NAVAL POSTGRADUATE SCHOOL

MONTEREY, CALIFORNIA

THESIS

PERFORMANCE OF HYBRID SIGNATURES FOR PUBLIC KEY INFRASTRUCTURE CERTIFICATES

by

John Lytle

June 2021

Thesis Advisor: Britta Hale
Second Reader: Chad A. Bollmann

THIS PAGE INTENTIONALLY LEFT BLANK

REPORT DOCUMENTATION PAGE	*Form Approved OMB No. 0704-0188*

1. AGENCY USE ONLY *(Leave blank)*	**2. REPORT DATE** June 2021	**3. REPORT TYPE AND DATES COVERED** Master's thesis

4. TITLE AND SUBTITLE PERFORMANCE OF HYBRID SIGNATURES FOR PUBLIC KEY INFRASTRUCTURE CERTIFICATES	**5. FUNDING NUMBERS**
6. AUTHOR(S) John Lytle	
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Naval Postgraduate School Monterey, CA 93943-5000	**8. PERFORMING ORGANIZATION REPORT NUMBER**
9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES) N/A	**10. SPONSORING / MONITORING AGENCY REPORT NUMBER**

11. SUPPLEMENTARY NOTES The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.

12a. DISTRIBUTION / AVAILABILITY STATEMENT Approved for public release. Distribution is unlimited.	**12b. DISTRIBUTION CODE** A

13. ABSTRACT (maximum 200 words)

The modern public key infrastructure (PKI) model relies on digital signature algorithms to provide message authentication, data integrity, and non-repudiation. To provide this, digital signature algorithms, like most cryptographic schemes, rely on a mathematical hardness assumption for provable security. As we transition into a post-quantum era, the hardness assumptions used by traditional digital signature algorithms are increasingly at risk of being solvable in polynomial time. This renders the entirety of public key cryptography, including digital signatures, vulnerable to being broken. Hybrid digital signature schemes represent a potential solution to this problem. In this thesis, we provide the first test implementation of true hybrid signature algorithms. We evaluate the viability and performance of several hybrid signature schemes against traditional hybridization techniques via standalone cryptographic operations. Finally, we explore how hybrid signatures can be integrated into existing X.509 digital certificates and examine their performance by integrating both into the Transport Layer Security 1.3 protocol.

14. SUBJECT TERMS hybrid digital certificates, public key cryptography, public key infrastructure, PKI	**15. NUMBER OF PAGES** 147
	16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	**18. SECURITY CLASSIFICATION OF THIS PAGE**	**19. SECURITY CLASSIFICATION OF ABSTRACT**	**20. LIMITATION OF ABSTRACT**
Unclassified	Unclassified	Unclassified	UU

NSN 7540-01-280-5500

Standard Form 298 (Rev. 2-89)
Prescribed by ANSI Std. 239-18

THIS PAGE INTENTIONALLY LEFT BLANK

Approved for public release. Distribution is unlimited.

PERFORMANCE OF HYBRID SIGNATURES FOR PUBLIC KEY INFRASTRUCTURE CERTIFICATES

John Lytle
Gunnery Sergeant, United States Marine Corps
BS, American Military University, 2017

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN APPLIED CYBER OPERATIONS

from the

NAVAL POSTGRADUATE SCHOOL
June 2021

Approved by: Britta Hale
Advisor

Chad A. Bollmann
Second Reader

Alex Bordetsky
Chair, Department of Information Sciences

THIS PAGE INTENTIONALLY LEFT BLANK

ABSTRACT

The modern public key infrastructure (PKI) model relies on digital signature algorithms to provide message authentication, data integrity, and non-repudiation. To provide this, digital signature algorithms, like most cryptographic schemes, rely on a mathematical hardness assumption for provable security. As we transition into a post-quantum era, the hardness assumptions used by traditional digital signature algorithms are increasingly at risk of being solvable in polynomial time. This renders the entirety of public key cryptography, including digital signatures, vulnerable to being broken. Hybrid digital signature schemes represent a potential solution to this problem. In this thesis, we provide the first test implementation of true hybrid signature algorithms. We evaluate the viability and performance of several hybrid signature schemes against traditional hybridization techniques via standalone cryptographic operations. Finally, we explore how hybrid signatures can be integrated into existing X.509 digital certificates and examine their performance by integrating both into the Transport Layer Security 1.3 protocol.

THIS PAGE INTENTIONALLY LEFT BLANK

Table of Contents

List of Figures

List of Tables

THIS PAGE INTENTIONALLY LEFT BLANK

List of Acronyms and Abbreviations

AEAD	Authenticated Encryption with Associated Data
API	Application Programming Interface
ASN.1	Abstract Syntax Notation One
AVX2	Advanced Vector Extensions 2
BMI	Bit Manipulation Instruction Set
CA	Certificate Authority
CDN	Content Delivery Network
CRL	Certificate Revocation List
CMA	Chosen Message Attack
CMS	Cryptographic Message Syntax
CPU	Central Processing Unit
DER	Distinguished Encoding Rules
DHE	Diffie-Hellman Ephemeral
DNSSEC	Domain Name System Security Extensions
DOD	Department of Defense
DSA	Digital Signature Algorithm
DSS	Digital Signature Standard
DTU	Technische Universität Darmstadt
ECDSA	Elliptic Curve Digital Signature Algorithm

ECDHE Elliptic Curve Diffie-Hellman Ephemeral

EUF Existential Unforgeability

FS Fiat-Shamir

HFE Hidden Field Equations

IETF Internet Engineering Task Force

ITU International Telecommunication Union

LAMPS Limited Additional Mechanisms for PKIX and SMIME

LTS Long Term Support

LWE Learning With Errors

MASINT Measurement and Signature Intelligence

MLWE Module Learning With Errors

MIT Massachusetts Institute of Technology

MSIS Module Short Integer Solution

NIST National Institute of Standards and Technology

NPS Naval Postgraduate School

NTP Network Time Protocol

OID Object Identifier

OCSP Online Certificate Status Protocol

OQS Open Quantum Safe

PEM Privacy Enhanced Mail

PKCS Public-Key Cryptography Standards

PKI Public Key Infrastructure

PQ	Post-Quantum
PQC	Post-Quantum Cryptography
QROM	Quantum Random Oracle Model (ROM)
R-LWE	Ring Learning With Errors (LWE)
RDTSC	Read Time-Stamp Counter
RDTSCP	Read Time-Stamp Counter and Processor ID
ROM	Random Oracle Model
SCT	Signed Certificate Timestamp
SIS	Short Integer Solution
SRWBR	Short Range Wide Band Radio
SSH	Secure Shell
SUF	Strong Unforgeability
TCP	Transmission Control Protocol
TLS	Transport Layer Security
TSC	Timestamp Counter
UDP	User Datagram Protocol
USG	United States Government
USN	U.S. Navy
UUF	Universal Forgery
XOF	Extendable-Output Function

THIS PAGE INTENTIONALLY LEFT BLANK

CHAPTER 1:
Introduction

Digital signatures are a fundamental cryptographic primitive used to create a digital counterpart to traditional, handwritten signatures. When verified successfully, a digital signature provides the recipient of a message a strong guarantee that the message was created by the original signer and that the integrity of the signed data was not compromised in transit [1]. Cryptographic protocols that require these properties incorporate digital signature schemes into their design. Popular examples of this include the Transport Layer Security (TLS) [2] and Secure Shell (SSH) [3] protocols that use digital signature algorithms for authentication.

Like most cryptographic schemes, every digital signature algorithm relies on a mathematical hardness problem for provable security. A hardness problem is one that cannot be solved efficiently in polynomial-time and is what guarantees a system is secure assuming a computationally limited adversary [4]. Hardness problems vary between signature algorithms, and traditional digital signature schemes rely on a single underlying hardness problem for security. At this time, no known theorem formally proves unconditional hardness for any problem, so a hardness problem is valid only based on the confidence and acceptance of previous research [5], [6]. With recent advancements in quantum computing, efficient algorithms that solve well-known hardness problems are becoming increasingly viable [7]. This has the potential to render the entirety of public key cryptography, to include digital signatures, vulnerable to being broken unless cryptographic schemes adopt different hardness problems.

Several potential candidates have been identified within public key cryptography that are resilient to quantum attacks; however, the transition away from vulnerable legacy algorithms to those considered secure in a Post-Quantum (PQ) world is a non-trivial task. First and foremost, a survey of systems that use legacy algorithms is needed to identify and understand the requirements that new algorithms need to satisfy. These systems include everything from standalone communication devices to operating system internals and encompass both hardware and software-based solutions. Additionally, the properties and characteristics of new candidates need to be evaluated and compared to legacy algorithms to identify any differences in performance and reliability. This entire process is extremely disruptive and

often takes several years to complete [8].

In anticipation of a PQ transition, organizations like the National Institute of Standards and Technology (NIST) have initiated processes to evaluate and standardize "quantum-resistant public-key cryptographic algorithms" [9]. As part of this multiyear effort, several PQ digital signature algorithms have been identified with each relying on different families of hardness problems. Unlike their classical counterparts, these problems do not have the same amount of historical research and are considered relatively new to applied cryptography. This creates a point of friction for the transition from classical to PQ signature algorithms because, if an efficient attack against either the algorithm or the hardness problem is found, the security of the entire cryptosystem would be broken.

Further complicating a seamless transition, digital signature algorithms have undergone relatively few changes compared to the rest of public key cryptography since their inception in the 1970s [10]. While strides have been made in the standardization and adoption of asymmetric key establishment schemes, the industry has been slow to adopt changes. Even with the introduction of more efficient and secure classical options like elliptic curve cryptography, digital signatures still primarily rely on the original RSA signature scheme for everything from code signing to certificates issued over the Internet. As a result, digital signatures and the infrastructure that relies on their security to function have formed a fragile, monolithic ecosystem on which many fundamental Internet applications, to include e-commerce and secure messaging, are based. If a scheme is found to be weakened or broken, the entirety of the system must shift to accommodate either changes to the signature scheme in use or the wholesale adoption of a new one. Any transition is limited in how quickly it can be implemented due to the complexity and distributed nature of Public Key Infrastructure (PKI), the bureaucratic process of standardization and government regulations, and the legal restrictions stemming from patents.

Whether due to the monolithic nature of existing cryptosystems or through a reluctance to adopt newer signature algorithms, a gap exists between the infrastructure and software deployed now and what will be needed in the near future. There are multiple approaches to solve this problem; however, hybridization, or the secure combination of multiple digital signature algorithms into a unified scheme, simultaneously solves both. A composable hybrid signature scheme combines its component algorithms in such a way that the entire

scheme does not rely on the hardness problem of a single algorithm. When implemented correctly, any security guarantee achieved by a component algorithm is also achieved by the hybrid scheme and, even if one of the component algorithms is later broken, the overall hybrid signature scheme still achieves the security of the unbroken component algorithm. As part of a PQ transition, a cryptosystem can combine a PQ signature algorithm with a classical signature algorithm via a hybrid scheme. This flexibility allows system engineers and developers to hedge their bets when adopting newer signature algorithms and for cryptographers to design custom solutions to specific problems like long-term security. Additionally, hybrid signature schemes prevent a monolithic ecosystem from reoccurring after the adoption of the next standardized PQ signature algorithms. Given the slow pace of change in adopting newer signature algorithms, a hybrid signature scheme can be leveraged to allow the introduction of newer alternatives without interrupting the overall security of a cryptosystem.

The characteristics, efficacy, and performance of hybrid digital signature schemes first need to be evaluated before they can be used to solve pending problems in the PQ transition. This work seeks to address these factors by examining the performance of various hybrid signature schemes using both classical and PQ signature algorithms. This information can be used to create a baseline that identifies the operational requirements of different signature algorithm combinations and informs early adopters of the potential limitations of hybridization as a transition mechanism.

1.1 Related Work

As a concept, hybrid signature schemes are not new to public key cryptography. Several schemes [11]–[14] have been designed and patented over the course of history; however, most were designed to meet niche requirements and no examples can be found in widespread use. Additionally, modern cryptographic libraries do not typically offer hybrid or dual mode support due to the increased costs, complexity, and lack of standardization [15], [16].

Renewed interest in hybridization has followed with the imminent standardization of PQ signature algorithms. Due to the increased size of the signatures and keys used by PQ algorithms when compared to their classical counterparts, researchers have examined non-hybrid PQ signature algorithms within specific use cases [16]–[18] to include the PKI

system [19]. This research is critical in understanding the viability of both PQ algorithms within existing standards, libraries, and protocols and establishing the constraints on which a hybrid system would be based.

Bindel et al. [15] establish hybrid digital signature schemes as a potential mechanism to facilitate the PQ transition within PKI. Specifically, their paper identifies different ways to combine component algorithms, provides conditions on when the resulting scheme is unforgeable, and offers an informal definition of non-separability that is unique to hybrid signature schemes. Extending that work, the Collaborative Research Center CROSSING at TU Darmstadt implemented backwards compatible hybrid certificates in two popular cryptographic libraries [20], [21].

1.2 Contribution

This thesis presents a detailed evaluation of several hybrid digital signature schemes by analyzing the quantitative performance at the algorithmic level and in protocol use (i.e., TLS). Several different methods for hybridization are examined to include simple concatenation [22], nesting [15], and newly introduced true hybrid schemes [23]. These schemes are then implemented with the end goal to test the viability and performance of each method.

At the algorithmic level, the component digital signature algorithms are chosen using constraints imposed by the hybrid signature scheme. Only algorithms that are currently approved by NIST or being evaluated in Round 2 and 3 of the call for PQ standardization are considered. Additions or alternations to component algorithms are clearly identified; however, an independent security review of those changes is considered out-of-scope for this paper. All changes should be considered experimental.

Given its ubiquity in previous research [18], the TLS protocol is used to test the impact of hybridization on performance at the protocol level. Specifically, different digital certificate structures are examined based on ease-of-implementation, security properties, and existing patents. Once implemented, each certificate structure is evaluated using TLS handshake latency as a metric. All software created during the course of the thesis was developed using publicly available cryptographic libraries and made available as a proof-of-concept only.

1.3 Overview

Chapter 2 provides a background on digital signatures, their security notions, and design history. This chapter also introduces the signature algorithms used in the rest of the paper. Chapter 3 delves into the construction of hybrid signature schemes and security notions unique to hybridization. Chapter 4 introduces the methodology used to implement the true hybrid schemes. Chapter 5 details methods for constructing hybrid digital certificates and specifies how they are used in the TLS protocol handshake. Finally, Chapters 6 and 7 establish testing methodology and publish both the results and their analysis.

THIS PAGE INTENTIONALLY LEFT BLANK

CHAPTER 2:
Background

Digital signatures continue to play a critical role in public key (i.e., asymmetric) cryptography since the introduction of the latter in 1976 [10]. The primary purpose of digital signatures is to replicate the authentication provided by their traditional, handwritten counterparts. Whereas the authenticity of a traditional signature is verified visually and inherently vulnerable to tampering, digital signatures are verified through a cryptographic scheme. They extend the functionality of traditional signatures by providing a mechanism for message authentication and non-repudiation. As a cryptographic primitive, digital signatures form an integral part of the foundation on which secure communication over the Internet is accomplished. This chapter briefly introduces digital signatures, provides a history of digital signature design, and details the classical and post-quantum digital signature schemes examined in subsequent chapters.

2.1 Definition of a Digital Signature Algorithm

In their simplest form, digital signatures are "the result of a cryptographic transformation that, when properly implemented, provide a mechanism for verifying authentication, data integrity and non-repudiation" [24]. Other conventional authentication techniques like message authentication codes exist that cover a subset of these properties; however digital signatures are unique in that they provide all three. A digital signature scheme consists of three components:

- key generation algorithm (KeyGen)
- signature algorithm (Sign)
- verification algorithm (Verify).

The KeyGen is a probabilistic, polynomial time algorithm that allows a user to create a pair of matching public (pk) and secret (sk) keys based on input (1^λ). The security parameter (λ) specifies various quantities to include signature length, security level, and any message length limitations that are required for the scheme's correctness and security [1]. In general, λ is defined by the scheme's original authors or through standardization and is subject to

change over time if efficient attacks are discovered or a higher level of security is required.

The Sign operation is a potentially probabilistic, polynomial time algorithm that takes a variable input message (m) and, using a sk, creates a unique output string known as the signature (σ) on m that is both associated only with the original input and the originating signer.

The Verify operation is a polynomial time algorithm that validates that the message was signed by the originating signatory by applying the pk to the attached signature. Verify accepts three inputs (m, pk, σ) and either *accepts* or *rejects* the signature as an output. For consistency, this operation must be deterministic such that it is not possible for Verify to accept a signature in one instance but reject it in another.

2.1.1 Security Notions

If message authentication, data integrity, and non-repudiation are the goals of a digital signature scheme then the underlying hardness assumption, adversary model, and forgery definition are the tools that measure if the scheme accomplishes those goals. Before a digital signature scheme can be declared secure, the scheme designers must first clearly identify the hardness assumption used, the adversary's capabilities and constraints, and the probability an adversary can win under those constraints.

Hardness Assumptions

Fundamentally, the security of digital signature schemes depend on a reduction to a well-defined computational hardness assumption. This assumption is based on a particular mathematical problem where no known polynomial time algorithm exists that can solve it (e.g. NP-hardness) [4], [25]. The hard problems used in a scheme can also be sub-problems of other hardness assumptions, thus creating families or categories of related hard problems [6]. Through a security reduction, a sub-problem can be shown to be at least as difficult to break as the original hard problem. For example, the RSA signature scheme, originally conceived in 1976, uses the RSA problem ($RSAP$) as its hardness assumption [25]. $RSAP$ is a sub-problem of integer factorization ($RSAP \leq_P FACTORING$) and has been found to be intractable under certain conditions such as a sufficiently large key size [26], [27].

Classic digital signature schemes like DSA, ECDSA, and RSA use sub-problems of integer factorization or the discrete logarithm problem [25], [28], [29]. These sub-problems are widely accepted to be resistant to solutions based on efficient polynomial time algorithms running on modern classical computers; however, quantum computers through variations of Shor's algorithm [9] can theoretically efficiently calculate both discrete logarithms and perform integer factorization. Recent research [30] suggests that a quantum computer would need at least 20 million qubits to factor a 2048 bit RSA integer; however, current technological constraints limit the development of quantum computers to less than 60 qubits [31]. Security researchers currently predict a timeline of roughly 20-30 years before quantum computers become viable threats to current standards [32].

The time until the quantum threat is realized is subject to rapid change as advancements in classical processing power, quantum computing, or the discovery of more efficient algorithms will accelerate this timeline. In 2019, two researchers demonstrated that only a 20-million qubit quantum computer would be needed to factor a 2048-bit RSA public key in an eight-hour period [30]. Prior to this research, the previously recognized number of qubits required for the same challenge was over a billion. Shor even speculated that "quantum algorithms could even break RSA on a quantum computer asymptotically faster than encrypting with RSA on a classical computer" [7].

