementation, such as the security parameter λ or choice of bilinear group, and the result of these choices may have a significant effect on the performance of the system. The following list describes the specific choices made for the prototype implementation. Elements listed in parenthesis represent the relevant mathematical terms presented in Chapter 4.

Security Parameter (λ) The security parameter is chosen to be $\lambda = 128$. This level of security is quite common for cryptographic implementations. This means that the prototype implementation can be used to transport 128-bit AES keys and provides a reasonable tradeoff between security and efficiency. As described in the *Elliptic Curve* entry below, there are efficiency advantages for pairing based cryptography by staying with 128-bit security, particularly for Type-3 pairings.

Cryptographic Hash ($\mathcal{H}$) For this prototype, $\mathcal{H} = SHA\text{-}256$. The implementation comes from the Qt5 library. It might seem odd that SHA-256 was chosen rather than SHA-128 if $\lambda = 128$. This is due to the nature of the underlying pairing implementation. In order to have 128-bit security, points on the elliptic curve described below are elements of $\mathbb{Z}_p$ where p is a 256-bit prime number. So a 256-bit hash algorithm is a more natural fit due to the size of the underlying finite field of the elliptic curve, rather the security parameter. In addition to its uses as described in Chapter 4, this hash has two other important functions. First, it creates a unique name for each global or public parameter that is created through the **GlobalSetup** and **AuthoritySetup** algorithms respectively by hashing the contents of the parameters to be used as a reference throughout the system. Secondly, the construction in Chapter 4 describes the entries in identity tuples as being elements of $\mathbb{Z}_p$. In a real world implementation, however, these elements tend to be strings. The hash algorithm $\mathcal{H}$ is used to map text strings to elements of $\mathbb{Z}_p$ by the calculation: $\mathsf{id} = \mathcal{H}(\mathsf{ID}_{string}) \, mod \, p$ where $\mathsf{id} \in \mathbb{Z}_p$ and ID_{string} represents a string value such as "Student" or "University". Since $\mathcal{H}$ is a cryptographic hash algorithm and therefore collision resistant, in practical terms (and with high probability) this mapping will be bijective over the strings used for identities.

Pairwise Hash (h) In order to perform the CCA transformation described in [19], a hash function from a family of pairwise independent hash functions must be selected. For MA-AHASBE, the relatively straightforward approach described in [26] is used. This is done with the following formula: $\mathsf{h}(x) = ((ax + b) \, mod \, q) \, mod \, m$ where x is an value to be hashed, q is a 448 bit prime number, $a, b \in_U \mathbb{Z}_q^*$ and m is 2^{129}. Note that this hash function is only used to generate a key to the MAC algorithm which only requires a 128-bit key, and is not used to map to finite field elements or points on the elliptic curve. Therefore, unlike $\mathcal{H}$, its values can remain 128-bits in size.

Elliptic Curve (p, e, $\mathbb{G}_1$, $\mathbb{G}_2$, $\mathbb{G}_T$) The choice of elliptic curve is narrowed down by the pairing library RELIC and the fact that MA-AHASBE is based on Type-3 pairings. At the 128-bit security level, Baretto-Naehrig (BN) curves [10, 35, 88] are the most efficient curves. The choice of generators for each group, the prime p, and the implementation of the pairing e is based on the BN family of curves. This is also the default choice for asymmetric pairings in RELIC.

Random Number Generation In cryptographic implementations, the source of entropy can become very important in terms of both performance and security. Sources with higher entropy (good for cryptographic purposes) such as an operating system's secure random generator can often incur a latency cost in order for the operating system to ensure enough entropy is available in the result. Other sources can offer quicker results (good for performance), but may have lower amounts of overall entropy (bad for cryptographic purposes). The random number generator in the MA-AHASBE implementation is set to draw from the operating system's secure random number generator. Specifically, the results in this chapter are using the **CryptGenRandom** function which is part of the Windows cryptographic application programming interface.

Data Structure Representation In an actual implementation, the data structures presented in Chapter 4 must have some type of in-memory and on-disk representation. This includes global parameters, authority parameters, policies, ciphertexts, keys, and key rings. While the parameters and keys are relatively flat, the policies and ciphertexts have a hierarchical aspect to them. YAML was chosen as a data serialization format because it is easy to manipulate the variety of data structures found in MA-AHASBE. Also as a prototype implementation, it is important that the data serialization format is easily readable and writable by a human to facilitate development. YAML uses significant whitespace, which means that whitespace has syntactical meaning. While

this helps boost readability, it also has a some cost in terms of the size of the data structures.

Finally, it is worth noting items mentioned in Chapter 4 that are not a part of the current implementation. The following items are currently not implemented:

- Cross domain attributes
- Updating single attributes of key structures
- Decentralized global parameter generation through SMPC

B.2.1 Side Effects and Optimization.

Before discussing the performance metrics, it is worth noting that the implementation of MA-AHASBE is not optimized nor rigorously scrubbed for potential side effects. In the case of optimization, the design of the implementation has been focused on the practicality, reliability and clarity of the code rather than on its performance. As such it could be expected that another design with a heavier emphasis on performance would perform faster. On the other hand, the code has not been thoroughly analyzed for potential side effects. As is often the case with implementations of cryptographic algorithms, the most serious vulnerabilities are discovered through so called *side-channel analysis*. Side effects are introduced in cryptographic implementations when there exists some detectable difference in performance when executing operations with sensitive key data. The performance difference could be in terms of power usage, sound emissions, heat, time, or a variety of other factors. RELIC offers a variety of options to select algorithms that are less susceptible to this type of analysis. For example, RELIC can use the Montgomery's ladder technique to perform elliptic curve point addition. This technique is not as efficient as other more direct techniques, but involves a deterministic set of operations that is independent of the operands. So while further optimization could improve the implementation performance, a more robust check of potential side effects could result in potentially slower algorithms.