As cryptography transitions into a PQ world, new hardness assumptions are required to replace those vulnerable to a quantum-capable adversary. Research into areas such as hash-based, code-based, lattice, and multivariate polynomial cryptography have yielded several promising candidates that address attacks from both classical and quantum computers [8]. While some of these areas have been studied for several years, they have not received the same level of scrutiny as classical hard problems. Compared to classical algorithms like RSA, the hardness assumptions used by PQ algorithms are more complex and are thus open to subtle attacks that are equally as complex. For example, Rainbow [33], a multivariate-based signature scheme introduced in 2005, has been subjected to direct and indirect attacks [34]–[36] as a result of general research into the security of multivariate signature schemes. As expressed by Petzoldt, Bulygin, and Buchmann [37], the complexities of these attacks increases the difficulty in selecting valid security parameters that protect against both quantum and classical adversaries. As mentioned in Chapter 1, increased complexity of PQ signature algorithms design is one of the motivations behind hybrid signature schemes.

Unforgeability

It is not enough to show the intractability of a hard problem. Digital signature schemes must also achieve both soundness and completeness. In other words, a scheme must always successfully verify a message legitimately signed by the sk using the Sign algorithm and reject all illegitimate signatures. If an adversary is able to efficiently (e.g. within polynomial time) attack a scheme in a way that violates this principle, it becomes trivial for them to forge a digital signature for a message without having access to the signer's key, thus breaking the security of the scheme.

While the concept is simple, different levels of security may be required depending on the application. For example, it may be sufficient that while an adversary is able to forge one message, they are unable to forge a specific message of their choice. As such, the conditions necessary for a successful attack and the advantages afforded to the adversary (e.g., abilities, rules, etc.) must be clearly defined via an experiment between an honest participant and an attacker (i.e., in a game).

In 1988, Goldwasser, Micali, and Rivest proposed a hierarchy of security levels of success that an adversary must be able to achieve with non-negligible probability using a chosen-message attack that runs in probabilistic polynomial time [38]. Out of these definitions, Existential Unforgeability (EUF) requires the least level of success from the adversary while simultaneously providing the most advantages.

Existential Unforgeability under Chosen-Message Attack (EUF-CMA) requires that an adversary forge a signature for at least one message. This message is chosen by the attacker and can be meaningless (e.g. random). Furthermore, the message must never have been validly signed by the signer. We present the EUF experiment from [39] in the following formalized manner:

Security [39]. Let us recall the *existential unforgeability against chosen message attacks* (EUF–CMA) security experiment [38], played between a challenger and a forger $\mathcal{A}$.

1. The forger, on input public parameters Π, may ask a *non-adaptive* chosen–message query. To this end, it submits a list of messages $M^{(1)}, ..., M^{q_0}$ to the challenger.
2. The challenger runs Sig.Gen(Π) to generate a keypair (vk, sk). The forger receives vk and a signature $\sigma^{(i)}$ for each chosen message $M^{(i)}, i \in [q_0]$.

3. Now the forger may ask *adaptive* chosen–message queries. Each query consists of a message $M^{(i)}, i \in [q_0 + 1, q]$, and is answered by the challenger with a signature $\sigma^{(i)}$ under sk for message $M^{(i)}$.
4. Finally, the forger outputs a message M^* and signature σ^*.

Definition 1 ([39])**.** An adversary is *adaptive*, if it asks at least one adaptive chosen–message query. Otherwise, it is *non-adaptive*. Let $\mathcal{A}$ be an adversary (adaptive or non–adaptive) that runs in time t, makes q chosen–message queries (in total), and outputs (M^*, σ^*). We say that $\mathcal{A}(\epsilon, t, q)$–breaks the EUF-CMA security of Sig if

$$\Pr[\text{Sig.Vfy}(vk, M^*, \sigma^*) = 1 \wedge M^* \notin \{M^{(1)}, ..., M^{(q)}\}] \geq \epsilon.$$

Other variants such as Strong EUF-CMA (SUF-CMA) give the adversary more capabilities and thus provide a stronger security guarantee. For example, SUF-CMA requires that the adversary output a message *and* signature pair that is not one of the previous queries. This simple difference prevents the adversary from randomizing a valid signature obtained from a query in a way that retains validity while looking different from the original signature [40]. Each digital signature algorithm must achieve a variant of EUF-CMA under specific security parameters to be considered for standardization by the NIST [24], [41].

2.1.2 Fiat–Shamir Transform

The Fiat-Shamir (FS) transform is a general technique for transforming a secure proof-of-knowledge interactive proof with an honest verifier into a non-interactive digital signature scheme [42]. The goal of the interactive proof is to prove to a party (i.e., verifier) that another party (i.e., prover) has knowledge of something without revealing any underlying information. Typically, this occurs in a three-round, public-coin scheme (e.g. canonical) in which the verifier randomly chooses all of their messages during its execution [43]. In this scheme, the prover, holding a sk, sends a commitment ω to the verifier. Upon receiving ω, the verifier returns a random string as challenge c. The prover then generates a response s by applying their sk to a message composed of both ω and c . The prover then sends r to the verifier. The verifier then must run a verification algorithm, which outputs a decision bit by applying the prover's pk to r. The verifier then accepts if and only if the decision bit is equal to one. There are several identification schemes matching this three-round format [42], [44]–[46].

With minimum constraints, the FS transform can be applied to canonical three-round identification schemes in order to generate a signature scheme that is secure under the Random Oracle Model (ROM) [47], [48]. Figure 2.1 depicts a generalization of the standard FS transform and is based on a construction described in [47].

Algorithm 1 Sign of Σ_{FS}

Require: m, sk

Ensure: $sig_{FS} = (\omega, z)$, where H is a hash function, k is a security parameter, and $Coins_P$ is a set of binary strings.

1: $R_P \leftarrow_{\$} \text{Coins}_P(k)$
2: $\omega \leftarrow P(sk; R_P)$
3: $c \leftarrow H(\omega, m)$
4: $z \leftarrow P(sk, \omega, c; R_P)$
5: **return** (ω, z)

Algorithm 2 Verify of Σ_{FS}

Require: m, pk, (ω, z)

Ensure: $\{1, 0\}$▷ accept, reject signature

1: $c \leftarrow H(\omega, m)$
2: $b \leftarrow Vf(pk; \omega, c, z)$
3: **return** b

Figure 2.1. Fiat–Shamir Transform. Adapted from [47].

The FS transform is the basis for several digital signature schemes [24], [29], [49]–[51], both classical and post-quantum, because of its ability to create efficient signature schemes [47]. Due to this ubiquity, the FS transform plays a critical role in the generalized hybrid schemes introduced in Chapter 3.

2.2 History of Digital Signature Design

The RSA digital signature scheme was the first functional scheme to gain widespread popularity. Building from the work of Diffie and Hellman in 1976 [10], Rivest, Shamir and Aldeman proposed a primitive signature scheme using the RSA algorithm in 1978 [52]. In 1983, Massachusetts Institute of Technology (MIT) was granted a patent [53] with licensing exclusively controlled by what is now RSA Security, Inc. Over the course of the next decade, RSA became the de facto international standard, but it was not until 1991 that RSA Security released the Public-Key Cryptography Standards (PKCS) that established an industry standard interface for public key cryptography in the United States (U.S.)

During the 1980s, several digital signature schemes were designed as alternatives to RSA; however, most are rarely used in practice. In 1985, Tahir ElGamal described the ElGamal signature scheme [54]. Based on the difficulty of computing discrete logarithms, it is significantly more expensive both in terms of signing and signature size than RSA. Other signature schemes, such as Guillou-Quisquater (GQ) [55] and Feige-Fiat-Shamir [46], were designed to reduce the number of modular multiplications needed for generating RSA signatures by using the FS transform. The Feige-Fiat-Shamir scheme significantly improved efficiency at the cost of increased signature length while the GQ signature scheme only slightly improved efficiency [50]. Given the large industry market share of RSA, neither sufficiently enticed adoption.

Efficiency is not the only contributing factor to limited adoption. Legal ambiguity surrounding patents and government intervention can also limit use. In 1989, Claus Schnorr described [50] and patented [56] the Schnorr signature algorithm. This design improved on the efficiency of the Feige-Fiat-Shamir and GQ schemes and offered a viable alternative to RSA. In 1991, the United States Government (USG) through NIST desired to standardize a digital signature scheme without any encryption functionality that could be made available worldwide on a royalty-free basis. Since both the RSA and Schnorr schemes were patented

and enforced through licensing, NIST proposed the Digital Signature Standard (DSS) [24] along with their own signature scheme: Digital Signature Algorithm (DSA). DSA and its successor Elliptic Curve Digital Signature Algorithm (ECDSA) specifically adapted elements of the ElGamal signature scheme for patent avoidance reasons. This immediately caused legal strife as multiple groups claimed DSA infringed on their existing patents; however, nothing was ever successfully argued in a court of law. As a result, the Schnorr signature scheme did not see any extensive use until after its patent expired in 2008.

The segmentation caused by legal ambiguity and lack of standardization has resulted in a limited cryptographic landscape for digital signature schemes. Under FIPS 186-4 [24], the US government still only has three approved signature algorithms: RSA, DSA, and ECDSA; however, under a current draft standard [57], DSA will only be used to verify legacy signatures prior to the standards implementation date. Using elliptic curve cryptography, ECDSA was formally standardized in 2000 and is more efficient in terms of time complexity and signature size [29] despite deliberately using an inefficient design to avoid patents. Additionally, RSA requires significantly longer keys to provide the same level of security as ECDSA [58]. Nonetheless, RSA is still used in the majority of digital signature certificates over two decades later.

In 2017, NIST began the Post-Quantum Cryptography (PQC) Standardization project [8] to investigate quantum-safe cryptographic counterparts to existing standards. Unlike past projects like AES and SHA-3, NIST anticipates that the "evaluation process for these post-quantum cryptosystems may be significantly more complex" due to more demanding requirements, limited understanding of the power of quantum computers, and the diversity of "design attributes and mathematical foundations" in proposed candidates [9]. Over three years later, the project is in the call for comments period for the third round of candidates. Table 2.1 depicts the third round of PQ digital signature candidates.

Table 2.1. NIST Round 3 PQ Signature Candidates

Finalists	Alternates
CRYSTALS-DILITHIUM	GeMSS
FALCON	Picnic
Rainbow	SPHINCS

While provable security and efficiency are critical to the selection process, NIST has officially stated that the "intellectual property covering an algorithm or its implementations and the availability and terms of licenses to interested parties" has and will be used as selection criteria [41]. NIST's goal remains largely the same as in their efforts in 1991: to enable world-wide adoption by standardizing royalty-free algorithms. NIST has also carefully chosen candidates with different design attributes to help safeguard against future attacks or hidden flaws. If more than one digital signature scheme is finalized, the flexibility of choice will be a direct counter to the stagnation of standardized digital signature schemes; however, the transition from classical to PQ is still left to individual applications. Additionally, current patents [59]–[62] may hinder PQ adoption despite NIST's best efforts. As a result, hybridization, despite its increased costs and complexity, may prove essential in providing a mechanism to not only aid transitioning, but also legally bypass broad but enforceable patents.

Other important aspects for NIST's PQ selection criteria are the flexibility and simplicity of the scheme. Specifically, NIST is targeting "simple and elegant designs" which reflect "better understanding and confidence of the design team and encourages further analysis" [41]. The entire process relies on the transparency of the algorithms' design through documentation and source code. Additionally, the security community plays a vital role in examining the stated security of each algorithm and identifying potential flaws. Unfortunately, even with active cooperation, it is highly unlikely that every gap in security will be identified prior to official standardization. Many of the proposed candidates use "new and interesting designs" and contain "unique features that are not present in the current NIST standardized public-key algorithms" [41]. As evidenced by the history of attacks against the RSA cryptosystem [63], securely implementing a signature scheme is a non-trivial task even when the underlying mathematical principles are well-understood as simple mistakes can drastically impact security. Therefore, any lack of public understanding or trust of an algorithm's characteristics or implementation hinders widespread adoption.

While the confidence and trust of PQ signature algorithms will improve with time, there is no guarantee that the next generation of digital signature algorithms will achieve the same levels as their predecessors. Therefore, identifying different ways to securely combine signature algorithms plays a vital role in combating any trust issues in both the short and long term. If the security of the overall signature scheme can be equally balanced between

two different signature algorithms, then it is possible to place trust in the combined security properties of the component algorithms. This limits the risk of exposing the entire scheme to failure in the event of failure of only a single signature algorithm. Not every hybrid scheme provides this guarantee (the security properties of hybrid schemes are covered in detail in Chapter 3).

2.3 Digital Signature Schemes

In this section, digital signature algorithms, both classical and PQ, are divided into their families of hardness problems. For each algorithm, a brief description of the algorithm, its specific hardness problem, and security properties are discussed. This section is not a comprehensive list of all families of hardness problems; however, each family listed is used by a tested signature scheme in this work and familiarity with this information is assumed in Chapter 3. A brief summary of this section can be found in Table 2.2.

Table 2.2. Summary of Considered Digital Signature Schemes

Name	Family	Hardness Problem	Quantum-Resistance
RSA-FDH	Integer Factorization	RSAP	
DSA	Discrete Logarithm	DLP	
ECDSA	Discrete Logarithm	ECDLP	
CRYSTALS-DILITHIUM	Lattice	MLWE, MSIS	✓
qTESLA	Lattice	R-LWE	✓
FALCON	Lattice	NTRU	✓
Rainbow	Multivariate	UOV	✓
GeMSS	Multivariate	HFE	✓
MQDSS	Multivariate	MQ	✓

2.3.1 Integer Factorization

The hardness of integer factorization is informally defined as the difficulty in factoring large composite integers into a product of smaller integers. Given sufficiently large size, an adversary should not be able to factor the product of two random integers within polynomial-time with non-negligible probability. The difficulty can be further increased if the factors

are limited to prime numbers (i.e., prime factorization). Variations of this problem can be motivated by security considerations to include restrictions on the primes and cryptographic primitive requirements.

RSA

The RSA digital signature scheme is an RSA-based signature scheme that uses the hash-and-sign paradigm to circumvent forgeability and other security flaws in the original plain RSA digital signature scheme. PKCS #1 [64], published by RSA Laboratories and available via RFC 8017, specifies two encoding methods: EMSA-PSS and EMSA-PKCS1-v1_5. Both encoding methods are applied to the hash of the signed message and are used as input to the signing algorithm. NIST approves both methods for use on government systems [24]. For simplicity, Figure 2.2 depicts the full domain hash (FDH) variant of the RSA digital signature scheme, thus ignoring how the hash is encoded prior to signing.

The security of the RSA digital signature scheme relies on the RSA problem:

> Given a positive integer n which is the product of at least two primes, an integer e coprime with $\phi(n)$ and an integer c, find an integer m such that $m^e \equiv c \pmod{n}$. [25]

Security proofs showing EUF-CMA exist for RSA-FDH [65], RSASSA-PKCS1-v1_5 [66], and RSASSA-PSS [67], [68].

Algorithm 3 Sign of Σ_{RSA}

Require: m, sk

Ensure: $sig_{RSA} = s$, where H is a hash function.

1: $c \leftarrow H(m)$
2: $s = (c||pad)^{sk} \bmod n$
3: **return** s

Algorithm 4 Verify of Σ_{RSA}

Require: m, pk, s

Ensure: $\{1, 0\}$▷ accept, reject signature

1: $(c||pad) \leftarrow (s)^{pk} \bmod n$
2: **if** $c = H(m)$ **then**
3: **return** 1
4: **end if**
5: **return** 0

Figure 2.2. RSA Signature Algorithm. Adapted from [64].

2.3.2 Discrete Logarithm

The discrete logarithm problem is well known hardness problem within public key cryptography and is considered intractable over specific groups (e.g., $\mathbb{Z}_p^*$). Essentially, while modulo exponentiation is simple to solve, the inverse operation is difficult:

> Let p be prime (e.g., $p \in \mathbb{Z}_p^*$) and q be a primitive root of p (e.g., $q \in \mathbb{Z}_p^+$). Given p, q, find x such that $x \equiv \log_q y \pmod{p}$. [25]

For cryptography, group choices are typically subgroups of a multiplicative group over prime fields, a multiplicative group over finite fields of characteristic 2, or elliptic curve groups over finite fields [1].

DSA

DSA is based on modular exponentiation and the discrete logarithm problem. DSA is no longer recommended for continued use in the most recent draft revision to DSS [69]. Instead, alternatives such as ECDSA, which uses elliptic curves, and EdDSA, which uses twisted Edwards curves, have gained popularity. Both ECDSA and DSA have achieved provable security under specific parameters [28], [70]. Figure 2.3 depicts the DSA Sign and Verify operations, respectively.

Algorithm 5 Sign of Σ_{DSA}

Require: m, sk

Ensure: $sig_{DSA} = (r, s)$, where H is a hash function.

1: $k \leftarrow_{\$} \mathbb{Z}_q^*$
2: $r \leftarrow (g^k \bmod p) \bmod q$
3: $s \leftarrow k^{-1}(H(m) + (sk)r) \bmod q$
4: **return** (r, s)

Algorithm 6 Verify of Σ_{DSA}

Require: m, pk, (r, s)

Ensure: $\{1, 0\}$▷ accept, reject signature

1: $b \leftarrow s^{-1} \bmod q$
2: $u_1 \leftarrow (H(m) \cdot b) \bmod q$
3: $u_2 \leftarrow (rb) \bmod q$
4: $v \leftarrow ((g^{u_1} \cdot g^{u_2}) \bmod p) \bmod q$
5: **if** $v = r$ **then**
6: **return** 1
7: **end if**
8: **return** 0

Figure 2.3. DSA Signature Algorithm. Adapted from [24].

2.3.3 Lattice

Lattice-based cryptography uses a lattice, or the set of all linear combinations of linearly independent vectors in real n-space $\mathbb{R}^n$, to form the base for a series of hard problems [71]. These problems include the shortest vector problem (SVP), the closest vector problem (CVP), or a decisional variant of the two. Most lattice-based schemes do not directly rely on the hardness of these problems. Instead, they use variants of either the Learning With Errors (LWE) or Short Integer Solution (SIS) problems which are, in turn, reliant on, and sometimes reducible to the hardness of lattice problems [25].

Crystals-Dilithium

The Dilithium signature scheme [72] is a PQ lattice-based scheme that was designed to minimize the sum of the size of the public key and signature. The scheme's security is based on the hardness of finding short vectors in lattices. Specifically, Dilithium has been found to be SUF-CMA secure in the ROM based on the hardness of both the Module Learning With Errors (MLWE) and Module Short Integer Solution (MSIS) lattice problems. Of interest, Dilithium uses a unique cryptographic hash function, known as "hashing to a ball," that that hashes onto a set of elements of a chosen cyclotomic ring. The output of this function is a random 256-element array with exactly 60 ±1s and 196 0s and is used directly as the commitment within the scheme. Figure 2.4 offers a basic overview of both the Sign and Verify operations, respectively.

Algorithm 7 Sign of Σ_{Di}

Require: m, sk

Ensure: $sig_{Di} = (z, c)$

$A \in R_q^{k \times \ell} \coloneqq \text{ExpandA}(\rho)$
$\mu \in \{0,1\}^{384} \coloneqq \text{CRH}(tr \| m)$
$\kappa \coloneqq 0, (z, h) \coloneqq \perp$
$\rho' \in \{0,1\}^{384} \coloneqq \text{CRH}(K \| \mu)$
while $(z, h) = \perp$ **do**
    $y \in S^{\ell}_{\gamma_1 - 1}$
    $w \coloneqq Ay$
    $w_1 \coloneqq \text{HighBits}(w, 2_{\gamma_2})$
    $c \in B_{60} \coloneqq H(\mu \| w_1)$
    $z \coloneqq y + cs_1$
    $(r_1, r_0) \coloneqq \text{Decompose}_q(w - cs_2, 2\gamma_2)$
    if $(\|z\|_\infty \geq \gamma_1 - \beta) \vee (\|r_0\|_\infty \geq \gamma_2 - \beta) \vee (r_1 \neq w_1)$ **then**
        $(z, h) \coloneqq \perp$
    else
        $h \coloneqq \text{MakeHint}_q(-ct_0, w - cs_2 + ct_0, 2\gamma_2)$
        if $(\|ct_0\|_\infty \geq \gamma_2) \vee$ (# of 1's in h is $\geq \omega$) **then**
            $(z, h) \coloneqq \perp$
        end if
        $\kappa \coloneqq \kappa + 1$
    end if
end while
return (z, h, c)

Algorithm 8 Verify of Σ_{Di}

Require: $m, pk, (z, c)$

Ensure: $\{1, 0\}$▷ accept, reject signature

$A \in R_q^{k \times \ell} \coloneqq \text{ExpandA}(\rho)$
$\mu \in \{0,1\}^{384} \coloneqq \text{CRH}(\rho \| t_1 \| m)$
$w_1' \coloneqq \text{UseHint}_q(h, Az - ct_1 \cdot 2^d, 2\gamma_2)$
if $(\|z\|_\infty < \gamma_1 - \beta) \wedge (c = H(\mu \| w_1')) \wedge$ (# of 1's in h is $\leq \omega$) **then**
    return 1
end if
return 0

Figure 2.4. CRYSTALS-DILITHIUM Signature Algorithm. Source: [72]. The ExpandA function maps a uniform seed $p \in \{0,1\}^{256}$ to a matrix A. CRH is a collision resistant hash function (e.g., SHAKE-256). The Decompose_q, HighBits, and MakeHint functions are used to reduce the size of the public key by extracting higher and lower order bits of elements in $\mathbb{Z}_q$. UseHint uses the hint produced by MakeHint to recover the high-order bits of the sum.

qTESLA

The qTESLA signature scheme [51] is a PQ lattice-based scheme that relies on the decisional Ring LWE (R-LWE) problem. The scheme was designed for implementation simplicity and practical security at the cost of increased signature and public key size. qTESLA has been proven EUF-CMA secure in the Quantum ROM (QROM). Figure 2.5 offers a basic overview of both the Sign and Verify operations, respectively.

Algorithm 9 Sign of Σ_{qT}

Require: m, sk

Ensure: $sig_{qT} = (z, c)$, where H is a hash function.

1: counter $\leftarrow 1$
2: $r \leftarrow_{\$} \{0,1\}^{\kappa}$
3: $r \leftarrow \text{PRF}_2(\text{seed}_y, r, G(m))$
4: $y \leftarrow \text{ySampler}(\text{rand}, \text{counter})$
5: $a_1, ..., a_k \leftarrow \text{GenA}(\text{seed}_a)$
6: **for** $i = 1, ..., k$ **do**
7: $\quad v_i = a_i y \bmod \pm q$
8: **end for**
9: $c' \leftarrow H(v_1, ..., v_k, G(m), g)$
10: $c \coloneqq \{\text{pos\_list}, \text{sign\_list}\}$
11: $z \leftarrow y + sc$
12: **if** $z \notin \mathcal{R}_{[B-S]}$ **then**
13: $\quad$ counter $\leftarrow$ counter + 1
14: $\quad$ Restart at step 4
15: **end if**
16: **for** $i = 1, ..., k$ **do**
17: $\quad w_i \leftarrow v_i - e_i c \bmod \pm q$
18: $\quad$ **if** $(\|[w_i]_L\|_\infty \geq 2^{d-1} - E) \vee (\|w_i\|_\infty \geq \lfloor q/2 \rfloor - E)$ **then**
19: $\quad\quad$ counter $\leftarrow$ counter + 1
20: $\quad\quad$ Restart at step 4
21: $\quad$ **end if**
22: **end for**
23: **return** (z,c)

Algorithm 10 Verify of Σ_{qT}

Require: m, pk, (z, c)

Ensure: $\{1, 0\}$▷ accept, reject signature

1: $c \coloneqq \{\text{pos\_list}, \text{sign\_list}\} \leftarrow \text{Enc}(c')$
2: $a_1, ..., a_k \leftarrow \text{GenA}(\text{seed}_a)$
3: **for** $i = 1, ..., k$ **do**
4: $\quad w_i = a_i z - t_i c \bmod \pm q$
5: **end for**
6: **if** $(z \notin \mathcal{R}_{[B-S]}) \vee (c^{prime} \neq H(w_i, ..., w_k, G(m), G(t_1, ..., t_k))$ **then**
7: $\quad$ **return** 0
8: **end if**
9: **return** 1

Figure 2.5. qTESLA Signature Algorithm. Source: [51]. PRF is a pseudo-random function that takes a pre-seed and maps it to $(k + 3)$ seeds of k bits each; the ySampler function samples a polynomial $y \in \mathcal{R}_{[B]}$; the GenA function regenerates the polynomials $a_1, ..., a_k$ created during key generation for each key pair; the challenge c is encoded as two arrays, pos_list and sign_list, that contain the positions and signs of its nonzero coefficients; the encoding function Enc uses an Extendable-Output Function (XOF) to generate values uniformly at random that are interpreted as the positions and signs of the nonzero entries of c.

Falcon

Falcon [73] is a PQ signature scheme based on NTRU lattices [74]. Using the SIS over NTRU lattices hardness problem, Falcon has been proven EUF-CMA secure in both the ROM and QROM using the GPV framework [75]. The scheme's main design principles are to minimize both the public key and signature size while maintaining performance and memory efficiency. While previous implementations required floating-point hardware support, the latest specification allows portability for systems without support. Figure 2.6 offers a high-level description of Falcon's Sign and Verify operations.

Algorithm 11 Sign of Σ_{Fa}

Require: m, sk, a bound β

Ensure: $sig_{Fa} = (r, s)$, where HashToPoint is an extendable-output hash function, FFT is a fast Fourier transform, and ffSampling is a fast Fourier sampling algorithm.

$r \leftarrow \{0,1\}^{320}$ uniformly
$c \leftarrow \text{HashToPoint}(r \| m)$
$t \leftarrow (\text{FFT}(c), \text{FFT}(0)) \cdot \hat{B}^{-1}$
do
    $z \leftarrow \text{ffSampling}_n(t, T)$
    $s = (t - z)\hat{B}$
while $\|s\| > \beta$
$(s_1, s_2) \leftarrow \text{invFFT}(s)$
$s \leftarrow \text{Compress}(s_2)$
return (r, s)

Algorithm 12 Verify of Σ_{Fa}

Require: m, pk, (r, s), a bound β

Ensure: $\{1, 0\}$▷ accept, reject signature

$c \leftarrow \text{HashToPoint}(r \| m, q, n)$
$c - s_2 h \bmod q$
if $\|(s_1, s_2)\| \leq \beta$ **then**
    return 1
end if
return 0

Figure 2.6. Falcon Signature Algorithm. Source: [73]. HashToPoint is an extendable output function defined for any $q \leq 2^{16}$. The ffSampling function uses fast Fourier sampling to adaptively apply a randomized rounding on the coefficients of t using information stored in the FALCON tree T. The FFT function is a fast Fourier transform function with the invFFT function defined as its inverse. The Compress function efficiently compresses polynomials into a concatenated bit string.

2.3.4 Multivariate

Drawing from algebraic geometry, this family of problems is related to solving multivariate quadratic equations over finite fields [71]. Otherwise known as the MQ problem, the associated decision problem is known to be NP-complete [76] and given sufficient randomness, it is believed to be intractable [77], [78]. No known algorithm, quantum or otherwise, exists that can solve the MQ problem in polynomial-time. This makes it ideal for several applications in cryptography to include asymmetric public key cryptosystems [79], [80]. The MQ problem is based on solving a system of m quadratic equations with n variables in polynomial time and relies on $m \approx n$. If the ratio is not maintained, the MQ problem is solvable in polynomial time [49], [71].

Rainbow

The Rainbow signature scheme [33] is a multi-layer Oil-Vinegar system that modifies and extends the original unbalanced Oil and Vinegar scheme [81]. The Oil and Vinegar scheme uses the idea that a system of quadratic equations over a finite field can be easily solved if they have an certain structure with "oil" and "vinegar" variables that only mix in predetermined way [33], [82]. Rainbow is specifically designed for increased efficiency, simplicity, and shorter signature sizes. In order to accomplish this, Rainbow has significantly larger public and private keys. Using a transformation [83] to include a randomized 128-bit salt, Rainbow achieves EUF-CMA security in the ROM. Figures 2.7 offers a basic overview of both the Sign and Verify operations, respectively.

Algorithm 13 Sign of Σ_{Ra}

Require: m, sk

Ensure: sig_{Ra} = z,

1: $h \leftarrow H(m)$
2: $x \leftarrow InvS \cdot (h - c_S)$
3: $y \leftarrow \text{InvF}(\mathcal{F}, x)$
4: $z \leftarrow InvT \cdot (y - c_T)$
5: **return** z

Algorithm 14 Verify of Σ_{Ra}

Require: m, z

Ensure: $\{1, 0\}$▹ accept, reject signature

1: $h \leftarrow H(d)$
2: $h' \leftarrow \mathcal{P}(z)$
3: **if** $h' == h$ **then**
4: **return** 1
5: **end if**
6: **return** 0

Figure 2.7. Rainbow Signature Algorithm. Source: [33]. The InvS and InvT functions invert the affline maps $\mathcal{S}$ and $\mathcal{T}$. The InvF function inverts the quadratic central map $\mathcal{F}$.

GeMSS

GeMSS [84] is a multivariate-based, PQ signature scheme built around a variant of the Hidden Field Equations (HFE) cryptosystem. Also known as the Great Multivariate Short Signature, GeMSS uses minus and vinegar modifiers (e.g., HFEv-) and is considered a faster variant to another multivariate signature scheme: QUARTZ [85]. GeMSS is considered EUF-CMA secure under the ROM. The advantages of this scheme include its well-studied HFE lineage, fast verification times, and the shortest signatures of all of the NIST PQ candidates. Figure 2.8 describes both the GeMSS Sign and Verify operations.

Algorithm 15 Sign of Σ_{Ge}

Require: m, sk

Ensure: $sig_{Ge} = (S_{\text{nb\_ite}}, X_{\text{nb\_ite}}, ..., X_1)$ where nb_ite is the number of iterations.

1: $h \leftarrow H(m)$
2: $S_0 \leftarrow 0 \in \mathbb{F}_2^n$
3: **for** i from 1 to nb_ite **do**
4: $\quad D_i \leftarrow$ first n bits of h
5: $\quad (S_i, X_i) \leftarrow \text{GeMSS.Inv}_p(D_i \oplus S_{i-1})$
6: $\quad h \leftarrow H(h)$
7: **end for**
8: **return** $(S_{\text{nb\_ite}}, X_{\text{nb\_ite}}, ..., X_1)$

Algorithm 16 Verify of Σ_{Ge}

Require: m, $(S_{\text{nb\_ite}}, X_{\text{nb\_ite}}, ..., X_1)$

Ensure: $\{1, 0\}$▷ accept, reject signature

1: $h \leftarrow H(m)$
2: **for** i from 1 to nb_ite **do**
3: $\quad D_i \leftarrow$ first n bits of h
4: $\quad h \leftarrow H(h)$
5: **end for**
6: **for** i from nb_ite − 1...0 **do**
7: $\quad S_i \leftarrow p(S_{i+1}, X_{i+1}) \oplus D_{i+1}$
8: **end for**
9: **if** $S_0 = 0$ **then**
10: $\quad$ **return** 1
11: **end if**
12: **return** 0

Figure 2.8. GeMSS Signature Algorithm. Source: [84].

MQDSS

MQDSS [49] is a PQ signature scheme that relies directly on the MQ problem and has been proven EUF-CMA secure in the ROM. At its core, the MQDSS scheme is based on applying the FS transform to the five-pass SSH identification scheme [77]. MQDSS has smaller key sizes compared to other MQ schemes and is naturally parallelizable; however, this comes at the cost of significantly larger signature sizes. Figure 2.8 depicts a high-level view of MQDSS's Sign and Verify operations.

Algorithm 17 Sign of Σ_{MQ}

Require: m, sk

Ensure: $sig_{MQ} = (R, \sigma_0, \sigma_1, \sigma_2)$,

$S_F, S_s, S_\rho, S_{rte} \leftarrow \mathrm{PRG}_{sk}(sk)$

$F \leftarrow \mathrm{XOF}_F(S_F)$

$s \leftarrow \mathrm{PRG}_s(S_s)$

$pk \coloneqq (S_F, F(s))$

$R \leftarrow H(sk \| m)$

$D \leftarrow H(pk \| R \| m)$

$\rho_0^{(1)}, ..., \rho_0^{(r)}, \rho_1^{(1)}, ..., \rho_1^{(r)} \leftarrow \mathrm{PRG}_\rho(S_\rho, D)$

$r_0^{(1)}, ..., r_0^{(r)}, t_0^{(1)}, ..., t_0^{(r)}, e_0^{(1)}, ..., e_0^{(r)} \leftarrow \mathrm{PRG}_{rte}(S_{rte}, D)$

for $j \in \{1, ..., r\}$ **do**

    $r_1^{(j)} \leftarrow s - r_0^{(j)}$

    $c_0^{(j)} \leftarrow Com_0(\rho_0^{(j)}, r_0^{(j)}, \mathrm{G}(t_0^{(j)}, e_0^{(j)})$

    $c_1^{(j)} \leftarrow Com_1(\rho_1^{(j)}, r_1^{(j)}, \mathrm{G}(t_0^{(j)}, r_1^{(j)} + e_0^{(j)})$

    $\mathrm{com}^{(j)} \coloneqq (c_0^{(j)}, c_1^{(j)})$

end for

$\sigma_0 \leftarrow H(\mathrm{com}^{(1)} \| \mathrm{com}^{(2)} \| \mathrm{com}^{(3)})$

$\mathrm{ch}_1 \leftarrow H_1(D, \sigma_0)$

Parse ch_1 as $\mathrm{ch}_1 = (\alpha^{(1)}, \alpha^{(2)}, ..., \alpha^{(r)}), \alpha^{(j)} \in \mathbb{F}_q$

for $j \in \{1, ..., r\}$ **do**

    $t_1^{(j)} \leftarrow \alpha^{(j)} r_0^{(j)} - t_0^{(j)}, e_1^{(j)} \leftarrow \alpha^{(j)} F(r_0^{(j)}) - e_0^{(j)}$

    $\mathrm{resp}_1^{(j)} \coloneqq (t_1^{(j)}, e_1^{(j)})$

end for

$\sigma_1 \leftarrow (\mathrm{resp}_1^{(1)} \| \mathrm{resp}_1^{(2)} \| ... \| \mathrm{resp}_1^{(r)})$

$\mathrm{ch}_2 \leftarrow H_2(D, \sigma_0, \mathrm{ch}_1, \sigma_1)$

Parse ch_2 as $\mathrm{ch}_2 = (b^{(1)}, b^{(2)}, ..., b^{(r)}), b^{(j)} \in \{0, 1\}$

for $j \in \{1, ..., r\}$ **do**

    $\mathrm{resp}_2^{(j)} \leftarrow r_{b^{(j)}}^{(j)}$

end for

$\sigma_2 \leftarrow (\mathrm{resp}_2^{(1)} \| \mathrm{resp}_2^{(2)} \| ... \| \mathrm{resp}_2^{(r)} \| c_{1-b^{(1)}}^{(1)} \| c_{1-b^{(2)}}^{(2)}) \| ... \| c_{1-b^{(r)}}^{(r)} \| \rho_{b^{(1)}}^{(1)} \| ... \| \rho_{b^{(r)}}^{(r)}$

return $(R, \sigma_0, \sigma_1, \sigma_2)$

Algorithm 18 Verify of Σ_{MQ}

Require: (m, pk, σ)
Ensure: $\{1, 0\}$▷ accept, reject signature

$F \leftarrow \mathrm{XOF}_F(S_F)$
$D \leftarrow H(pk \| R \| m)$
$\mathrm{ch}_1 \leftarrow H_1(D, \sigma_0)$
Parse ch_1 as $\mathrm{ch}_1 = (\alpha^{(1)}, \alpha^{(2)}, ..., \alpha^{(r)}), \alpha^{(j)} \in \mathbb{F}_q$
$\mathrm{ch}_2 \leftarrow H_2(D, \sigma_0, \mathrm{ch}_1, \sigma_1)$
Parse ch_2 as $\mathrm{ch}_2 = (b^{(1)}, b^{(2)}, ..., b^{(r)}), b^{(j)} \in \{0, 1\}$
Parse σ_1 as $\sigma_1 = (\mathrm{resp}_1^{(1)} \| \mathrm{resp}_1^{(2)} \| ... \| \mathrm{resp}_1^{(r)})$
Parse σ_2 as $\sigma_2 = (\mathrm{resp}_2^{(1)} \| \mathrm{resp}_2^{(2)} \| ... \| \mathrm{resp}_2^{(r)} \| c^{(1)}_{1-b^{(1)}} \| c^{(2)}_{1-b^{(2)}}) \| ... \| c^{(r)}_{1-b^{(r)}} \| \rho^{(1)}_{b^{(1)}} \| ... \| \rho^{(r)}_{b^{(r)}}$
for $j \in \{1, ..., r\}$ **do**
    Parse $\mathrm{resp}_1^{(j)}$ as $\mathrm{resp}_1^{(j)} = (t_1^{(j)}, e_1^{(j)})$
    if $b^{(j)} == 0$ **then**
        $r_0^{(j)} = \mathrm{resp}_2^{(j)}$
        $c_0^{(j)} \leftarrow Com_0(\rho_0^{(j)}, r_0^{(j)}, \alpha^{(j)} r_0^{(j)} - t_1^{(j)}, \alpha^{(j)} F(r_0^{(j)}) - e_1^{(j)})$
    else
        $r_1^{(j)} = \mathrm{resp}_2^{(j)}$
        $c_1^{(j)} \leftarrow Com_1(\rho_1^{(j)}, r_1^{(j)}, \alpha^{(j)}(v - F(r_1^{(j)})) - \mathrm{G}(t_1^{(j)}, r_1^{(j)} + e_1^{(j)})$
    end if
    $\mathrm{com}^{(\mathrm{j})} := (\mathrm{c}_0^{(\mathrm{j})}, \mathrm{c}_1^{(\mathrm{j})})$
end for
$\sigma_0' \leftarrow H(\mathrm{com}^{(1)} \| \mathrm{com}^{(2)} \| ... \| \mathrm{com}^{(r)})$
if $\sigma_0' == \sigma_0$ **then**
    return 1
end if
return 0

Figure 2.8. MQDSS Signature Algorithm. Source: [49].