In the end, the prototype implementation is not designed to be production quality, but to serve more as an indicator to what MA-AHASBE performance might be in a real-world implementation.

B.3 Performance

The performance of MA-AHASBE is primarily dependent on policy size, but if certain attribute protection mechanisms are used the performance can also depend on the number of decryption keys possessed by the decrypting party. In order to present the performance characteristics, two examples have been chosen to demonstrate to points of complexity. The first scenario has a relatively simple policy, while the second scenario contains a more complex policy with the attribute protection mechanism turned on. The first policy is shown in Figure B.1 and the second policy is shown in Figure B.2.

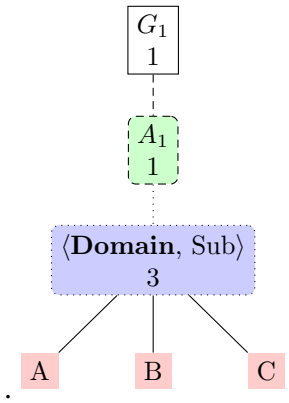
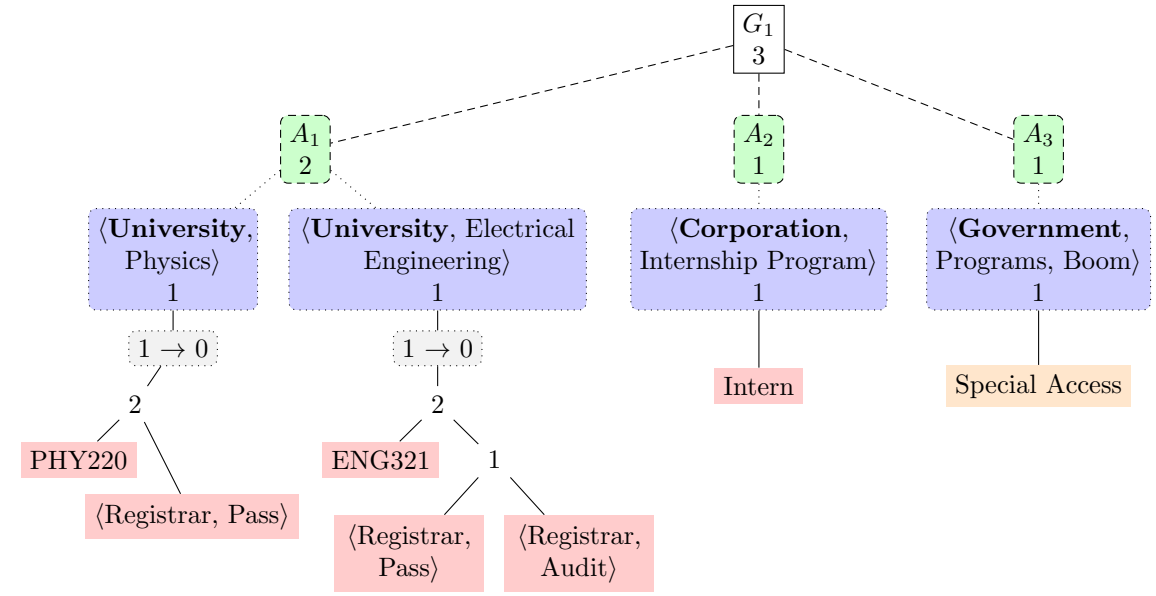


Figure B.1: Scenario 1

Information regarding the policy sizes are shown in Table B.1. The size of the key rings necessary for decryption are shown in Table B.2. Finally, the time required for performing encryption and decryption is shown in Table B.3. Each result shows the average of five samples, with the standard deviation shown in parenthesis.



.

Figure B.2: Scenario 2

	Scenario 1	Scenario 2
Domains	1	4
Total Nodes	7	17
Leaf Nodes	3	7
Hidden Nodes	0	1
Policy Size	0.295 kb (0.001)	1.552 kb (0.002)
Ciphertext Size	1.505 kb (0.004)	3.289 kb (0.003)

Table B.1: Policy Sizes

B.4 Sign/Verify Constructions

Constructions for creating and verifying signatures with MA-AHASBE are presented without proof here (the proof should be a straightfoward application of the argument presented in [47] regarding hierarchical IBE):

	Scenario 1	Scenario 2
Domains	1	4
Attribute Sets	1	8
Attributes	4	13
Key Ring Size	5.412 kb (0.005)	16.251 kb (0.015)

Table B.2: Key Sizes

	Scenario 1	Scenario 2
Hidden Node Recovery	N/A	0.562 s (0.194)
Key Ring Generation	1.680 s (0.038)	5.329 s (0.027)
Encryption	0.091 s (0.002)	0.221 s (0.002)
Decryption	1.007 (0.008)	2.613 s (0.207)

Table B.3: Timing

Sign$(GP, PP_A, \mathcal{M}, USK, \mathcal{ID}_{attribute})$ Let $\mathcal{H}(\mathcal{M})$ mod $p = m$ where $\mathcal{H}(\mathcal{M})$ is the cryptographic hash of the input message. Run **Delegate**$(USK, PP_A, \mathcal{ID}_{attribute} = (\mathsf{id}_1, ..., \mathsf{id}_\ell, m = \mathsf{id}_{\ell+1}, \ell + 1)$. Return $\sigma = (m, \mathcal{ID}_{attribute}, \mathbb{J})$ where $\mathbb{J}$ is the set of J components in the key returned by the **Delegate** algorithm.