CHAPTER 3:
Hybrid Signature Schemes

From a broad perspective, hybrid signature schemes construct a single signature scheme from two or more component digital signature algorithms that are applied to a common message. The resulting signature can be constructed using any number of techniques as long as the security notions provided by the component algorithms are still valid during both the signing and verification phases. This allows a hybrid signature scheme to achieve at least the same level of security as the verified component algorithms. There are multiple techniques that can be applied to create a hybrid scheme with each achieving differing levels of security. For example, simply concatenating the output of two component signatures meets the above hybrid scheme definition. This approach requires the successful verification of both signatures to achieve the benefits of using a hybrid scheme, but maintains a level of simplicity and backwards compatibility. As long as the verifier recognizes at least one of the component algorithms, signature verification still occurs albeit without the security benefits of the second signature algorithm.

The ability to guarantee that the security of a hybrid scheme is at least as strong as that of its component algorithms offers a solution to maintaining uninterrupted cryptographic security during the PQ transition. With the current disparity in understanding between classical and PQ signature schemes, system engineers, developers, etc. may be hesitant to convert legacy cryptosystems to use PQ signature schemes. These newer schemes represent an increase in complexity which, in turn, increases the likelihood of software vulnerabilities during implementation. Additionally, new attacks against a PQ algorithm may appear that jeopardizes its security. By combining a newer signature algorithm with one supported by the existing system, developers can "hedge their bets" without sacrificing overall security.

Developers that desire to offload risk associated with adopting relatively new PQ signature algorithms need to examine and understand not only the security notions and parameters of the component signatures, but also the effects hybridization may impose. Hybrid signature schemes can directly impact the security proofs and guarantees of the component algorithms, thus requiring a new proof before they can be implemented securely. Additionally, it is difficult to equate the overall security level between algorithms based on different hardness

problems. This can be further complicated by the differences between classical and post-quantum models and requires caution by anyone implementing a new signature scheme.

With this in mind, the value of hybrid signature schemes also extends beyond the PQ transition. By their nature, hybrid schemes provide developers with flexibility in deciding which component algorithms to combine. This can be valuable in situations where a developer is required to use a standardized digital signature algorithm, but does not trust the algorithm itself or its implementation. For example, specific requirements for cryptographic modules, detailed in FIPS 140 [86], must be met before they can be used by any entity within the USG under federal law. These cryptographic modules include any hardware or software implementations that use digital signatures. FIPS 140 specifically states all cryptographic modules "shall employ approved cryptographic algorithms." [86]. If a developer does not trust the approved signature algorithms for their application, they can use a hybrid scheme where one component signature is approved and the other is of their choosing. NIST only requires that one of the component signatures be approved under FIPS 140 in a hybrid scheme and does not offer any specific guidance for hybrid signature modes [9]. This effectively leaves the decision to developers to implement efficient and secure hybrid modes that are compatible with other cryptosystems.

This chapter seeks to aid early adopters by informally defining the security notions unique to hybrid signature schemes and providing a high-level overview of various ways component signature algorithms can be combined to achieve hybridization. For the sake of clarity and efficiency, all considered hybrid signature schemes are composed of only two component signature algorithms; however, this is an arbitrary limit.

3.1 Hybrid Security Notions

Hybrid signature schemes introduce two unique security properties: *non-separability* and *hybrid verification*. Both examine the relationship between the output produced by the component algorithms and the final hybrid signature. This section provides informal definitions for both non-separability and hybrid verification and discusses their importance in regard to hybrid signature schemes.

Non-Separability [15]. It is difficult for an attacker to take a hybrid signature and turn it into a signature that the verifier accepts as a valid single, component algorithm signature.

As defined, non-separability examines whether or not it is possible to isolate the components of a hybrid signature in a way that it verifies correctly for one of the underlying signature schemes without indication of the other scheme's existence. At its core, non-separability implies that it is impossible to verify a component algorithm's signature without also acknowledging that the other component algorithm exists.

Within a hybrid scheme, non-separability can apply to either component algorithm, both component algorithms, or none of the component algorithms. Therefore, it is possible for a hybrid scheme to achieve different types of separation. For example, simply signing a common message twice is a valid hybrid scheme; however, it is completely separable if nothing in the message (e.g, a custom header or field) indicates that it is part of a hybrid scheme. Using the same example, if the message being signed by both component algorithms contains a field that indicates the hybrid scheme, both component algorithms achieve non-separability *assuming* that the "indicator" field is properly parsed and acknowledged. Finally, using the same example, if the first component algorithm signs the original message and the second signs the original message and the field, only the second signature would be non-separable.

As shown by the last example, the placement of the field (e.g., hybrid indicator) also affects which component algorithms achieve non-separability. In general, there are two types of non-separability: complete and partial. If the indicator field is part of the message that both component signatures sign, the signature scheme achieves complete non-separability. If the indicator is only signed by one of the two signatures, then the scheme only achieves partial non-separability.

Non-separability is an important security property for hybrid signature schemes because it prevents an attacker from simply downgrading from a hybrid scheme to any one of the scheme's component algorithms without the knowledge of the receiving party. For example, if a message without any indicator of a hybrid scheme is signed twice, an adversary could split the message into two parts and only forward one to the recipient. In this scenario, the recipient has no way of knowing that the original hybrid scheme existed even after verifying the received signature. Using the same scenario, if the signed message contains a hybrid indicator field, the recipient would know that the single signature received from the attacker is incomplete. Partial non-separability is subject to the same style of downgrade

attack; however, the adversary is limited to using the separable signature and corresponding component algorithm.

The content of the hybrid indicator field must also specifically list the component algorithms that are being used in the scheme and be included with the original message as input to both signature algorithms (e.g., complete non-separability). If both of these criteria are not met, the hybrid scheme is vulnerable to more nuanced attacks. For example, imagine a hybrid scheme that achieves complete non-separability by including a hybrid indicator with the message that both component signature algorithms sign. If the indicator *only* describes that a hybrid scheme is used (e.g., a Boolean value) and does not specify the component algorithms, an adversary can replace either or both of the component algorithms in a hybrid scheme with one of their choosing that may be registered for the signer but compromised. Upon receiving the signed message, the recipient would be unaware of the tampering, because both signatures would still verify successfully, thus violating the original intent of the hybrid scheme.

Non-separability alone is insufficient to prevent every type of attack against a hybrid signature scheme. A completely non-separable hybrid scheme that includes an indicator that specifically lists the component algorithms is *still* vulnerable to a downgrade attack if the verifier accepts anything other than the specific two signature algorithms listed in the field. This would be possible in situations where two parties negotiate which signature algorithms they will accept prior to sending the signed message. If an adversary is able to change the accepted signature algorithm to either of the component algorithms, the receiver may choose to ignore the hybrid indicator even if it is part of the signed message. These assumptions are beyond those typically captured by digital signature security experiments as they are contingent on verifier behavior. This leads to the following property of hybrid verification.

Hybrid Verification [23]. Given a signature from a hybrid scheme, it should be infeasible to verify one or more component signatures without verifying *all* component signatures.

The second hybrid security property is centered around verifiability. Depending on the design of a hybrid scheme, it may or may not be possible to verify the signature of only one of the underlying algorithms. Hybrid verification establishes that verifying one component signature requires the verification of all signatures. This prevents a verifier from selectively choosing a single component signature for verification and guarantees to the signer that all

the individual security notions of a hybrid scheme are achieved upon successful validation. Hybrid verification also implies that the signature scheme is non-separable; not only does verification of a received signature ensure that the receiver has indication that a hybrid algorithm was used, but also that the receiver has verified all components.

Not all hybrid signature schemes achieve hybrid verification. In general, this security property necessitates the intermingling of the internal elements of two or more component algorithms in a way that meets the goal of hybrid verification without simultaneously negating the security of the component algorithms. Any changes to the function of a component algorithm requires reassessing the original security proof to ensure that unintentional vulnerabilities are not introduced. As such, hybrid verification straddles the line between being a security notion, in and of itself, and being its own unique scheme. It is also important to note that both non-separability and hybrid verification require an honest verifier. While hybrid verification does guarantee that verifying the hybrid signature requires successful completion of all component algorithms, it does not prevent a dishonest verifier from always returning success regardless of input.

3.2 Hybridization Techniques

There are several methods of combining component signature algorithms into a hybrid scheme. This section introduces two of the most widely observed methods (e.g., concatenation and nesting) and examines the security properties each achieve.

3.2.1 Concatenation

Concatenation is the simplest form of instituting a hybrid signature scheme. It requires the signer to independently sign a message using two component algorithms and then combine the results into one signature. Assuming the verifier has access to the corresponding public keys, they must also be able to correctly separate the concatenated signature before verifying them independently. Figure 3.1 depicts a naive hybrid concatenation scheme.

Algorithm 19 Sign_h of Σ_h = Concatenation

Require: sk_A, sk_B, m

1: $\sigma_A \leftarrow \text{Sign}_A(sk_A, m)$
2: $\sigma_B \leftarrow \text{Sign}_B(sk_B, m)$
3: **return** (σ_A, σ_B)

Algorithm 20 Verify of Σ_{Di}

Require: $pk_A, pk_B, m, (\sigma_A, \sigma_B)$

Ensure: $\{1, 0\}$▷ accept, reject signature

1: $\sigma_A, \sigma_B \leftarrow (\sigma_A, \sigma_B)$
2: **if** $\text{Verify}_A(pk_A, m, \sigma_A) \wedge \text{Verify}_B(pk_B, m, \sigma_B)$ **then**
3: **return** 1
4: **end if**
5: **return** 0

Figure 3.1. Naive Hybrid Concatenation Scheme

The advantage of concatenation is its ease of implementation. If a cryptographic library supports the component algorithms, then it merely is an issue of correctly separating the hybrid signature. Furthermore, since the component signature algorithms are left untouched, this scheme also receives the security properties achieved by its components *assuming* the component algorithms are correctly implemented and that both signatures are verified. So, if either σ_A or σ_B are unforgeable (e.g., EUF-CMA secure), the resulting hybrid signature is also unforgeable.

One of the downsides to concatenation is that it does not achieve non-separability unless the message is modified to include a hybrid indicator. This leaves the hybrid scheme vulnerable to a downgrade attack because an attacker can produce a valid signature for either scheme, σ_A or σ_B, by simply extracting one and excluding the other from the end product. This allows an adversary to attack the entire hybrid scheme since it can successfully produce a valid signature for one of the component algorithms without the recipient being aware of the original hybrid signature. The attack can further be amplified if the chosen algorithm is susceptible to a separate vulnerability that affects unforgeability, data integrity, or non-repudiation. Even by adding a hybrid indicator to the message, the concatenation scheme

still does not achieve hybrid verification. In the event of a naïve-but-honest verifier that only validates a single algorithm, the signer has no guarantee which signature will be verified. Therefore, the hybrid scheme, as a whole, does not achieve the security guarantees of both component algorithms.

3.2.2 Nesting

Nesting is another method of combining two component signature algorithms into a hybrid scheme. In its simplest form, this method requires that the second algorithm signs the output of the first algorithm (termed *weak nesting*). In weak nesting, the outer algorithm thus does not sign the message itself but only the inner signature (e.g., σ_A). While it is not possible to verify the outer signature without first verifying the inner signature, this technique does not bind the outer signature to the original message. As such, weak nesting critically depends on the unforgeability of only the inner signature algorithm (e.g., Σ_A). and not the outer signature algorithm (e.g., Σ_B).

The imbalance in the weakly nested scheme can be corrected if the outer algorithm Σ_B signs both the message and σ_A (e.g. strong nesting). This technique produces a scheme where if either Sign_A or Sign_B is unforgeable, then the hybrid scheme is also unforgeable [15]. Figures 3.2 and 3.3 depicts a weakly nested hybrid scheme and strongly nested hybrid scheme, respectively.

Algorithm 21 Sign_h of Σ_h = Weak Nesting

Require: sk_A, sk_B, m

1: $\sigma_A \leftarrow \text{Sign}_A(sk_A, m)$
2: $\sigma_B \leftarrow \text{Sign}_B(sk_B, \sigma_A)$
3: **return** (σ_A, σ_B)

Algorithm 22 Verify_h of Σ_h = Weak Nesting

Require: $pk_A, pk_B, m, (\sigma_A, \sigma_B)$

Ensure: $\{1, 0\}$▷ accept, reject signature

1: $b_2 \leftarrow \text{Verify}_B(pk_B, m, \sigma_A)$
2: $b_1 \leftarrow \text{Verify}_A(pk_A, \sigma_A, \sigma_B)$
3: **if** $b_1 \wedge b_2 = 1$ **then**
4: **return** 1
5: **end if**
6: **return** 0

Figure 3.2. Weakly Nested Hybrid Scheme. Adapted from [15].

Algorithm 23 Sign_h of Σ_h = Strong Nesting

Require: sk_A, sk_B, m

1: $\sigma_A \leftarrow \text{Sign}_A(sk_A, m)$
2: $\sigma_B \leftarrow \text{Sign}_B(sk_B, (m, \sigma_A))$
3: **return** (σ_A, σ_B)

Algorithm 24 Verify_h of Σ_h = Strong Nesting

Require: $pk_A, pk_B, m, (\sigma_A, \sigma_B)$

Ensure: $\{1, 0\}$▷ accept, reject signature

1: $b_1 \leftarrow \sigma_A \leftarrow \text{Sign}_A(sk_A, m)$
2: $b_2 \leftarrow \sigma_B \leftarrow \text{Sign}_B(sk_B, (m, \sigma_A))$
3: **if** $b_1 \wedge b_2 = 1$ **then**
4: **return** 1
5: **end if**
6: **return** 0

Figure 3.3. Strongly Nested Hybrid Scheme. Adapted from [15].

Nesting only partially achieves non-separability. In both weak and strong nesting, the inner signature σ_A is not connected to the outer signature σ_B. In the same vein as the attack against the concatenated hybrid scheme, an adversary is able to submit only σ_A and the signed message to the verifier. Unless the message includes a hybrid indicator, the verifier may successfully validate only Sign_A without knowledge of Sign_B. Conversely, under both schemes, if the verifier validates σ_B, they are implicitly aware that σ_A exists, thus achieving partial non-separability.

Neither form of nesting meets the definition of hybrid verification. It will always be possible for a verifier to only validate the inner signature without verifying the outer signature. While this limits the assumptions a signer can make in an honest setting, nesting allows for backward compatibility with legacy systems. If the inner signature scheme is supported, but the outer signature scheme is not, the system can still achieve the security provided by the inner signature algorithm. This can be extremely useful when transitioning to newer signature algorithms without requiring broad "day one" support.

3.3 True Hybrid Schemes

True hybrid schemes are an alternative to traditional hybridization techniques. Instead of combining two signature algorithms through concatenation or nesting, these schemes intertwine the component algorithms into a single scheme. True hybrid schemes can either explicitly state the component algorithms or use a generalized design where one or more of the component algorithms capture a category of schemes. In the latter design, the selected component algorithms must meet specific requirements dependent on the true hybrid scheme. Figure 3.4 shows the relationship between hybrid schemes for the purposes of this work.

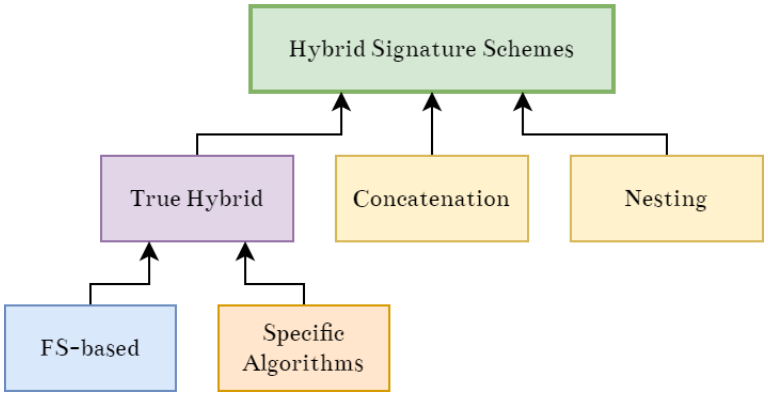


Figure 3.4. Terminology Relationship between Hybrid Signature Schemes

A "sub–class" of true hybrid schemes are FS–based generalized hybrid schemes. These schemes use a modular design for combining the component algorithms into a single scheme where at least one component algorithm uses the FS transform. In Section 2.1.2, we identified that the FS transform is commonly used in the creation of digital signature schemes. Several of the PQ candidates are internally designed on this method, thereby providing a convenient commonality for generalization. To be compatible with the FS–based true hybrid designs, component algorithms are required to use the FS transform to create a challenge c derived from the output of a hash function applied to a commitment ω. Table 3.1 lists the PQ signature algorithms considered in this work that meet this requirement.

There are many benefits to adopting a generalized true hybrid design. For example, generalization reduces the lag between the discovery of a new attack and the revision to an existing standard. One of the limiting factors opposing widespread adoption of PQ signature

Table 3.1. PQ Signature Algorithms with FS Compatibility

Signature Algorithm	Type
CRYSTALS-DILITHIUM	Lattice
qTESLA	Lattice
MQDSS	Multivariate

schemes is their increased complexity. Not all PQ signature schemes are as well-studied as their classical counterparts. If an efficient attack is found after standardization, NIST would need to rapidly transition to another candidate or, if possible, publish an updated standard for the broken signature scheme. While generalization does not solve this issue, it does protect current technology by creating an additional layer of defense. If only the generalized true hybrid scheme is standardized instead of explicitly specifying different algorithm combinations, implementations will be able to modularly swap a component algorithm without waiting for a revision to the original standard. As long as one component signature algorithm holds, then the security that it achieves is guaranteed. The generalized true hybrid model also works well with current NIST policy. In response to questions from the cryptographic community, NIST has publicly stated that they will accommodate the use of "dual signatures in FIPS 140 validation when suitably combined with a NIST-approved scheme" [9].