Verify$(GP, \mathcal{M}, PP_A, \mathcal{ID}_{attribute}, \sigma)$ Let $\mathcal{H}(\mathcal{M})$ mod $p = m$ where $\mathcal{H}(\mathcal{M})$ is the cryptographic hash of the input message. If m or $\mathcal{ID}_{attribute}$ do not match their corresponding values in the signature, return *false*. Let $\mathsf{attr}_m = (\mathcal{ID}_{attribute}, m)$. Create a set of ciphertexts similar to those created for a local leaf node in the **Encrypt** algorithm, but without multiplying by the value $q_t(0)$. Then run the **DecryptNode** algorithm on the ciphertext as if it was a local leaf node, replacing the K components with the J components from the

signature. If the result is equal to one, then the signature verifies and return *true*. Return *false* otherwise. The decryption equation is given below:

$$1 \stackrel{?}{=} \frac{e(C_{1,1}, J_{1,1})e(C_{1,2}, J_{1,2})e(C_{1,3}, J_{1,3})}{e(C_{2,1}, J_{2,1})e(C_{2,2}, J_{2,2})e(C_{2,3}, J_{2,3})} \stackrel{?}{=} \frac{e(\mathcal{H}_1(\mathsf{attr}_m), w_2 P_2 + r_3 V_2)e(a\mathcal{H}_1(\mathsf{attr}_m), r_3 V_2')e(-\tau\mathcal{H}_1(\mathsf{attr}_m), r_3 F_2)}{e(P_1, w_2\mathcal{H}_2(\mathsf{attr}_m) + r_4 V_2)e(aP_1, r_4 V_2')e(-\tau P_1, r_4 F_2)}$$

$$\stackrel{?}{=} \frac{e(\mathcal{H}_1(\mathsf{attr}_{\mathbf{com}}), P_2)^{w_2} e(\mathcal{H}_1(\mathsf{attr}_m), V_2)^{r_3} e(\mathcal{H}_1(\mathsf{attr}_m), V_2')^{ar_3} e(\mathcal{H}_1(\mathsf{attr}_m), F_2)^{-\tau r_3}}{e(P_1, \mathcal{H}_2(\mathsf{attr}_m))^{w_2} e(P_1, V_2)^{r_4} e(P_1, V_2')^{ar_4} e(P_1, F_2)^{-\tau r_4}}$$

Appendix C: Notes on Encrypted Search and Real-Time Collaboration

The attribute-based encryption system presented in Chapters 4 and 5 addresses the problem of encrypting documents in preparation for storing them in the cloud. However, once encrypted documents are placed in the cloud, the CSP is now limited in the amount of processing and services it can provide. Therefore, performing text based search over encrypted documents is a critical task in order to consider placing a large volume of potentially sensitive documents on untrusted cloud resources. This appendix examines how MA-AHASBE can be combined with existing encrypted search techniques to provide an end-to-end solution.

C.1 Related Work on Enrypted Search

Storing encrypted documents in the cloud reduces amount of trust a user needs to place in the CSP, but at the same time it also restricts the useful things that the CSP can do with the data. Perhaps the most straightforward consequence is that it is no longer a straightforward task to index and search over the store of documents. Text search and information retrieval have become cornerstones of a digital age. Due to the popularity of Internet search engines, keyword based search with ranked results has become a natural way to perform information retrieval over a large collection of documents. Search engines are able to build their search indices quickly and efficiently because the engines have direct, plaintext access to the documents.

One approach is to encrypt all data client-side before sending it to the CSP. While this does prevent the CSP from accessing potentially sensitive data, it could also prevent the CSP from doing any meaningful work on the data. A motivating example is the service of providing keyword search. Traditional search engine indexing schemes rely on having access to the plaintext data in order to create efficient search indices in order to quickly

return the most relevant results to a user query. A plaintext search engine can perform sophisticated lexical analysis of both the search query terms as well as the underlying dataset independent of any client processing of the data. However, when the dataset is encrypted on the client side, the client must assume responsibility for processing the data. Therefore, minimizing this processing burden on the client while maintaining the security and privacy benefits of client-side encryption are desirable properties of an encrypted search algorithm.

It is worthwhile to note that the concept of remote encrypted storage retrieval is distinct from another related concept of private information retrieval (PIR). With PIR, the goal is to allow users to make queries on a shared database without other parties or the database server knowing the contents of the query. Often PIR systems utilize databases that are stored in plaintext, since the goal is not to protect the stored information, but rather the query. This is not the same as protecting the content of the data stored on a remote, untrusted server.

One of the earliest papers to discuss the topic of remote encrypted storage retrieval by keyword was by Song et. al [96]. Building on this work, a number of additional papers have since been published proposing various enhancements. A method for efficient conjunctive keyword search is described by Golle et al in [50]. Proposed improvements to this method are in [22] and [7]. However, analysis of [22], [7] show they may not be semantically secure as described in [61]. Another set of attacks on conjunctive keyword search schemes is described in [92]. This illustrates that it is not trivial to extend these cryptographic systems in ways that provide efficient indexed search as well as preserving privacy. In addition to conjunctive search, techniques for fuzzy keyword searches are found in [8], [9] and [10]. Finally, techniques for performing ranked keyword search can be found in [24, 25, 106, 107]. The system presented in [25] called MSRE represents the state of the art in multi-keyword ranked search.

C.1.1 Homomorphic Encryption.