Generalized and non-generalized true hybrid schemes do come with obvious disadvantages. Unlike nesting and concatenation, the original component signature schemes must be modified in order to expose the necessary internal components. As a result, these schemes blur the line between being a composition of two existing signature schemes and being an entirely new one. Thus, depending on the modification, the composability of the entire scheme may need to be examined before the overall security can be assumed. In this work, comparison

of feasibility and efficiency of hybrid signature schemes primarily focuses on FS-based true hybrid schemes as they satisfy both complete non-separability *and* hybrid verification. Additionally, by not specifying specific component algorithms, a wide selection of signature algorithms could be cross-compared within one hybrid scheme. This allows for the exploration of any hidden idiosyncrasies between different combinations while simultaneously providing a robust baseline derived from both classic and PQ signature algorithms.

The following FS–based (i.e. using a drop-in FS algorithm selection) generalized true hybrid designs are presented for context in understanding implementation decisions in Chapter 4 and interpreting results in Chapters 6 and 7. Each of these designs are based on unpublished communications [23] and their security assessment is considered out of scope in this work.

Figures 3.5–3.10 depict FS–based true hybrid signature schemes that require at least one of the component algorithms to use the FS transform in a compatible way. Each of these generalized hybrid schemes specify the non-challenge portion of the FS signature as z and use the Rec function to reconcile ω during the verification operation. The message digest is annotated as $D(m)$. Three separate FS-DSA schemes are included in Figures 3.8–3.10. Each of these schemes use a distinct approach and are compatible with both the DSA and ECDSA signature algorithms. Figure 3.7 depicts a true hybrid scheme that is specific to the Falcon and RSA signature schemes specified in Section 2.3. As such, it is an example of a non-generalized true hybrid signature scheme.

3.4 Summary

Hybrid digital signature schemes provide a novel solution to many problems to include facilitating the transition from one signature algorithm to another, offloading the risk of adopting newer signature algorithms, and providing flexibility in the design and development of cryptosystems that use digital signatures. This chapter covered an overview of hybrid digital signature scheme goals, including two security properties unique to hybrid signature schemes – non-separability and hybrid verification in Section 3.1 – and the current hybridization techniques of concatenation and nesting (Section 3.2). Finally, Section 3.3 covered five true hybrid schemes which are the primary methods of hybridization used for feasibility and efficiency comparison in this work.

Algorithm 25 Sign_h of Σ_h = FS-FS

Require: m, sk
Ensure: Sign_h-H = (c, z_1, z_2), where $sig_1 = (c, z_1), sig_2 = (c, z_2)$, and H is a hash function.

1: $rand_1 \leftarrow_{\$}$ Rand
2: $rand_2 \leftarrow_{\$}$ Rand
3: $\omega_1 \leftarrow g_1(sk_1, rand_1)$
4: $\omega_2 \leftarrow g_2(sk_2, rand_2)$
5: $c \leftarrow H((\omega_1, \omega_2), D(m))$
6: $z_1 \leftarrow f_1(c, rand_1, sk_1)$
7: $z_2 \leftarrow f_2(c, rand_2, sk_2)$
8: **return** (c, z_1, z_2)

Algorithm 26 Verify of Σ_h = FS-FS

Require: $m, (c, z_1, z_2), pk$
Ensure: $\{1, 0\}$▷ accept, reject signature

1: $\omega_1 \leftarrow Rec(c, z_1, pk_1)$
2: $\omega_2 \leftarrow Rec(c, z_2, pk_2)$
3: $b_1 \leftarrow \text{verify}_1(pk_1; c, z_1, D(m))$
4: $b_2 \leftarrow \text{verify}_2(pk_2; c, z_2, D(m))$
5: **if** $b_1 \wedge b_2 \wedge (c = H(\omega_1, \omega_2, D(m)))$ **then**
6: **return** 1
7: **end if**
8: **return** 0

Figure 3.5. FS-FS Sign and Verify Algorithms. Source: [23].

Algorithm 27 Sign_h of Σ_h = FS-RSA

Require: m, sk
Ensure: Sign_h-H = (z, s), where $sig_1 = z, sig_2 = s$, and H is a hash function.

1: $rand \leftarrow_{\$}$ Rand
2: $\omega \leftarrow g(sk_1, rand)$
3: $c \leftarrow H(\omega, D(m))$
4: $z \leftarrow f(c, rand_1, sk_1)$
5: $s = (c \| pad)^{sk_2} \bmod n$
6: **return** (z, s)

Algorithm 28 Verify of Σ_h = FS-RSA

Require: $m, (z, s), pk$
Ensure: $\{1, 0\}$▷ accept, reject signature

1: $(c \| pad) \leftarrow (s)^{pk_2} \bmod n$
2: $\omega \leftarrow Rec(c, z, pk_1)$
3: Check $\text{verify}_1(pk_1; c, z, D(m))$
4: **if** $c = H(\omega, D(m))$ **then**
5: **return** 1
6: **end if**
7: **return** 0

Figure 3.6. FS-RSA Sign and Verify Algorithms. Source: [23].

Algorithm 29 Sign_h of Σ_h = Falcon-RSA

Require: m, sk
Ensure: Sign_h-H $= (z_2, z_3, s)$, where $sig_1 = (z_2, z_3), sig_2 = s$, and H is a hash function.

1: $r \leftarrow_{\$} \text{Rand}$
2: $c \leftarrow H(r \| m)$
3: $(z_1, z_2) \leftarrow f_1(c, sk)$ such that $z_1 + z_2 h = c \bmod q$
4: $s = (c \| pad)^{sk_2} \bmod n$
5: $z_3 \leftarrow z_1 \oplus r$
6: **return** (z_2, z_3, s)

Algorithm 30 Verify of Σ_h = Falcon-RSA

Require: $m, (z_2, z_3, s), pk$
Ensure: $\{1, 0\}$▷ accept, reject signature

1: $(c \| pad) \leftarrow (s)^{pk_2} \bmod n$
2: $z_1 \leftarrow c - z_2 h \bmod q$
3: $r \leftarrow z_1 \oplus z_3$
4: **if** $\|(z_1, z_2)\| \leq \beta \wedge c = H(r \| m)$ **then**
5: **return** 1
6: **end if**
7: **return** 0

Figure 3.7. Falcon-RSA Sign and Verify Algorithms. Source: [23].

Algorithm 31 Sign_h of Σ_h = FS-DSA

Require: m, sk
Ensure: Sign_h-H $= (z, c, r, s)$, where $sig_1 = (z, c), sig_2 = (r, s)$, and H is a hash function.

1: $rand \leftarrow_{\$} \text{Rand}$
2: $k \leftarrow_{\$} \mathbb{Z}_q^*$
3: $r \leftarrow F(g^k)$
4: $\omega \leftarrow g(sk_1, rand)$
5: $s \leftarrow k^{-1}(H(\omega, D(m)) + (sk_2)r) \bmod q$
6: $c \leftarrow H(\omega, h(r, s) \oplus h(m))$
7: $z_1 \leftarrow f_1(c, rand, sk_1)$
8: **return** (z, c, r, s)

Algorithm 32 Verify of Σ_h = FS-DSA

Require: $m, (z, c, r, s), pk$
Ensure: $\{1, 0\}$▷ accept, reject signature

1: $\omega \leftarrow Rec(c, z, pk_1)$
2: $b \leftarrow \text{verify}_1(pk_1; c, z, h(r, s) \oplus h(m))$
3: **if** $(b = 1) \wedge (r = g^{H(\omega, D(m))s^{-1}}(pk_2)^{rs^{-1}})$ **then**
4: **return** 1
5: **end if**
6: **return** 0

Figure 3.8. FS-DSA #1 Sign and Verify Algorithms. Source: [23].

Algorithm 33 Sign_h of Σ_h = FS-DSA

Require: m, sk
Ensure: Sign_h-H = (c, z, r, s), where $sig_1 = (c, z), sig_2 = (r, s)$, and H is a hash function.

1: $rand \leftarrow_{\$} \text{Rand}$
2: $k \leftarrow_{\$} \mathbb{Z}_q^*$
3: $\omega \leftarrow g(sk_1, rand)$
4: $r \leftarrow F(g^k)$
5: $c \leftarrow H((\omega, r), D(m))$
6: $c^* \leftarrow H(\omega, D(m))$
7: $z \leftarrow f(c, rand, sk_1)$
8: $s \leftarrow k^{-1}(c \oplus c^* + (sk_2)r) \bmod q$
9: **return** (c, z, r, s)

Algorithm 34 Verify of Σ_h = FS-DSA

Require: $m, (c, z, r, s), pk$
Ensure: $\{1, 0\}$▷ accept, reject signature

1: $\omega \leftarrow Rec(c, z, pk_1)$
2: $c^* \leftarrow H(\omega, D(m))$
3: $b \leftarrow \text{verify}(pk_1; c, z, D(m))$
4: **if** $(b = 1) \wedge (r = g^{(H((\omega,r),D(m))\oplus c^*)\cdot s^{-1}} \cdot pk_2^{r\cdot s^{-1}})$ **then**
5: **return** 1
6: **end if**
7: **return** 0

Figure 3.9. FS-DSA #2 Sign and Verify Algorithms. Source: [23].

Algorithm 35 Sign_h of Σ_h = FS-DSA

Require: m, sk
Ensure: Sign_h-H = (c, c', z, r, s), where $sig_1 = (c, z_1), sig_2 = (r, s)$, and H is a hash function.

1: $rand \leftarrow_{\$} \text{Rand}$
2: $a \leftarrow_{\$} \mathbb{Z}_q^*$
3: $k \leftarrow_{\$} \mathbb{Z}_q^*$
4: $\omega \leftarrow g(sk_1, rand)$
5: $r \leftarrow F(g^k)$
6: $c \leftarrow H(\omega, D(m))$
7: $z \leftarrow f(c, rand, sk_1)$
8: $s \leftarrow k^{-1}(c + (sk_2)r - a) \bmod q$
9: $c' \leftarrow H(\omega, g^{s^{-1}\cdot a}, s, D(m))$
10: **return** (c, c', z, r, s)

Algorithm 36 Verify of Σ_h = FS-DSA

Require: $m, (c, c', z, r, s), pk$
Ensure: $\{1, 0\}$▷ accept, reject signature

1: $\omega \leftarrow Rec(c, z, pk_1)$
2: $b \leftarrow \text{verify}(pk_1; c, z, D(m))$
3: $a' \leftarrow r^{-1} \cdot g^{(H(\omega, D(m)))\cdot s^{-1}} \cdot pk_2^{r\cdot s^{-1}} \bmod q$
4: **if** $(b = 1) \wedge (c' = H(\omega, a', s, D(m)))$ **then**
5: **return** 1
6: **end if**
7: **return** 0

Figure 3.10. FS-DSA #3 Sign and Verify Algorithms. Source: [23].

CHAPTER 4: Methodology

This chapter details the methodology used to implement the true hybrid signature schemes outlined in Section 3.3, using a selection of the component algorithms detailed in Section 2.3. The goal is to create a repeatable process that can be applied to other signature algorithms that are not covered in this work. As a result, the rationale behind design choices and the process of determining which signature algorithms are compatible with the generalized hybrid schemes are given in detail.

In addition to describing the methodology, this chapter also highlights several challenges, such as the differing hash strengths between component algorithms and the increased computational overhead of hybrid schemes that emerged during implementation.

4.1 Approach

The true hybrid schemes presented in Section 3.3 require that component algorithms meet certain conditions before they can be successfully integrated. If both components are directly specified like in Falcon–RSA (Fig. 3.7) or only a single component is specified like RSA in FS–RSA (Fig. 3.6), the requirements for the hybrid scheme are explicit. When unspecified like in FS–FS (Fig. 3.5) or the FS in FS–DSA (Fig. 3.8), the true hybrid scheme requires that the component algorithm uses the FS transform in a specific way. Thus, the first step in implementing the true hybrid schemes is to identify potential signature algorithm candidates that meet this requirement.

Identifying FS compatible signature algorithms is non-trivial without detailed knowledge of how the algorithm functions. Several signature algorithms use the FS transform; however, it is unreliable to assume compatibility with the true hybrid schemes based solely on this. From our experience implementing FS variants in this work, the component algorithm requires a symmetry in the derivation of the challenge c during signing and the recreation of the challenge based on critical values in the signature during verification. Thus, the shared c used to create the signature must be present in some form in the signature. Any transformative steps applied to c during signing must be reversible during verification using

only the signature, message, and public key. This symmetry is critical because c is used to reconcile the original commitment ω value(s) during verification. For correctness, every true hybrid scheme follows this symmetric pattern (i.e., the FS transform) to some degree. Section 3.3 and Table 3.1 provide further detail and list the compatible signature algorithms selected for this work, respectively.

Once the FS-compatible algorithms are identified, their original Sign and Verify operations must be modified to work with the specified true hybrid scheme. This involves separating both operations into sub-operations that yield the specific output needed to successfully complete the hybrid scheme. These sub-operations are already inherent in any FS-based component algorithm, so this separation is to account for and denote core aspects of the component algorithms and does not in itself introduce a change.

For the hybrid scheme's Sign operation, the component FS algorithm's Sign operation [87] is separated into an ω *generation* function and a *confirm shared challenge* function. The templates for both are informally described in Figure 4.1. The ω generation function yields the FS signature algorithm's commitment ω while the confirm shared challenge function takes the shared challenge c and generates the rest of the component algorithm's signature (i.e., z). The division between the ω generation function and the confirm shared challenge function pivots around the creation of the shared challenge c, which may be affected by the generalized hybrid approach. For example, in the FS–FS case, this c is the product of a hash function that takes as input the ω value, the digest of the message m, and the ω equivalent of the other component algorithm as input. This challenge is then used as input to the confirm shared challenge function specific to the FS component algorithm.

Algorithm 37 GenOmega$_{FS}$

Require: sk

1: $rand \leftarrow_{\$}$ Rand
2: $\omega \leftarrow g(sk, rand)$
3: **return** $(\omega, rand)$

Algorithm 38 ConfirmC$_{FS}$

Require: $c, rand, sk$

1: $z \leftarrow f(c, rand, sk)$
2: **return** z

Figure 4.1. Generalized Template for Converting Original FS Sign Operation into Sub-operations Required for True Hybrid Schemes

The shared challenge introduces a point of friction when integrating component algorithms. First, the hash function between both signature algorithms may differ entirely (e.g., SHAKE128 vs. SHA256). Second, the majority of the PQ algorithms in Section 2.3 use a SHA-3 hash function with extendable output which means the length of the challenge can differ between algorithms even if the same hash function is used. These differences must be reconciled for implementation. Table 4.1 lists the hash functions and challenge lengths for each FS-based component algorithm implemented in this paper.

Similar to the component Sign operation, the FS algorithm's Verify operation is separated into a ω' *reconciliation* function and a *verify challenge* function as depicted by the template in Figure 4.2. The ω' reconciliation function retrieves the original commitment ω from the received signature and the public key. The verify challenge function represents the additional steps that the FS-based component algorithm uses to verify that the signature is correct, which is signified by a Boolean return value b.

It is important to note that Figure 4.2 only depicts the operations unique to verifying a signature for a FS-based component algorithm and does not describe the additional verification steps specific to the true hybrid scheme.

Algorithm 39 RecOmega$_{FS}$

Require: $(c, z), pk$

1: $\omega \leftarrow Rec(pk; c, z)$
2: **return** ω

Algorithm 40 VerifyC$_{FS}$

Require: $(c, z), m, pk$

1: $b \leftarrow \text{verify}(pk; c, z, D(m))$
2: **return** b

Figure 4.2. Generalized Template for Converting FS-Based Component Verify Operation into Sub-operations Required for True Hybrid Schemes

Even with the provided template, other issues specific to a component algorithm can complicate integration. To demonstrate this, Figures 4.3 and 4.4 illustrate applying this methodology to the Dilithium signature algorithm originally described in Figure 2.4. For security and correctness, Dilithium uses rejection sampling to ensure that the final signature meets certain criteria (see line 8 of Algorithm 43). Normally, this rejection loop would be interior to the algorithm's original Sign operation; however, the rejection loop is split between both sub-operations within the true hybrid scheme. As a result, the rejection loop must be replicated at a higher level so that any rejection in the signature increments the counter κ and

Algorithm 41 Partial Sign_{FS} = Dilithium

Require: m, sk

$\kappa := 0$
$(z, h) := \perp$
while $(z, h) = \perp$ **do**
    $(\omega, \mu) \leftarrow \text{GenOmega}(m, sk, \kappa)$
    Generate shared c.
    $(z, h) \leftarrow \text{ConfirmC}(c, \mu, sk)$
    $\kappa = \kappa + 1$
end while
return (z, h, c)

Algorithm 42 GenOmega_{FS} = Dilithium

Require: m, sk, κ

$A \in R_q^{k \times \ell} := \text{ExpandA}(\rho)$
$\mu \in \{0,1\}^{384} := \text{CRH}(tr \| m)$
$\rho' \in \{0,1\}^{384} := \text{CRH}(K \| \mu)$
$y \in S^{\ell}_{\gamma_1 - 1}$
$w := Ay$
$w_1 := \text{HighBits}(w, 2_{\gamma_2})$
return (w_1, μ)

Algorithm 43 ConfirmC_{FS} = Dilithium

Require: c_1, μ, sk

$c \in B_{60} := H(\mu, c_1)$
$z := y + cs_1$
$(r_1, r_0) := \text{Decompose}_q(w - cs_2, 2\gamma_2)$
if $(\|z\|_\infty \geq \gamma_1 - \beta) \vee (\|r_0\|_\infty \geq \gamma_2 - \beta) \vee (r_1 \neq w_1)$ **then**
    $(z, h) := \perp$
else
    $h := \text{MakeHint}_q(-ct_0, w - cs_2 + ct_0, 2\gamma_2)$
    if $(\|ct_0\|_\infty \geq \gamma_2) \vee$ (# of 1's in h is $\geq \omega$) **then**
        $(z, h) := \perp$
    end if
end if
return (z, h)

Figure 4.3. Example of Converted Dilithium FS Sign Operation for Use in FS-based True Hybrid Schemes. Algorithm 41 represents the Dilithium-specific portion of an FS-based hybrid scheme.

restarts the signing operation (see Algorithm 41). This is not captured by the generalized template in Figure 4.1 as it is specific to the Dilithium signature algorithm and could be potentially missed during implementation. Additionally, extracting the rejection loop as shown can introduce additional computational overhead that is not present in the original signature algorithm. For example, if the sub-operations are implemented as function calls and the average number of rejection loop iterations is non-negligible, this will create a performance disparity between the converted and original signature algorithm with the former performing worse due to the increase in function calls.

Algorithm 44 RecOmega$_{FS}$ = Dilithium

Require: $m, (z, h, c), sk$

1: $A \in R_q^{k \times \ell} \coloneqq \text{ExpandA}(\rho)$
2: $\mu \in \{0,1\}^{384} \coloneqq \text{CRH}(\rho \| t_1 \| m)$
3: $w_1' \coloneqq \text{UseHint}_q(h, Az - ct_1 \cdot 2^d, 2\gamma_2)$
4: **return** (w_1', μ)

Algorithm 45 VerifyC$_{FS}$ = Dilithium

Require: $\mu, \omega, (z, h, c)$

1: **if** $(\|z\|_\infty < \gamma_1 - \beta) \wedge (c = H(\mu \| w_1')) \wedge$ (# of 1's in h is $\leq \omega$) **then**
2: **return** 1
3: **end if**
4: **return** 0

Figure 4.4. Example of Converted Dilithium FS Verify Operation for Use in FS-based Generalized Hybrid Schemes

The template also masks the implementation complexity by generalizing the input and output of each sub-function. For instance, Dilithium requires several variables (e.g., μ, y) that are defined in the ω generation function outside of those listed in the template. This requires the calling function to allocate memory to store and pass critical variables between both sub-operations and to securely de-allocate said memory once signing is completed.

4.2 Challenges

Throughout the adaptation process, we encountered several issues such as FS compatibility, differences in hash strength, and increased computational overhead that affect the integration of particular signature algorithms into the true hybrid schemes. Each required a solution that not only addressed the issue, but also minimized the impact to the original, underlying signature algorithms. The process of integration also exposes several critical variables and inner functionality that can directly impact the correctness of the component algorithms. Even a subtle, minor change at this level can invalidate the algorithm's original security proof and introduce unforeseen flaws and vulnerabilities. Therefore, for successful integration, it is essential to fully understand not only how the algorithm works, but also how any deviations or modifications impact the security of the original algorithm.

4.2.1 Fiat–Shamir Compatibility

As discussed in Section 4.1, FS compatibility refers to identifying whether or not a potential component signature algorithm uses the standard version of the FS transform as specified in Section 2.1.2. This is the first step in adapting a non-specified signature algorithm into

one of the true hybrid schemes. Once identified, the generalized templates in Figures 4.1 and 4.2 can be applied to a component algorithm. This transformative process does not pose a significant issue for the majority of the true hybrid schemes where only a single component algorithm is required to be FS-based. For these hybrid schemes, the shared challenge c that is generated during the Sign operation is either based directly on the hash of the commitment ω and the digest of the message $D(m)$, or the hash of the same values combined with additional elements of the other component algorithm. The challenge c is then used in the same way in the confirm challenge function as in the component FS signature algorithm's Sign operation. This is beneficial for algorithms that apply custom encoding/expansion functions to the challenge c to minimize signature size, like Dilithium (see Algorithm 2.4), as it requires no additional changes to the component FS algorithm beyond applying the generalized templates.

Unlike the other true hybrid schemes, the FS–FS hybrid scheme requires both component algorithms to use the FS transform. During the Sign operation, both algorithms generate a shared challenge c based on their respective ω values and the digest of the message. The challenge c is then included unmodified in the final signature. This implies that, during verification, the FS algorithms are able to reconcile their respective ω' values based on only the hybrid signature and the pk. The ω' values are then combined with a digest of the original message to form c'. Of the tested signature algorithms, this is possible for both MQDSS and qTESLA; however, Dilithium does not use its original challenge c during verification. Instead, a "hashing to a ball" function is applied during the Sign operation that takes ω as input and outputs an array B_{60} . This array is then sent as part of the signature and is used in conjunction with other included information (e.g., h) to compute the high order bits needed for verification. While correct within the Dilithium algorithm, it is impossible to derive the original ω value needed for successful verification in the FS–FS scheme from this output alone.

The product of the "hashing to the ball" function, the encoded challenge c, creates an asymmetry between Dilithium and the other FS-based schemes that complicates integration. It is possible to treat the encoded challenge, c, as part of the rest of the signature, z, and pass it in tandem with the original challenge; however, this violates the hybrid verification property. The encoded challenge c used to reconcile ω during verification is not the same challenge used by the other component algorithm. Therefore, the final verification step

in the FS–FS signature scheme where the generated challenge c is compared to the hash of both reconciled ω values and the digest of the message would only apply to the other signature algorithm and not both, as required. This problem can be avoided by only sending the non-encoded challenge, c, in the signature and applying the "hash to the ball" function to this value during the ω' reconciliation function. This solution does change the original Dilithium verification procedure and affects efficiency by adding additional operations; however, it allows other FS compatible signature algorithms to be combined with Dilithium in the FS–FS true hybrid scheme.

4.2.2 Differing Hash Strengths

Another common issue encountered during implementation is differing hash strengths. Each signature algorithm submitted to the Post-Quantum Cryptography Standardization project defines recommended security parameters for each NIST security level as part of their specification. These parameters identify the hash function used as well as the size of its output, if extendable. When an algorithm is incorporated into a true hybrid scheme like FS–FS, there may be a mismatch between these parameters even with matching NIST security levels. Table 4.1 provides an overview of the hash functions used by each FS-compatible PQ candidate.

Table 4.1. Hash Strengths of FS-Compatible Schemes

Name	Parameters	Hash Function	$\lvert c \rvert$ (bits)
CRYSTALS–DILITHIUM	1024x768, 1536x1280	SHAKE256	1088
qTESLA	p–I	SHAKE128	256
qTESLA	p–III	SHAKE256	384
MQDSS	31–48	SHAKE256	256
MQDSS	31–64	SHAKE256	384

In this work, the strongest hash function between the two component algorithms is used to generate the challenge within all true hybrid schemes. This involves using newer hash functions like SHAKE with classic algorithms like DSA and RSA. Currently, no SHA-3 XOF (i.e., SHAKE128 or SHAKE256) is approved for use with these signature algorithms; however, past efforts by the Limited Additional Mechanisms for PKIX and SMIME (LAMPS) working group of the Internet Engineering Task Force (IETF) have sought to create new

algorithm identifiers to promote use of the SHAKE function family with both RSA and ECDSA [88]. For USG systems, NIST only permits hash algorithms that are specified in FIPS 180, the Secure Hash Standard, to be used to create message digests for approved digital signature algorithms [24]. Currently, only fixed length SHA-3 algorithms, SHA3–224, SHA3–256, SHA3–384 and SHA-512, are approved as alternative hash functions; however, guidance for "using the XOFs will be provided in the future" [89].

Component signature algorithms may also apply *custom* hash algorithms to create the challenge like line 10 in qTESLA's Sign operation or line 9 in Dilithium's Sign operation. When possible, this work defaulted to both the stronger hash algorithm and the longer challenge length between both component algorithms to generate the shared challenge. The output of this (e.g., c) was then used as input for any custom hash functions or encoding in order to preserve the correctness of the component algorithm while minimizing changes. This is only required for the FS–FS true hybrid scheme where more than one FS-compatible signature algorithm is combined. As shown in Table 4.1, no two FS-compatible signature algorithms use both the same c length and hash algorithm.

4.2.3 Overhead

Every change that is made to a component signature algorithm during implementation directly impacts the performance of both the algorithm and the hybrid scheme. Often, this change is linear. For example, replacing a component algorithm's original challenge c generation function (i.e., hash function) with one to generate the hybrid scheme's shared challenge c will uniformly increase the required computational load if the challenge function is of greater strength as detailed in Section 4.2.2. Another example is in the additional verification steps required for some of the true hybrid schemes. The FS–FS hybrid scheme includes the verification operations of the individual component algorithms *and* the comparison of c with a hash of both ω values and the message digest. While it may be possible to modify redundancies in the component algorithms' implementation verification operation, we made the design choice to minimize changes to all implementations of component algorithm operations. As a result, our implementation will always carry an increased overhead when compared to concatenated hybrid schemes; however, since the modified operations are not computationally expensive, the increase may be negligible.

CHAPTER 5:
Hybrid Digital Certificates

The PKI model facilitates secure entity authentication between parties by establishing a chain of trust from a root Certificate Authority (CA) to an end-entity. The model is used as a secure way to authenticate a public key to an individual entity for use in public key cryptography and provides the foundation for information security and digital identity for the majority of the Internet.

Several applications and their underlying protocols use the PKI model to distribute and manage public keys through the use of digital certificates. Each certificate contains a variation of the end-entity's public key, an expiration date, identifying information, and any specific cryptographic algorithm parameters [90]. The certificate is then signed by the public key of a trusted CA that both end points implicitly trust. The CA's signature is a guarantee that all the information is not only correct, but that the entity described in the certificate is in sole control of the related private key.

Given the size and complexity of modern PKI systems, root CAs do not issue end-entity certificates directly due to the risk involved. Instead, root CAs delegate issuing certificates to intermediate CAs by signing their corresponding digital certificates. For a user to validate any certificate, the digital signatures on every certificate in the chain are verified until reaching the implicitly trusted root CA (i.e., trust anchor). The entire operation fails if any errors, to include invalid or revoked certificates, are encountered in the chain.

While alternate certificate formats exist [91], the X.509 standard has become the default method for "securely binding the identity of an individual or device to a public key" in the PKI model [90]. This chapter specifically examines X.509 digital certificates and how they can be adapted to support hybrid digital signature schemes. A brief overview of X.509 certificates is provided and three hybrid certificate variants are cross-examined. Finally, the TLS 1.3 protocol is used to demonstrate how hybrid digital certificates can affect a commonly used protocol.

5.1 X.509 Certificates

RFC 5280 [90] specifies the format and semantics of X.509 certificates for use of PKI within the Internet. This standard has been repeatedly revised since 1988 as PKI systems evolved and required additional information to be included in a digital certificate. Today, certificates have three versions, with each adding support for additional fields. As a rule, the certificate's version number should match the minimum version that supports the fields being used.

The X.509 certificate can be broken into three parts: the *tbsCertificate*, the signature algorithm field, and the signature. The tbsCertificate, short for "to be signed", contains all information that the CA will sign and includes critical fields such as the "names of the subject and issuer, a public key associated with the subject, a validity period" [90]. Any information outside of the tbsCertificate will not be signed and can be changed without affecting the verification process (aside from the signature itself). The second part of the certificate is the *signatureAlgorithm* field. This field identifies both the signature and hash algorithms used to generate the signature. As such, it must match the signature algorithm identifier inside the tbsCertificate to be considered valid; however, it is important to note that this outer field is not part of the data that will be signed and is not protected from malicious or benign modification. The third part of the certificate is the signature. This is the product of applying the signature algorithm identified in the signatureAlgorithm field to the tbsCertificate.

Transmitting binary data like cryptographic keys over textual transports requires application and library developers to encode any binary data into human-readable text. Prior to IETF efforts to standardize this encoding in 1993 [92] with PKCS, developers often created competing formats which created situations where a certificate from one application would be incompatible with another application. Textual encoding of PKI structures which include X.509 certificates are covered by RFC 7468 [93]. Additionally, X.509 digital certificates fall under PKCS #7 and use attributes defined in either PKCS #9 or #10 depending on the certificate's intended purpose.

PKCS #7 defines the Cryptographic Message Syntax (CMS) which is a general syntax for storing and signing encrypted data to include digital signatures [94]. The CMS syntax uses Abstract Syntax Notation One (ASN.1) structures for each data field and employs a specific

encoding format known as the Distinguished Encoding Rules (DER). DER is defined in the International Telecommunication Union (ITU) X.690 standard and provides exactly one way to represent any ASN.1 value as an octet string [95]. These structures or classes can also store multiple values and data types. For example, the Subject Public Key Information field contains both the algorithm identifier used to create the public key as well as the key itself.

Figure 5.1 provides an overview of a X.509 certificate. Each field listed in the tbsCertificate is a DER encoded ASN.1 structure. The fields listed in blue are found in every version of the X.509 standard. Of these, the *version* field describes the version of the X.509 certificate (e.g., version 1, 2, or 3). The *serialNumber* field is assigned by the CA and must be unique for every issued certificate. The *issuer* field contains the distinguished name of the certificate's issuer (e.g., CA) and may contain additional attributes like the CA's common name, location, and organizational identifier [90]. The combination of the certificate's *serialNumber* and the *issuer* field determine the uniqueness of the certificate. The window of time that a certificate should be considered valid is recorded in the validity field. Specifically, this field contains the "time interval during which the CA warrants that it will maintain information about the status of the certificate" [90]. Similar to the issuer field, the *subject* field contains the distinguished name and any additional descriptive attributes of the entity associated with the public key. Finally, the *subjectPublicKeyInfo* field is an ASN.1 sequence containing the signature algorithm's public key and an algorithm identifier which itself contains the algorithm's unique Object Identifier (OID) [90].

The two fields displayed in purple are introduced in the second version of the X.509 standard. Their purpose is to allow "reuse of the issuer and/or subject names over time"; however, the current X.509 standard recommends that names should never be reused for different entities [90]. As such, these fields are considered historical artifacts and should not be generated for new digital certificates. Even so, programs that handle digital certificates must be able to parse the field and should handle it as a bit string sequence of arbitrary length [90].

In version 3, *extensions* were added to allow additional attributes to be associated with X.509 certificates. These include relationships between CAs, key identifiers, and private extensions that are unique to a particular community of users [90]. Each extension includes

a unique OID and a corresponding ASN.1 structure. Some extensions like key usage are considered standard and are widely supported. This extension defines the purpose of the public key stored in the certificate. This is useful in situations where a key can possibly be used for multiple purposes (e.g., encryption, signing), but should be restricted to one use [90]. Another common extension is the subject alternative name which allows multiple identities to be bound to the identity listed in the subject name field. This is useful for services that choose to use a single certificate for multiple entities like a Content Delivery Network (CDN) that provides access to multiple domains or for web servers with multiple virtual hosts. Instead of issuing a certificate for each unique domain name which would incur a large key management overhead, a CDN can use a single digital certificate for multiple domains or sub-domains as long as they are listed in the subject alternative name extension. It is crucial that an issuing CA independently verify any information listed in this extension, because this information is considered to be "definitively bound to the public key" and can be substituted for any information listed in the subject name field [90].

Separate from commonly used extensions, private extensions are not formally standardized and are intended for limited use within a specific community of users. In general, these extensions provide developers with flexibility in expanding PKI functionality using X.509 digital certificates. For example, hybrid digital signature schemes require at least two public keys to verify a hybrid signature. For compatibility with existing protocols, this information must be stored within a X.509 certificate; however, the current standard does not support multiple public keys within the same certificate [62]. In order to comply with current standards and minimize impact to existing PKI systems, a CA could use a private extension to store the second public key. This approach prevents the certificate from being rejected by systems conforming to RFC 5280; however, it also carries the risk that independent developers could create competing private extensions that conflict.

Both public and private extensions can be marked as either critical or non-critical. If an extension is marked as critical, the verifying recipient must support the extension and be able to process its data. For example, if the private extension storing a hybrid signature scheme's second public key is marked as critical, the recipient must both recognize the extension and be able to process the key. If the critical extension is not recognized or it contains data that cannot be processed, the certificate must be rejected [90]. This approach can be used to force a verifier to recognize that a hybrid signature scheme is being used even if the hybrid

scheme does not guarantee hybrid verification. While effective, a certificate that marks a hybrid extension as critical can also alienate users outside of the CA's community who do not recognize the extension or are unable to process the data.

5.2 Design Considerations for Hybrid Certificates

Several hybrid X.509 digital certificates have been proposed in recent years to aid the PQ transition. Each approach the problem in similar ways due to the nature of ASN.1 structures and the flexibility of the X.509 standard. Regardless of the hybridization technique used in a signature scheme, the signatures presented in a hybrid certificate must be tied to the names of the subject and issuer, the corresponding public keys associated with the subject, and a validity period (e.g., *tbsCertificate*) [90]. This requirement limits divergence in terms of certificate structure and is one of the reasons for the stability of the X.509 standard. Even with limited divergence, hybrid certificates have unique characteristics that must be considered for any design. This section examines design considerations for hybrid certificates and introduces three hybrid certificate constructions found in related works.

5.2.1 Backwards Compatibility

The goal of backwards compatible designs are generally to allow a system to transition from one algorithm to another without interrupting cryptographic security. As such, the differences between hybrid certificate designs often involve the desire to maintain backwards compatibility with existing PKI. When designing a hybrid certificate, nesting (Section 3.2.2) is often used to allow a legacy system to verify only one of the included signatures by ensuring the outer signature is the one that is supported. Depending on the certificate construction, this also has the added benefit of not requiring the standardization of new signature algorithm identifiers since it is possible to use custom certificate extensions.

A potential downside to maintaining backwards compatibility is that there is no impetus for new standards to be adopted that specify hybridization. For those that wish to adopt hybrid certificates, this can lead to incompatible systems that automatically default to a less secure legacy mode. Additionally, protocols that use nested hybrid digital certificates require more overhead than non-hybrid signature schemes as at least two signatures and two public keys must be transmitted with each certificate. Depending on the limitations of

the legacy system, to include the necessary underlying protocol and cryptographic libraries may require more memory than available or practical. A simpler solution would be for a system to have multiple digital certificates and use a protocol to negotiate which certificate to use in a given instance.

5.2.2 Verification Considerations and Critical Extensions

Hybrid certificates are designed to tie a cryptographic identity to multiple public keys within a single certificate; however, there is no requirement to use every included signature algorithm and key within a hybrid certificate in every situation. While this allows for backwards compatibility and more flexible certificate designs, it also creates a situation where there is no guarantee that a verifier uses more than a single signature algorithm during verification. Depending on the context, a developer may want to achieve a guarantee that every included signature algorithm is used to successfully verify a signature. As explained in Chapter 3, not every hybrid signature scheme achieves hybrid verification. In these situations, it may be possible to use hybrid certificates to achieve a similar, but limited and non-cryptographic notion. For example, a signer using a hybrid certificate can store the second public key and signature in critical extensions. By the RFC standard, the verifier would have to parse and accept the extension in order to successfully verify any of the included signatures. Unlike hybrid verification, this requires that the application parsing the certificate is honest as it would still be possible to verify only a single signature even after acknowledging the second signature existed.

Alternatively, a hybrid scheme that achieves hybrid verification (i.e., true hybrid schemes) can be used in any certificate design without relying on critical extensions. The second public key can even be split between two standard digital certificates without requiring additional modification. Essentially, hybrid verification ensures that even if a system tried to accept a critical extension it did not support, it would be cryptographically impossible for it to verify only one of the signatures. Naturally, this comes at the cost of backwards compatibility in that the hybrid scheme would require a new algorithm OID.