Homomorphic encryption is often cited as the gateway to remote encrypted storage and private cloud computing. Homomorphic encryption allows a third party to perform a arithmetic operation (such as add, subtract, multiply and divide) on the encrypted form of data without having the decryption key, and the effect will carry over to the corresponding plaintext. Many encryption schemes used in public key cryptography have natural homomorphic properties. These schemes are usually only homomorphic for some subset of operations, such as just addition/subtraction with Paillier [85], or multiplication with RSA [93]. These schemes are referred to as partially homomorphic encryption systems.

The first fully homomorphic scheme was discovered by Gentry [45]. It uses a worst case generation of an ideal lattice structure and the difficulty of searching through such a structure to bootstrap a partially homomorphic system into a fully homomorphic encryption system. Since Gentry's discovery, improvements on the efficiency of this system have been made. Research has also been done on the practical applications of homomorphic encryption [16].

A major challenge for fully homomorphic encryption algorithms to be used in search over encrypted data is the large cost of encryption and decryption, as well as key generation and storage. For example, the implementation of Gentry's algorithm in [46] describes "small" key sizes of 70 Megabytes and "large" key sizes of 2.3 Gigabytes. Also the time to generate these keys range from 30 seconds for the small case and 30 minutes for the large case. Some more recent work at optimizing fully homomorphic encryption schemes has been done using ring learning with error (RWLE) problems [20, 98]. Even as the efficiency of fully homomorphic encryption algorithms increases, the applications so far require encrypting each bit or token of plaintext using the fully homomorphic encryption. In practical systems, this operation is too expensive even for partially homomorphic encryption such as RSA. Typically, a fast symmetric key is used for bulk encryption of data. The symmetric key,

which is very small compared to the size of the data that needs to be protected, can then itself be protected through a more expensive operation such as Paillier or RSA. It seems that if homomorphic encryption will be used in securing cloud computing, that it will be in conjunction, rather than in place of, traditional cryptographic concepts such symmetric key encryption, public key encryption and secure hash techniques. The difficulty of mapping a fully homomorphic encryption primitive to the general class of private, multi-client computing problems as described in [103] is still an open research problem.

C.2 Multi-Keyword Ranked Search Over Encrypted Data

The multi-keyword ranked search over encrypted data algorithm MSRE [25] is a recent advancement in the area of encrypted search. It uses a variation of a secure k-nearest neighbor search using a document vector model. In this approach, documents are represented as vectors in a vector space. A query consists of various keywords that represent content the user is interested in. When a query is made, it is first converted to a vector in the same manner as documents. This query vector is then compared against every document in the collection, using the cosine of the angles of the vectors as a similarity measure. Query and document vectors that are oriented in the same way in vector space are assumed to be closely related. Since the similarity measure comes out as a numerical quantity, it also provides for a ranking of the similarity. MSRE uses a symmetric key to protect both the contents of the query and the indices built from the documents. The primary limitation is that the dictionary size used to build the indices must be a fixed size. Also, MSRE requires a security parameter that trades security for search accuracy. MSRE appears to be a promising approach for doing keyword searches over encrypted data.

C.3 Related Work on Group Collaboration

Work on real-time, group collaboration algorithms date back to 1989 to the work of Gibbs and Ellis [36]. This introduced the idea of the operational transformation (OT) which is

an algorithmic strategy that addresses the issues that arise in high latency, concurrent, real-time systems that require a shared working state. The advantages of the OT algorithm is that it does not require locks and edits to the shared state can be applied to the local copy immediately. In order for the OT algorithm to work, there must be a transformation algorithm that can work pairwise over every type of operation (e.g., add a character, remove a character, move cursor). The transformation algorithm is used to effectively reorder operations that arrive from remote clients. This process allows local edits to be applied immediately, which reduces the perception of latency to the local user. The OT algorithm can then process any remote edits that were, with respect to global time, performed before the local edits. This reduces the impact of a high latency communication channel and is perfectly suited for applications such as real-time document editing.

C.3.1 SPORC.

SPORC [37] is a group collaboration protocol that operates on a security model of untrusted servers. It combines the previous techniques of OT and fork* consistency. In order to detect malicious servers, the protocol relies on the clients in the system having access to an out-band communication channel in order to detect if the server is malicious; though this channel may be very low bandwidth. The role the server is reduced to providing availability of the data, which works well under the A-Trusted security paradigm discussed in Chapter 5.

Bibliography

[1] "CS 6260 Number-theoretic primitives". URL http://www.cc.gatech.edu/~aboldyre/teaching/Fall05cs6260/dl.pdf.

[2] "DISA: Rapid Access Computing Environment". URL http://www.disa.mil/Services/Enterprise-Services/Infrastructure/RACE.

[3] "Elliptic Curve". URL https://en.wikipedia.org/wiki/Elliptic_curve.

[4] Aranha, D. F. and C. P. L. Gouvêa. "RELIC is an Efficient LIbrary for Cryptography". http://code.google.com/p/relic-toolkit/.

[5] Bach, Eric. *Discrete logarithms and factoring*. Computer Science Division, University of California Berkeley, 1984.

[6] Bachrach, Dustin, Christopher Nunu, Dan S Wallach, and Matthew Wright. "# h00t: Censorship Resistant Microblogging". *arXiv preprint arXiv:1109.6874*, 2011.

[7] Ballard, Lucas, Seny Kamara, and Fabian Monrose. "Achieving efficient conjunctive keyword searches over encrypted data". *Information and Communications Security*, 414–426. Springer, 2005.

[8] Barazzutti, Raphaël, Pascal Felber, Hugues Mercier, Emanuel Onica, and Etienne Rivière. "Thrifty privacy: efficient support for privacy-preserving publish/subscribe". *Proceedings of the 6th ACM International Conference on Distributed Event-Based Systems*, 225–236. ACM, 2012.

[9] Barreto, Paulo SLM, Ben Lynn, and Michael Scott. "Constructing elliptic curves with prescribed embedding degrees". *Security in Communication Networks*, 257–267. Springer, 2003.

[10] Barreto, Paulo SLM and Michael Naehrig. “Pairing-friendly elliptic curves of prime order”. *Selected areas in cryptography*, 319–331. Springer, 2006.

[11] Bellare, Mihir and Phillip Rogaway. “Random oracles are practical: A paradigm for designing efficient protocols”. *Proceedings of the 1st ACM conference on Computer and communications security*, 62–73. ACM, 1993.

[12] Belokosztolszki, András, David M Eyers, Peter R Pietzuch, Jean Bacon, and Ken Moody. “Role-based access control for publish/subscribe middleware architectures”. *Proceedings of the 2nd international workshop on Distributed event-based systems*, 1–8. ACM, 2003.

[13] Bethencourt, John, Amit Sahai, and Brent Waters. “Ciphertext-policy attribute-based encryption”. *Security and Privacy, 2007. SP’07. IEEE Symposium on*, 321–334. IEEE, 2007.

[14] Biddick, Michael. *Federal Cloud Computing*. Cavalier Trail Books, first edition, November 2012.

[15] Blake, Ian F, Gadiel Seroussi, and Nigel Smart. *Elliptic curves in cryptography*, volume 265. Cambridge university press, 1999.

[16] Bobba, Rakesh, Omid Fatemieh, Fariba Khan, Arindam Khan, Carl A Gunter, Himanshu Khurana, and Manoj Prabhakaran. “Attribute-based messaging: Access control and confidentiality”. *ACM Transactions on Information and System Security (TISSEC)*, 13(4):31, 2010.

[17] Bobba, Rakesh, Himanshu Khurana, and Manoj Prabhakaran. “Attribute-sets: A practically motivated enhancement to attribute-based encryption”. *Computer Security–ESORICS 2009*, 587–604. Springer, 2009.

[18] Boneh, Dan and Matt Franklin. "Identity-based encryption from the Weil pairing". *Advances in Cryptology–CRYPTO 2001*, 213–229. Springer, 2001.

[19] Boneh, Dan and Jonathan Katz. "Improved efficiency for CCA-secure cryptosystems built using identity-based encryption". *Topics in Cryptology–CT-RSA 2005*, 87–103. Springer, 2005.

[20] Brakerski, Zvika, Craig Gentry, and Vinod Vaikuntanathan. "(Leveled) fully homomorphic encryption without bootstrapping". *Proceedings of the 3rd Innovations in Theoretical Computer Science Conference*, 309–325. ACM, 2012.

[21] Broker, Reinier and Peter Stevenhagen. "Constructing elliptic curves of prime order". *arXiv preprint arXiv:0712.2022*, 2007.

[22] Byun, Jin Wook, Dong Hoon Lee, and Jongin Lim. "Efficient conjunctive keyword search on encrypted data storage system". *Public Key Infrastructure*, 184–196. Springer, 2006.

[23] Canetti, Ran, Uri Feige, Oded Goldreich, and Moni Naor. "Adaptively secure multi-party computation". *Proceedings of the twenty-eighth annual ACM symposium on Theory of computing*, 639–648. ACM, 1996.

[24] Cao, Ning, Cong Wang, Ming Li, Kui Ren, and Wenjing Lou. "Privacy-preserving multi-keyword ranked search over encrypted cloud data". *INFOCOM, 2011 Proceedings IEEE*, 829–837. IEEE, 2011.

[25] Cao, Ning, Cong Wang, Ming Li, Kui Ren, and Wenjing Lou. "Privacy-preserving multi-keyword ranked search over encrypted cloud data". *Parallel and Distributed Systems, IEEE Transactions on*, 25(1):222–233, 2014.

[26] Carter, J Lawrence and Mark N Wegman. "Universal classes of hash functions". *Proceedings of the ninth annual ACM symposium on Theory of computing*, 106–112. ACM, 1977.

[27] Chai, Qi and Guang Gong. "Verifiable symmetric searchable encryption for semi-honest-but-curious cloud servers". *Communications (ICC), 2012 IEEE International Conference on*, 917–922. IEEE, 2012.

[28] Chase, Melissa. "Multi-authority attribute based encryption". *Theory of Cryptography*, 515–534. Springer, 2007.

[29] Chase, Melissa and Sherman SM Chow. "Improving privacy and security in multi-authority attribute-based encryption". *Proceedings of the 16th ACM conference on Computer and communications security*, 121–130. ACM, 2009.

[30] Chatterjee, Sanjit and Palash Sarkar. *Identity-Based Encryption*. Springer, 2011.

[31] Chen, Cheng, Zhenfeng Zhang, and Dengguo Feng. "Efficient ciphertext policy attribute-based encryption with constant-size ciphertext and constant computation-cost". *Provable Security*, 84–101. Springer, 2011.

[32] Cheung, Ling and Calvin Newport. "Provably secure ciphertext policy ABE". *Proceedings of the 14th ACM conference on Computer and communications security*, 456–465. ACM, 2007.

[33] De Cristofaro, Emiliano, Claudio Soriente, Gene Tsudik, and Albert Williams. "Hummingbird: Privacy at the time of twitter". *Security and Privacy (SP), 2012 IEEE Symposium on*, 285–299. IEEE, 2012.

[34] Deftu, Andrei and Jan Griebsch. "A scalable conflict-free replicated set data type". *Distributed Computing Systems (ICDCS), 2013 IEEE 33rd International Conference on*, 186–195. IEEE, 2013.