5.2.3 Size Differences

As discussed in Chapter 3, a hybrid scheme will require a significant increase in the overall size of a digital certificate. At the very least, the certificate will need to contain two

or more public keys and be signed by two or more signature algorithms. While the X.509 standard [90] does not place a limit on the total size of a certificate, several protocols do have limits on the fields within a certificate. For example, the TLS protocol limits the public key size to $2^{24}-1$ bytes and the signature size to $2^{16}-1$ bytes [2]. Certain algorithm combinations already exceed this size limitation. Figure 5.1 depicts the concatenated signature sizes of the algorithm combinations considered in this work. This figure includes the sizes for both the true hybrid schemes and a simple concatenated scheme using the technique described in Section 3.2.1. Additionally, the signature sizes reflect the ASN.1 structures used to pack the bit strings produced using our implementation. This may vary in the future as our implementation is only a proof-of-concept. Of the listed combinations, only the MQDSS-31-64_qTesla-p-III combination of the FS–FS true hybrid scheme produced a signature that was larger than the TLS limitation.

Table 5.1. Signature Sizes of Considered True and Concatenated Hybrid Schemes (bytes)

Name	True Hybrid	Concatenated
Dilithium 2, RSA3072	2292	2428
MQDSS-31-48, RSA3072	28752	28784
qTesla-p-I, RSA3072	2944	2976
Falcon-512, RSA3072	1018	1050
Dilithium 3, RSA3072	2949	3085
MQDSS-31-64, RSA3072	60264	60312
qTesla-p-III, RSA3072	38768	6048
Falcon-1024, RSA3072	1632	1664
Dilithium 2, DSA3072_1	2116	2116
MQDSS-31-48, DSA3072_1	28472	28472
qTesla-p-I, DSA3072_1	2664	2664
Dilithium 3, DSA3072_1	2773	2773
MQDSS-31-64, DSA3072_1	60000	60000
qTesla-p-III, DSA3072_1	5736	5736

Dilithium 2, DSA3072_2	2116	2116
MQDSS-31-48, DSA3072_2	28472	28472
qTesla-p-I, DSA3072_2	2664	2664
Dilithium 3, DSA3072_2	2773	2773
MQDSS-31-64, DSA3072_2	60000	60000
qTesla-p-III, DSA3072_2	5736	5736
Dilithium 2, DSA3072_3	2168	2116
MQDSS-31-48, DSA3072_3	28508	28472
qTesla-p-I, DSA3072_3	2700	2664
Dilithium 3, DSA3072_3	2909	2773
MQDSS-31-64, DSA3072_3	60048	60000
qTesla-p-III, DSA3072_3	5784	5736
Dilithium 2, P-256_1	2108	2108
MQDSS-31-48, P-256_1	28464	28464
qTesla-p-I, P-256_1	2656	2656
Dilithium 3, P-384_1	2797	2797
MQDSS-31-64, P-384_1	60024	60024
qTesla-p-III, P-384_1	5760	5760
Dilithium 2, P-256_2	2108	2108
MQDSS-31-48, P-256_2	28464	28464
qTesla-p-I, P-256_2	2656	2656
Dilithium 3, P-384_2	2797	2797
MQDSS-31-64, P-384_2	60024	60024
qTesla-p-III, P-384_2	5760	5760
Dilithium 2, P-256_3	2168	2108
MQDSS-31-48, P-256_3	28508	28464
qTesla-p-I, P-256_3	2700	2656
Dilithium 3, P-384_3	2933	2797
MQDSS-31-64, P-384_3	60072	60024

qTesla-p-III, P-384_3	5808	5760
Dilithium 2, MQDSS-31-48	30412	30444
Dilithium 2, qTesla-p-I	4604	4636
MQDSS-31-48, qTesla-p-I	30960	30992
Dilithium 3, MQDSS-31-64	62581	62629
Dilithium 3, qTesla-p-III	8317	8365
MQDSS-31-64, qTesla-p-III	65544	65592

Even if a protocol has restrictions on the size of either the public key or signature, it is possible to bypass a restriction by separating the offending fields into two or more extensions; however, this requires that the recipient is able to correctly recreate the separated fields into the original data structure in order for the scheme to work. For example, a signature can be separated into the signature field and a critical private extension. Upon receiving the message, the verifier would first need to correctly reassemble the signature by parsing both the critical extension and the signature fields before verification. Additionally, developers must also consider why size limitations were applied to protocols in the first place as alterations may lead to unexpected inefficiencies or critical vulnerabilities.

5.2.4 Hybrid Certificate Variants

The remainder of this section introduces three hybrid certificate designs gathered from implementations, draft standards, and published patents. It is important to emphasize that none of the designs are an original work of this paper. Our goal is to highlight different approaches and discuss the advantages and disadvantages of each.

OQS Hybrid Certificate

As part of their efforts to develop and prototype "quantum-resistant cryptography" [96], the Open Quantum Safe (OQS) project created a fork of the OpenSSL cryptographic library with PQ algorithm support enabled. This project also includes hybrid algorithms in which PQ algorithms can be combined with RSA or ECDSA based on the intended NIST security level (e.g., L1, L3, or L5). In their scheme, the public keys for both algorithms are concatenated and stored in the subjectPublicKeyInfo field. Then, both algorithms independently sign

the *tbsCertificate* and their corresponding signatures are concatenated and attached to the certificate in the standard signatureValue field. Figure 5.2 depicts the certificate structure with modified fields in bold.

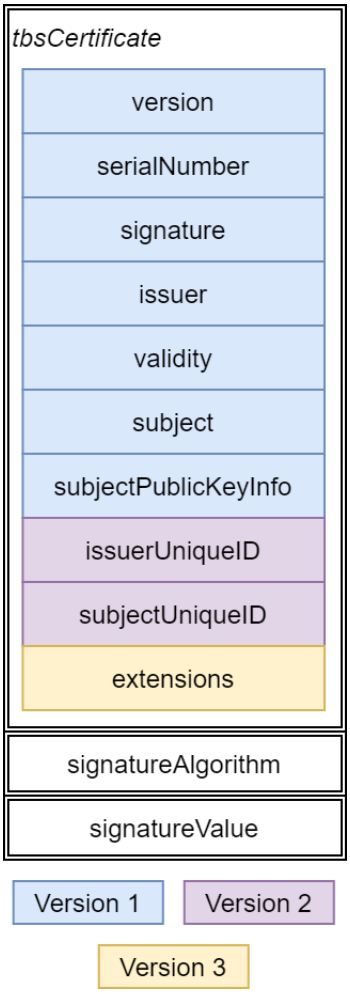


Figure 5.1. X.509 Certificate Structure

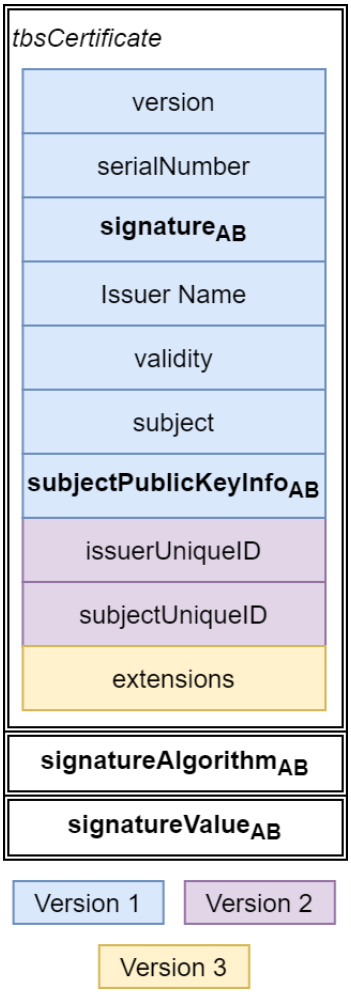


Figure 5.2. OQS X.509 Hybrid Certificate Structure

This option allows for backward compatibility as it does not add any private critical extensions, but it still requires that the recipient is able to parse the hybrid signature algorithm identifier and both concatenated fields. Additionally, simple concatenation does not achieve non-separability. As a result, it is up to the individual application developers to ensure that both signatures contained within the certificate are verified.

ISARA Patent

In 2016, the ISARA corporation filed a patent [62] detailing how a digital certificate could be used simultaneously with multiple cryptosystems. The authors describe how an enterprise PKI environment can migrate from one signature algorithm to another through a series of transitional steps. Figure 5.3 depicts one of the transitional X.509 certificate structures listed in the patent. This certificate design uses a private extension with the same ASN.1 structure as the subjectPublicKeyInfo field to store the second public key (i.e., subjectPublicKeyInfo$_B$). This second public key is associated with the target signature algorithm that the PKI system is transitioning to (e.g., PQ signature algorithm) and is used to sign the tbsCertificate as per the standard process (i.e., signatureValue$_B$).

At this point the inner tbsCertificate, signatureAlgorithm$_B$, and signatureValue$_B$ are technically a complete X.509 certificate (depicted by the magenta box in Figure 5.3); however, the certificate would be invalid because Signature$_B$ uses the public key stored in the private extension instead of the public key stored in Public Key Information$_A$. As such, this inner X.509 certificate is then used as input to the first signature algorithm using the public key stored in Public Key Information$_A$. This signature algorithm is the legacy algorithm the PKI system is transitioning from and the signature from this process is stored in Signature$_A$.

The result is a single hybrid X.509 certificate that is backwards compatible with legacy systems *assuming* the legacy system ignores the private extension, inner signatureAlgorithm$_B$, and signatureValue$_B$ fields when parsing the certificate during verification. This requires that the private extension is not marked as critical and that the inner signature algorithm identifier (depicted in blue in Figure 5.3) matches the outer signatureAlgorithm$_A$ field. This precludes the ISARA certificate design from using hybrid identifiers in the traditional signature algorithm identifier field as it would disrupt backwards compatibility. The inner signatureAlgorithm$_B$ field could reflect a hybrid scheme if and only if it also describes the signature algorithm used to create signatureValue$_B$.

Full implementation of this particular certificate design would require a modification to the standard signing process currently being used by cryptographic libraries. The modified signing procedure would require a system to recognize and process the inner private extension, signatureAlgorithm$_B$, and signatureValue$_B$ fields in order to fully support hybridization. The system would then need to independently verify both signatures and return the result.

Legacy systems and libraries that do not support hybridization would need to be tested to ensure that they correctly ignore the additional inner fields and handle any potential size limitations on the total certificate size.

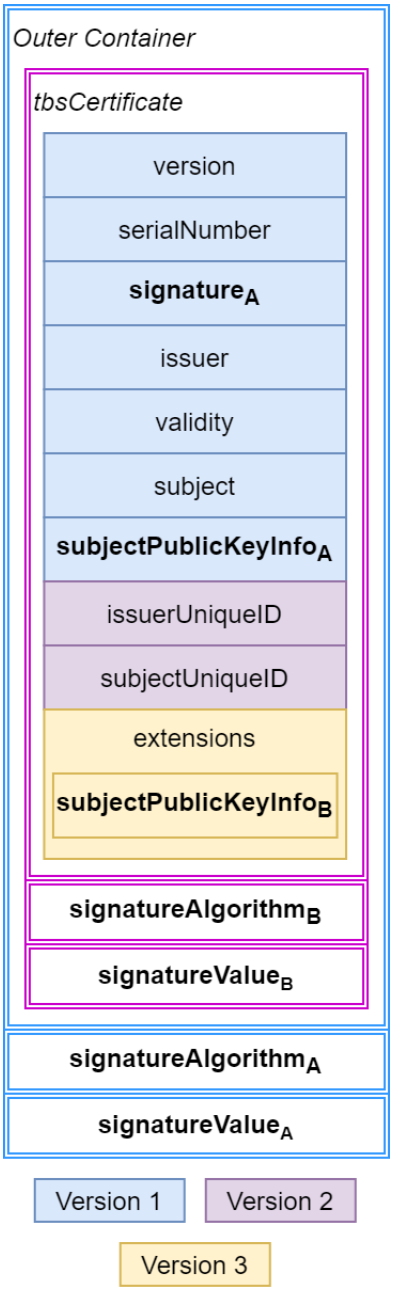


Figure 5.3. ISARA X.509 Hybrid Certificate Structure

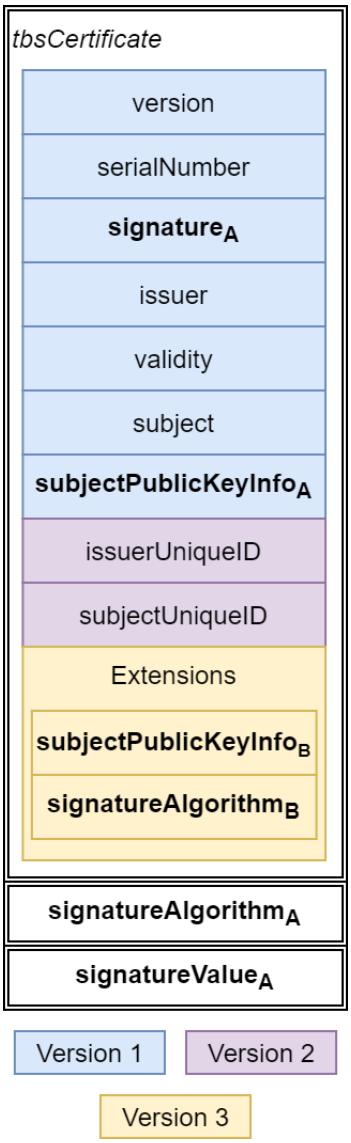


Figure 5.4. CROSSING X.509 Hybrid Certificate Structure

DTU OpenSSL

In 2019, the Collaborative Research Center (CROSSING) at Technische Universität Darmstadt (DTU) released a fork of the OQS OpenSSL [20] and Java-based BouncyCastle [21] projects that included support for backwards-compatible hybrid certificates. Both libraries implement a hybrid scheme using private certificate extensions to store both the PQ public key and signature. This nested approach resembles a 2018 RFC draft [97] which also uses X.509 version 3 certificate extensions in a similar fashion; however, the CROSSING draft differs in how it handles the ASN.1 structures, OID values, and signature algorithm information.

Figure 5.4 depicts the nested certificate structure with modified fields in bold. The subjectPublicKeyInfo$_A$ field stores the legacy signature algorithm's public key and the hybridKey extension stores the PQ algorithm's public key and hybrid algorithm OID. The subjectPublicKeyInfo$_B$ extension stores the PQ algorithm's signature while signatureValue$_A$ stores the legacy algorithm's signature. To maintain backwards compatibility, the signature$_A$ field within the tbsCertificate and the signatureAlgorithm$_A$ field only contain the OID identifier associated with the legacy signature algorithm.

While similar to the ISARA patent, this certificate design follows a different procedural flow. First, a "dummy" tbsCertificate containing both public keys and a null signatureAlgorithm$_B$ field is signed by the PQ signature algorithm. The signatureAlgorithm$_B$ extension is then populated and the entire tbsCertificate is again signed by the legacy signature algorithm. During verification, the legacy algorithm's signature (e.g., signatureValue$_A$) is processed normally, but the same "dummy" certificate must first be recreated by setting the signatureAlgorithm$_B$ extension to null before verifying the PQ signature.

The CROSSING certificate design is backwards compatible assuming the legacy system can verify the outer signature and the hybrid extensions are not marked as critical. Additionally, unlike the ISARA patent, the CROSSING design only uses private extensions and does not rely on ambiguous inner certificate fields that may or may not be compatible with existing cryptographic libraries.

5.2.5 Legal Issues

Another consideration that must be taken into account when designing or evaluating hybrid certificates is whether or not a design has any legal restrictions or intellectual property claims, such as in the case of patents like the ISARA design in Section 5.2.4. These restrictions, if enforced, can significantly affect the monetary cost associated with using a particular design and open implementers to legal liability if handled incorrectly.

This chapter relies on information derived from the X.509 standard as described in RFC 5280 [90]. The IETF, the organization responsible for publishing RFCs, does not take a position "regarding the validity or scope of any intellectual property rights or other rights that might be claimed...or the extent to which any license under such rights might or might not be available" [98]. As such, it is the responsibility of individuals to verify if any intellectual property claims exist by directly contacting the authors of the RFC or by checking the litany of legal sources that publish patents and copyright information.

In the case of hybrid digital certificates, it is difficult to ascertain if the concept, as a whole, is protected by existing patents or copyrights. For example, the ISARA patent claims any cryptographic method where a digital certificate is received "comprising a plurality of signatures of a certificate authority" with private extensions containing a "second signature value field" and "a policy field comprising a policy comprising instructions" [62]. The policy field, which is described as either a bitmap or OID, contains a policy for processing the additional public keys or signatures in the X.509 certificate in *priority* order.

Upon initial review, the CROSSING design may infringe on this claim as it also uses private extensions to hold a second signature value field. Even so, the CROSSING design does not contain a policy field that contains information prioritizing verification of one signature over the other. Instead, it relies on the verifier not processing unsupported private extensions (i.e., non-critical) for backwards compatibility. The uncertainty of infringement is further complicated by the fact that the ISARA patent mentions that other systems outside of their design exist that include "one or more additional public keys and/or one or more additional signature values associated with one or more corresponding cryptosystems" [62]. This suggests ambiguity of scope on patent right related to various hybrid certificate constructs using multiple public keys.

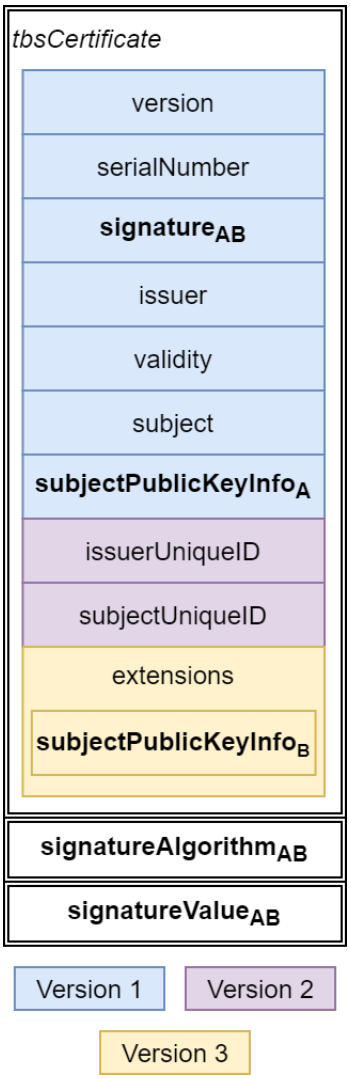


Figure 5.5. Proposed X.509 Certificate Structure for True Hybrid Schemes

Hybrid certificates designed around the true hybrid schemes described in Section 3.3 may potentially avoid the entire problem by using a modified design that combines the CROSSING and OQS certificates. Figure 5.5 depicts this certificate structure where the second public key is stored in a critical private extension and the scheme's signature is stored in the traditional signatureValue field. It is possible that this does not infringe on the patent because the true hybrid scheme does not rely on the certificate to determine which signature algorithm to verify (e.g., priority).