[35] Devegili, Augusto Jun, Michael Scott, and Ricardo Dahab. "Implementing cryptographic pairings over Barreto-Naehrig curves". *Pairing-Based Cryptography–Pairing 2007*, 197–207. Springer, 2007.

[36] Ellis, Clarence A and Simon J Gibbs. "Concurrency control in groupware systems". *ACM SIGMOD Record*, 18(2):399–407, 1989.

[37] Feldman, Ariel J, William P Zeller, Michael J Freedman, and Edward W Felten. "SPORC: Group Collaboration using Untrusted Cloud Resources." *OSDI*, 337–350. 2010.

[38] Fickus, Matthew C. "Lecture Notes for MATH 631: Algebraic Structures", 2014.

[39] Figliola, Andrews and Fischer. "Department of Defense Implementation of the Federal Data Center Consolidation Initiative", April 2013.

[40] Fischer and Figliola. "Overview and Issues for Implementation of the Federal Cloud Computing Initiative", April 2013.

[41] Freeman, David. "Constructing pairing-friendly elliptic curves with embedding degree 10". *Algorithmic number theory*, 452–465. Springer, 2006.

[42] Freeman, David, Michael Scott, and Edlyn Teske. "A taxonomy of pairing-friendly elliptic curves". *Journal of Cryptology*, 23(2):224–280, 2010.

[43] Galbraith, Steven D, Kenneth G Paterson, and Nigel P Smart. "Pairings for cryptographers". *Discrete Applied Mathematics*, 156(16):3113–3121, 2008.

[44] (GAO), Government Accounting Office. "Progress Made but Future Cloud Computing Efforts Should be Better Planned", 2012.

[45] Gentry, Craig. *A fully homomorphic encryption scheme*. Ph.D. thesis, Stanford University, 2009.

[46] Gentry, Craig and Shai Halevi. "Implementing Gentry's fully-homomorphic encryption scheme". *Advances in Cryptology–EUROCRYPT 2011*, 129–148. Springer, 2011.

[47] Gentry, Craig and Alice Silverberg. "Hierarchical ID-based cryptography". *Advances in cryptology–ASIACRYPT 2002*, 548–566. Springer, 2002.

[48] GNU. "Words to Avoid (or Use with Care) Because They Are Loaded or Confusing", 2014. URL https://www.gnu.org/philosophy/words-to-avoid.html.

[49] Goldwasser, Shafi and Silvio Micali. "Probabilistic encryption". *Journal of computer and system sciences*, 28(2):270–299, 1984.

[50] Golle, Philippe, Jessica Staddon, and Brent Waters. "Secure conjunctive keyword search over encrypted data". *Applied Cryptography and Network Security*, 31–45. Springer, 2004.

[51] Google. "Google Ngram Viewer". URL https://books.google.com/ngrams.

[52] Goyal, Vipul, Abhishek Jain, Omkant Pandey, and Amit Sahai. "Bounded ciphertext policy attribute based encryption". *Automata, Languages and Programming*, 579–591. Springer, 2008.

[53] Gregg, John Aaron. "On factoring integers and evaluating discrete logarithms". *Harvard University*, 2003.

[54] Grobauer, Bernd, Tobias Walloschek, and Elmar Stocker. "Understanding cloud computing vulnerabilities". *Security & Privacy, IEEE*, 9(2):50–57, 2011.

[55] GSA. "FedRAMP Concept of Operations (CONOPS)", February 2012.

[56] Gupta, Tanya. "The Attack on Treasury: Cloud Computing and the Implications on Cybersecurity", May 2010. URL http://www.examiner.com/article/the-attack-on-treasury-cloud-computing-and-the-implications-on-cybersecurity.

[57] Holzmann, Gerard J. "The model checker SPIN". *IEEE Transactions on software engineering*, (5):279–295, 1997.

[58] Hu, Vincent C, David Ferraiolo, Rick Kuhn, Adam Schnitzer, Kenneth Sandlin, Robert Miller, and Karen Scarfone. "Guide to Attribute Based Access Control (ABAC) Definition and Considerations". *NIST Special Publication*, 800:162, 2014.

[59] Ion, Mihaela, Giovanni Russello, and Bruno Crispo. "Supporting publication and subscription confidentiality in pub/sub networks". *Security and Privacy in Communication Networks*, 272–289. Springer, 2010.

[60] Ion, Mihaela, Jianqing Zhang, and Eve M Schooler. "Toward content-centric privacy in ICN: attribute-based encryption and routing". *ACM SIGCOMM Computer Communication Review*, volume 43, 513–514. ACM, 2013.

[61] Jeong, Ik Rae and Jeong Ok Kwon. "Analysis of some keyword search schemes in encrypted data". *Communications Letters, IEEE*, 12(3):213–215, 2008.

[62] Joux, Antoine. "A new index calculus algorithm with complexity L (1/4+ o (1)) in very small characteristic." *IACR Cryptology ePrint Archive*, 2013:95, 2013.

[63] Katz, Jonathan and Yehuda Lindell. *Introduction to Modern Cryptography*. Chapman & Hall CRC, 2008.

[64] Kundra, Vivek. "In the Cloud", 2009. URL http://www.whitehouse.gov/blog/Streaming-at-100-In-the-Cloud.

[65] Kundra, Vivek. "25 Point Implementation Plan to Reform Federal Information Technology Management", 2010.

[66] Kundra, Vivek. "Federal Data Center Consolidation Initiative", February 2010.

[67] Kundra, Vivek. "Federal cloud computing strategy", 2011.

[68] Lamport, Leslie. *Specifying systems: the TLA+ language and tools for hardware and software engineers*. Addison-Wesley Longman Publishing Co., Inc., 2002.

[69] Lewko, Allison, Tatsuaki Okamoto, Amit Sahai, Katsuyuki Takashima, and Brent Waters. "Fully secure functional encryption: Attribute-based encryption and (hierarchical) inner product encryption". *Advances in Cryptology–EUROCRYPT 2010*, 62–91. Springer, 2010.

[70] Lewko, Allison and Brent Waters. "Decentralizing attribute-based encryption". *Advances in Cryptology–EUROCRYPT 2011*, 568–588. Springer, 2011.

[71] Lewko, Allison and Brent Waters. "Unbounded HIBE and attribute-based encryption". *Advances in Cryptology–EUROCRYPT 2011*, 547–567. Springer, 2011.

[72] Lin, Huang, Zhenfu Cao, Xiaohui Liang, and Jun Shao. "Secure threshold multi authority attribute based encryption without a central authority". *Information Sciences*, 180(13):2618–2632, 2010.

[73] Mao, Wenbo. *Modern Cryptography: Theory & Practice*. Hewlett-Packard Books, 2004.

[74] Martin, Luther. *Introduction to Identity-Based Encryption*. Artech House, 2008.

[75] Maurer, Ueli. "Abstract models of computation in cryptography". *Cryptography and Coding*, 1–12. Springer, 2005.

[76] McClure, David. "Cloud Computing (Testimony from Dr. David McClure)". URL http://www.gsa.gov/portal/content/159101.

[77] Mecella, Massimo, Mourad Ouzzani, Federica Paci, and Elisa Bertino. "Access control enforcement for conversation-based web services". *Proceedings of the 15th international conference on World Wide Web*, 257–266. ACM, 2006.

[78] Metheny, Matthew. *Federal Cloud Computing: The Definitive Guide for Cloud Service Providers*. Syngress, 225 Wyman Street, Waltham, MA 02451, USA, first edition, 2013.

[79] Montalbano, Elizabeth. "Amazon Cloud Outage Didn't Stop Recovery.gov", April 2011. URL http://www.informationweek.com/government/cloud-saas/amazon-cloud-outage-didnt-stop-recoveryg/229402174.

[80] Nishide, Takashi, Kazuki Yoneyama, and Kazuo Ohta. "Attribute-based encryption with partially hidden encryptor-specified access structures". *Applied cryptography and network security*, 111–129. Springer, 2008.

[81] NIST. "NIST SP 800-33: Underlying Technical Models for Information Technology Security", 2002.

[82] NIST. "NIST SP 800-162: Guide to Attribute Based Access Control (ABAC) Definition and Considerations (Draft)", 2013.

[83] NIST, SP. "800-145: The NIST definition of cloud computing", 2012.

[84] Oliveira, Leonardo B, Diego F Aranha, Conrado PL Gouvêa, Michael Scott, Danilo F Câmara, Julio López, and Ricardo Dahab. "TinyPBC: Pairings for authenticated identity-based non-interactive key distribution in sensor networks". *Computer Communications*, 34(3):485–493, 2011.

[85] Paillier, Pascal. "Public-key cryptosystems based on composite degree residuosity classes". *Advances in cryptology–EUROCRYPT'99*, 223–238. Springer, 1999.

[86] Pal, Partha, Greg Lauer, Joud Khoury, Nick Hoff, and Joe Loyall. "P3S: A privacy preserving publish-subscribe middleware". *Middleware 2012*, 476–495. Springer, 2012.

[87] Papadopoulos, Panagiotis, Antonis Papadogiannakis, Michalis Polychronakis, Apostolis Zarras, Thorsten Holz, and Evangelos P Markatos. "k-subscription: Privacy-Preserving Microblogging Browsing Through Obfuscation". *Proceedings of the 29th Annual Computer Security Applications Conference*, 49–58. ACM, 2013.

[88] Pereira, Geovandro CCF, Marcos A Simplício Jr, Michael Naehrig, and Paulo SLM Barreto. "A family of implementation-friendly BN elliptic curves". *Journal of Systems and Software*, 84(8):1319–1326, 2011.

[89] Ramanna, Somindu C and Palash Sarkar. "Anonymous constant-size ciphertext HIBE from asymmetric pairings". *Cryptography and Coding*, 344–363. Springer, 2013.

[90] Ramanna, Somindu C and Palash Sarkar. "Anonymous HIBE from Standard Assumptions over Type-3 Pairings using Dual System Encryption." *IACR Cryptology ePrint Archive*, 2013:528, 2013.

[91] Review, National Performance. "Improve Government's Information Infrastructure", 1993. URL http://govinfo.library.unt.edu/npr/library/reports/it09.html.

[92] Rhee, Hyun Sook, Ik Rae Jeong, Jin Wook Byun, and Dong Hoon Lee. "Difference set attacks on conjunctive keyword search schemes". *Secure Data Management*, 64–74. Springer, 2006.

[93] Rivest, Ronald L, Adi Shamir, and Len Adleman. "A method for obtaining digital signatures and public-key cryptosystems". *Communications of the ACM*, 21(2):120–126, 1978.

[94] Sahai, Amit and Brent Waters. "Fuzzy identity-based encryption". *Advances in Cryptology–EUROCRYPT 2005*, 457–473. Springer, 2005.

[95] Shamir, Adi. "Identity-based cryptosystems and signature schemes". *Advances in cryptology*, 47–53. Springer, 1985.

[96] Song, Dawn Xiaoding, David Wagner, and Adrian Perrig. "Practical techniques for searches on encrypted data". *Security and Privacy, 2000. S&P 2000. Proceedings. 2000 IEEE Symposium on*, 44–55. IEEE, 2000.

[97] Spires, Richard. "Testimony of Mr. Richard Spires Chief Information Officer U.S. Department of Homeland Security Before the Committee on Homeland Security Subcommittee on Cybersecurity, Infrastructure Protection, and Security Technologies", 2011.

[98] Stehlé, Damien and Ron Steinfeld. "Faster fully homomorphic encryption". *Advances in Cryptology–ASIACRYPT 2010*, 377–394. Springer, 2010.

[99] Stewart, Kyle, Michael Clark, Kenneth Hopkinson, and Todd Andel. *MA-AHASBE: Multi-Authority Anonymous Hierarchical Attribute-Set Based Encryption from Asymmetric Pairings*. Technical Report 88ABW-2014-4193, Air Force Institute of Technology, 2015.

[100] Takai, Teresa. *DoD Cloud Computing Strategy*. Technical report, US Department of Defense, 2012.

[101] Tariq, Muhammad Adnan, Boris Koldehofe, and Kurt Rothermel. "Securing broker-less publish/subscribe systems using identity-based encryption". *Parallel and Distributed Systems, IEEE Transactions on*, 25(2):518–528, 2014.

[102] Treasury. "Department of the Treasury Data Center Consolidation Plan", September 2011.

[103] Van Dijk, Marten and Ari Juels. "On the Impossibility of Cryptography Alone for Privacy-Preserving Cloud Computing." *IACR Cryptology ePrint Archive*, 2010:305, 2010.

[104] VanRoekel, Steven. "Security Authorization of Information Systems in Cloud Computing Environments", 2011.

[105] Wan, Zhiguo, Jun'e Liu, and Robert H Deng. "HASBE: A Hierarchical Attribute-Based Solution for Flexible and Scalable Access Control in Cloud Computing". *Information Forensics and Security, IEEE Transactions on*, 7(2):743–754, 2012.

[106] Wang, Cong, Ning Cao, Jin Li, Kui Ren, and Wenjing Lou. "Secure ranked keyword search over encrypted cloud data". *Distributed Computing Systems (ICDCS), 2010 IEEE 30th International Conference on*, 253–262. IEEE, 2010.

[107] Wang, Cong, Ning Cao, Kui Ren, and Wenjing Lou. "Enabling secure and efficient ranked keyword search over outsourced cloud data". *Parallel and Distributed Systems, IEEE Transactions on*, 23(8):1467–1479, 2012.

[108] Wang, Guojun, Qin Liu, and Jie Wu. "Hierarchical attribute-based encryption for fine-grained access control in cloud storage services". *Proceedings of the 17th ACM conference on Computer and communications security*, 735–737. ACM, 2010.

[109] Waters, Brent. “Dual system encryption: Realizing fully secure IBE and HIBE under simple assumptions”. *Advances in Cryptology–CRYPTO 2009*, 619–636. Springer, 2009.

[110] Zhang, Ye, Chun Jason Xue, Duncan S Wong, Nikos Mamoulis, and Siu Ming Yiu. “Acceleration of composite order bilinear pairing on graphics hardware”. *Information and Communications Security*, 341–348. Springer, 2012.

[111] Zhou, Lan, Vijay Varadharajan, and Michael Hitchens. “Achieving Secure Role-Based Access Control on Encrypted Data in Cloud Storage”. *IEEE Transactions on Information Forensics and Security*, 8(11-12):1947–1960, 2013.

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)*	2. REPORT TYPE	3. DATES COVERED *(From — To)*
17–09–2015	Doctoral Dissertation	Sep 2012–Sep 2015

4. TITLE AND SUBTITLE

Novel Techniques for Secure Use of
Public Cloud Computing Resources

5a. CONTRACT NUMBER

5b. GRANT NUMBER

5c. PROGRAM ELEMENT NUMBER

6. AUTHOR(S)

Stewart, Kyle E., Captain, USAF

5d. PROJECT NUMBER

5e. TASK NUMBER

5f. WORK UNIT NUMBER

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

Air Force Institute of Technology
Graduate School of Engineering and Management (AFIT/EN)
2950 Hobson Way
WPAFB, OH 45433-7765

8. PERFORMING ORGANIZATION REPORT NUMBER

AFIT-ENG-DS-15-S-018

9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)

Intentionally Left Blank

10. SPONSOR/MONITOR'S ACRONYM(S)

11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION / AVAILABILITY STATEMENT

DISTRIBUTION STATEMENT A:
APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED

13. SUPPLEMENTARY NOTES

This work is declared a work of the U.S. Government and is not subject to copyright protection in the United States.

14. ABSTRACT

The federal government has an expressed interest in moving data and services to third party service providers in order to take advantage of the flexibility, scalability, and potential cost savings. This approach is called cloud computing. The thesis for this research is that efficient techniques exist to support the secure use of public cloud computing resources by a large, federated enterprise. The primary contributions of this research are the novel cryptographic system MA-AHASBE (Multi-Authority Anonymous Hierarchical Attribute-Set Based Encryption), and the techniques used to incorporate MA-AHASBE in a real world application. Performance results indicate that while there is a cost associated with enforcing the suggested security model, the cost is not unreasonable and the benefits in security can be significant. The contributions of this research give the DoD additional tools for supporting the mission while taking advantage of the cost efficient public cloud computing resources that are becoming widely available.

15. SUBJECT TERMS

cryptography, cloud computing, attribute-based encryption, public key infrastructure publish-subscribe

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	c. THIS PAGE			Dr. Kenneth M. Hopkinson (ENG)
U	U	U	UU	187	19b. TELEPHONE NUMBER *(include area code)* (937) 255-3636 x4579 kenneth.hopkinson@afit.edu