It is important to understand that the legal landscape surrounding cryptography is complex and continually changing as evidenced by several historical examples like the patents surrounding the origin of public key cryptography [52], [99]. Any material presented in this section should not be mistaken as an in-depth legal review of the intellectual property claims on hybrid certificates and any conclusions should not be taken as legal advice. Instead, this section highlights the legal ambiguity surrounding hybrid digital certificates and reinforces the need for standards that do not carry any intellectual property burden.

5.3 TLS 1.3 Authentication

This section examines server authentication using hybrid digital certificates within the TLS 1.3 protocol and is used as the basis for experiments in Section 7.2. The TLS 1.3 protocol is specified in RFC 8446 and is designed to create a secure channel between a client and server in order to prevent eavesdropping, tampering, and message forgery [2]. The secure channel is established using a *handshake* whereby the client and server exchange a sequence of messages that produce the security parameters for the connection. During the handshake, the server and client have to negotiate a ciphersuite, authenticate the identities of the server and (potentially) client, and establish the session keys that will symmetrically encrypt any application-layer traffic upon handshake completion [2]. This sequence can be divided into two distinct phases: the key-exchange and authentication [100]. Figure 5.6 depicts the messages exchanged in a basic TLS 1.3 handshake between a client and server assuming a new, initial connection (e.g., no resumption). This figure also indicates the cryptographic operations (i.e., Sign and Verify) associated with digital signatures that influence the performance during an initial handshake.

To begin the handshake, the client first sends the plaintext ClientHello message which starts the key-exchange phase. This message contains the supported ciphersuites and a list of optional extension requests that include a 256-bit randomly sampled nonce, one or more public key shares that will be used to generate a client handshake traffic key (i.e., key_share) and the signature algorithms that the client supports (i.e. signature_algorithms, signature_algorithms_cert) [2]. Upon receiving the ClientHello message, the server replies with a ServerHello message. This message contains the ciphersuite selected by the server and a freshly generated key share using the same algorithm of the public key sent by the client (i.e., key_share). At this point, the server is able to derive both the client and server

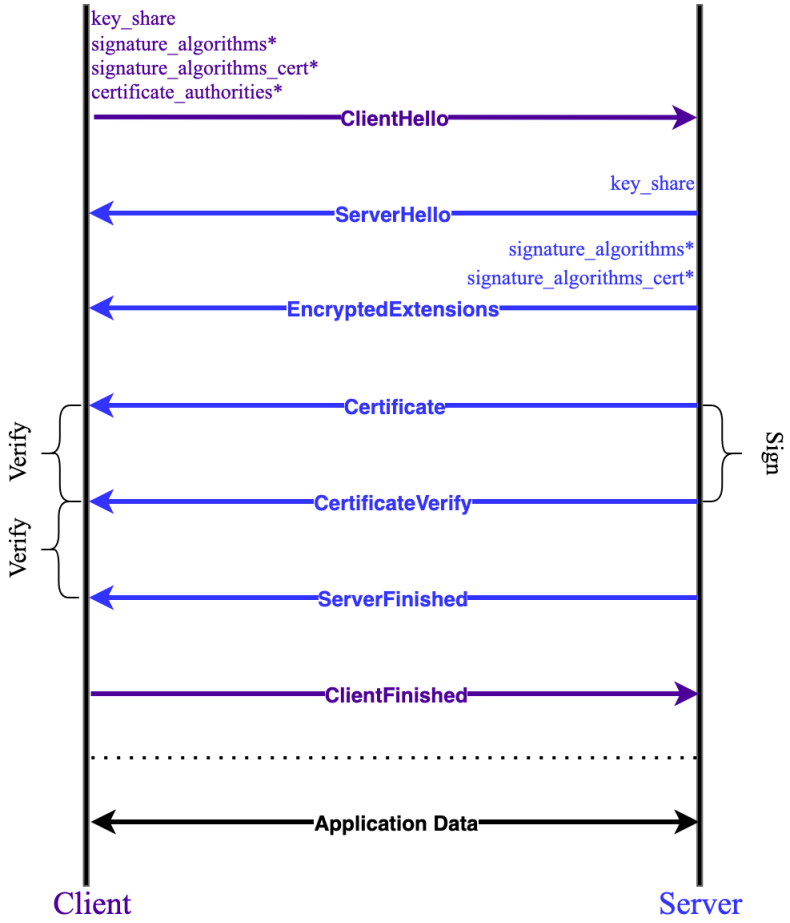


Figure 5.6. Basic TLS 1.3 Handshake. Adapted from [18, figure 2].

handshake traffic keys that will be used to encrypt the rest of the TLS handshake [100].

After the ServerHello message, the handshake transitions to the authentication phase and all subsequent messages are encrypted by either the client or server handshake traffic key. First, the server encrypts and sends the EncryptedExtensions message which contains extension responses to any extension requests in the ClientHello message. Unsolicited extensions are not allowed as the server only sends responses to requests [2]. Specifically, these extensions are not needed for negotiating the symmetric encryption key; however, they are encrypted to protect against eavesdroppers and man-in-the-middle attacks.

TLS 1.3 supports several optional extensions that provide expanded capabilities over older versions of TLS. Two of these extensions, signature_algorithms and signa-

ture_algorithms_cert, govern which signature algorithms may be used in digital signatures [2]. Specifically, the signature_algorithms extension applies to any signatures in the CertificateVerify message while the signature_algorithms_cert extension applies to signatures in digital certificates. If the signature_algorithms_cert lists the same signature algorithm as the signature_algorithms extension, it may be excluded. These extensions are required for server authentication and, if not present in the ClientHello message, the server must abort the handshake [2].

The Certificate message contains, at a minimum, the X.509 digital certificate of the server which contains the identity and long-term public key of the server. This message can also contain a certificate_list which is a chain of digital certificates that certify the one immediately preceding it [2]. The server then calculates the session hash, which is a continually updated hash of all prior handshake messages (e.g., message transcript). The server signs this hash using the certificate's private key and sends the signature to the client via the CertificateVerify message [2].

Upon receiving the Certificate message, the client must verify the certificate's signature. If the message contains a certificate_list, the client must also verify each entry. If the certificate is successfully verified by tracing it back to a trusted source (e.g., CA), then the client knows that the public key and identity are linked and valid; however, the actual handshake is still not authenticated until the CertificateVerify message is verified. Once this signature is verified by the client, the identity of the server is tied to the identity listed in the digital certificate contained in the Certificate message. By signing the entire transcript to include the ClientHello, TLS 1.3 avoids classic downgrade vulnerabilities that were prevalent in previous versions of the TLS protocol.

In terms of efficiency, the signature algorithms used for authentication can significantly impact the TLS protocol. As noted in Figure 5.6, the total handshake time is influenced both by the Sign and Verify times of the signature algorithm. Unlike the signature of the server's certificate which is generated once when the certificate is created, the server must sign the hash of the message transcript in real time before sending it to the client via the CertificateVerify message. The total size of the server's digital certificate also impacts the handshake completion time. The larger a certificate, the more time is needed to transfer the certificate to the client. Upon receiving the certificate and signature, the client then must

verify the signature is correct before sending the ClientFinished message to the server. Only at this point is the handshake considered complete.

Using more than one signature algorithm during a TLS handshake drastically increases both the size of the digital certificate and the time required for sign/verify operations, thus directly impacting overall performance. Given that the signature algorithm is the product of negotiation between the client and server and a constrained session duration, the practicality of using more than one signature scheme can seem questionable. A server could potentially support multiple signature algorithms with separate certificates for each, and depending on the capabilities of the client, choose only one for the selected session. This solves the requirement for backwards compatibility without using more than one signature algorithm in the TLS handshake and minimizes the performance impact on the protocol; however, this also assumes that a client and server can come to an agreement on a single signature algorithm. A hybrid signature scheme could be used in instances where neither side completely agrees on a specific parameter without sacrificing security or trust. In this situation, both sides would eventually have to agree on *both* component algorithms; however, they would *not* need to agree on the preference of one or the other.

Another alternative approach to using a hybrid signature scheme *within* the TLS handshake is to use a single backwards compatible hybrid digital certificate for a server instead of multiple independent certificates. An application designer could use the backward compatible certificate to minimize the total number of certificates on the server. For example, a nested certificate structure like Figure 5.3 can be used to support two different signature algorithms with the inner signature algorithm being the one most likely to be supported by the most clients. Depending on the signature algorithm negotiation phase of the handshake, the server could choose to separate the inner certificate entirely from the outer container and only send the inner certificate. This eliminates the overhead of including both signatures; however, the second public key would still be included as part of the inner certificate.

THIS PAGE INTENTIONALLY LEFT BLANK

CHAPTER 6:
Experiments and Results for Hybrid Algorithms

This chapter details the design considerations, methodology, and results for testing the performance of hybrid signature schemes. All supporting documents can be found at: https://github.com/jllytle/hybrid-digital-signatures. The ultimate goal of this testing is to provide a comparative quantitative baseline for each considered hybrid signature scheme. Every hybrid scheme is evaluated based on the relative speed of their standalone cryptographic operations. The OQS library *liboqs* [87], an open source C library for PQ algorithms, was modified in order to support the true hybrid schemes introduced in Chapter 3. Table 6.1 depicts the hybrid signature algorithm identifiers used for testing along with the claimed NIST PQ security category. For distinction, the algorithm identifiers used in the true hybrid schemes are separated by a "_" while the algorithm identifiers used in the concatenated hybrid schemes are separated by a "+".

Table 6.1. Supported Algorithm Combinations for True and Concatenated Hybrid Schemes

Name	Notation	Type	NIST PQ Security
MQDSS-31-48_RSA3072	mq48_rsa3072	True Hybrid	Level 1
MQDSS-31-48 + RSA3072	mq48+rsa3072	Concatenated	Level 1
qTESLA-p-I_RSA3072	qt1_rsa3072	True Hybrid	Level 1
qTESLA-p-I + RSA3072	qt1+rsa3072	Concatenated	Level 1
Dilithium 2_RSA3072	di2_rsa3072	True Hybrid	Level 1
Dilithium 2 + RSA3072	di2+rsa3072	Concatenated	Level 1
Falcon-512_RSA3072	falcon512_rsa3072	True Hybrid	Level 1
Falcon-512 + RSA3072	falcon512+rsa3072	Concatenated	Level 1
MQDSS-31-48_DSA3072 #1	mq48_dsa3072_1	True Hybrid	Level 1
MQDSS-31-48_DSA3072 #2	mq48_dsa3072_2	True Hybrid	Level 1
MQDSS-31-48_DSA3072 #3	mq48_dsa3072_3	True Hybrid	Level 1

MQDSS-31-48 + DSA3072	mq48+dsa3072	Concatenated	Level 1
qTESLA-p-I_DSA3072 #1	qt1_dsa3072_1	True Hybrid	Level 1
qTESLA-p-I_DSA3072 #2	qt1_dsa3072_2	True Hybrid	Level 1
qTESLA-p-I_DSA3072 #3	qt1_dsa3072_3	True Hybrid	Level 1
qTESLA-p-I + DSA3072	qt1+dsa3072	Concatenated	Level 1
Dilithium 2_DSA3072 #1	di2_dsa3072_1	True Hybrid	Level 1
Dilithium 2_DSA3072 #2	di2_dsa3072_2	True Hybrid	Level 1
Dilithium 2_DSA3072 #3	di2_dsa3072_3	True Hybrid	Level 1
Dilithium 2 + DSA3072	di2+dsa3072	Concatenated	Level 1
MQDSS-31-48_ECDSA P-256 #1	mq48_P-256_1	True Hybrid	Level 1
MQDSS-31-48_ECDSA P-256 #2	mq48_p256_2	True Hybrid	Level 1
MQDSS-31-48_ECDSA P-256 #3	mq48_p256_3	True Hybrid	Level 1
MQDSS-31-48 + ECDSA P-256	mq48+p256	Concatenated	Level 1
qTESLA-p-I_ECDSA P-256 #1	qt1_p256_1	True Hybrid	Level 1
qTESLA-p-I_ECDSA P-256 #2	qt1_p256_2	True Hybrid	Level 1
qTESLA-p-I_ECDSA P-256 #3	qt1_p256_3	True Hybrid	Level 1
qTESLA-p-I + ECDSA P-256	qt1+p256	Concatenated	Level 1
Dilithium 2_ECDSA P-256 #1	di2_p256_1	True Hybrid	Level 1
Dilithium 2_ECDSA P-256 #2	di2_p256_2	True Hybrid	Level 1
Dilithium 2_ECDSA P-256 #3	di2_p256_3	True Hybrid	Level 1
Dilithium 2 + ECDSA P-256	di2+p256	Concatenated	Level 1
MQDSS-31-64_RSA3072	mq64_rsa3072	True Hybrid	Level 3
MQDSS-31-64 + RSA3072	mq64+rsa3072	Concatenated	Level 3
qTESLA-p-III_RSA3072	qt3_rsa3072	True Hybrid	Level 3
qTESLA-p-III + RSA3072	qt3+rsa3072	Concatenated	Level 3
Dilithium 3_RSA3072	di3_rsa3072	True Hybrid	Level 3
Dilithium 3 + RSA3072	di3+rsa3072	Concatenated	Level 3
Falcon-1024_RSA3072	falcon1024_rsa3072	True Hybrid	Level 3
Falcon-1024 + RSA3072	falcon1024+rsa3072	Concatenated	Level 3
MQDSS-31-64_DSA3072 #1	mq64_dsa3072_1	True Hybrid	Level 3
MQDSS-31-64_DSA3072 #2	mq64_dsa3072_2	Concatenated	Level 3
MQDSS-31-64 + DSA3072	mq64+dsa3072	True Hybrid	Level 3

qTESLA-p-III_DSA3072 #1	qt3_dsa3072_1	True Hybrid	Level 3
qTESLA-p-III_DSA3072 #2	qt3_dsa3072_2	True Hybrid	Level 3
qTESLA-p-III_DSA3072 #3	qt3_dsa3072_3	True Hybrid	Level 3
qTESLA-p-III + DSA3072	qt3+dsa3072	Concatenated	Level 3
Dilithium 3_DSA3072 #1	di3_dsa3072_1	True Hybrid	Level 3
Dilithium 3_DSA3072 #2	di3_dsa3072_2	True Hybrid	Level 3
Dilithium 3_DSA3072 #3	di3_dsa3072_3	True Hybrid	Level 3
Dilithium 3 + DSA3072	di3+dsa3072	Concatenated	Level 3
MQDSS-31-64_ECDSA P-384 #1	mq64_p384_1	True Hybrid	Level 3
MQDSS-31-64_ECDSA P-384 #2	mq64_p384_2	True Hybrid	Level 3
MQDSS-31-64_ECDSA P-384 #3	mq64_p384_3	True Hybrid	Level 3
MQDSS-31-64 + ECDSA P-384	mq64+p384	Concatenated	Level 3
qTESLA-p-III_ECDSA P-384 #1	qt3_p384_1	True Hybrid	Level 3
qTESLA-p-III_ECDSA P-384 #2	qt3_p384_2	True Hybrid	Level 3
qTESLA-p-III_ECDSA P-384 #3	qt3_p384_3	True Hybrid	Level 3
qTESLA-p-III + ECDSA P-384	qt3+p384	Concatenated	Level 3
Dilithium 3_ECDSA P-384 #1	di3_p384_1	True Hybrid	Level 3
Dilithium 3_ECDSA P-384 #2	di3_p384_2	True Hybrid	Level 3
Dilithium 3_ECDSA P-384 #3	di3_p384_3	True Hybrid	Level 3
Dilithium 3 + ECDSA P-384	di3+p384	Concatenated	Level 3
Dilithium 3_MQDSS-31-64	di3_mq64	True Hybrid	Level 3
Dilithium 3 + MQDSS-31-64	di3+mq64	Concatenated	Level 3
Dilithium 3_qTESLA-p-III	di3_qt3	True Hybrid	Level 3
Dilithium 3 + qTESLA-p-III	di3+qt3	Concatenated	Level 3
MQDSS-31-64_qTESLA-p-III	mq64_qt3	True Hybrid	Level 3
MQDSS-31-64 + qTESLA-p-III	mq64+qt3	Concatenated	Level 3

As discussed in Section 3.3, the primary implementation goal is to minimize the impact hybridization has on the original functionality of the component algorithms. The OpenSSL library (version 1.1.1i) [101] is used for both classical signature algorithms and ASN.1 encoding support. To avoid name space confusion during integration and for clarity, the modified liboqs code used for all of the PQ algorithms is referenced as *libhds* both in

this paper and in the source code. While one of the primary contributions of this paper is implementing the true hybrid signature schemes, the critical work performed by the OQS team in maintaining an open-source framework for the NIST PQ submissions proved invaluable during implementation and testing.

As listed in Table 6.2, all experiments are conducted on an Ubuntu 20.04.1 Long Term Support (LTS) host equipped with an Intel Xeon Gold 6140 processor clocked at 2.40GHz with 4 GBs of ECC memory. *CMake* (v. 3.16.3) and *ninja* (v. 1.10.0) are used to respectively generate and build the *libhds* library for an x86-64 architecture. Similarly, *make* (v 4.2.1) is used to build OpenSSL . For compilation, all source code is compiled using gcc (v. 9.3.0).

Table 6.2. Testing Environment

Processor	Intel Xeon Gold 6140
Frequency	2.30GHz
Microarchitecture	Skylake
Memory	4GB ECC

6.1 Methodology

The experimentation in this chapter is focused on comparing the performance between different hybrid signature schemes by measuring the Central Processing Unit (CPU) usage of each scheme's Sign and Verify operations. CPU performance is captured by our *benchmark* utility. This program profiles each Sign and Verify operation by directly accessing a per-core timestamp register available on modern Intel x86/x64 processors. This register (i.e. Timestamp Counter (TSC)) tracks every cycle that occurs on a core and can be easily accessed with minimal overhead via the Read Time-Stamp Counter (RDTSC) and Read Time-Stamp Counter and Processor ID (RDTSCP) assembly instructions. The benchmark utility uses these opcodes via inline assembly to count the number of CPU clock cycles expended during a single operation. The total number of clock cycles is collected over a set number of iterations controlled by a simple `while` loop. In addition to tracking the cumulative clock cycle count, the program calculates the minimum, mean, and standard deviation of the entire set.

As with any benchmarking method, the accuracy of the TSC can be impacted by external factors. For example, the capabilities of the TSC can vary depending on hardware implementation, and other CPU features such as out-of-order execution can affect the results. To minimize these impacts, we confirmed our testing environment supports invariant TSC via observed CPU flags and low-level testing [102]. We also incorporated the benchmarking methodologies outlined by the Intel Corporation [103] to include using the RDTSCP instruction to reduce overhead and minimize out-of-order execution. Additionally, hyper-threading, variable overclocking, and background services are disabled to decrease variances caused by normal system operations, as recommended by the VAMPIRE lab [104].

Using the benchmark utility, we establish a quantitative baseline for each individual signature algorithm as a control group and then conduct the same tests on the true hybrid schemes introduced in Chapter 3. Clock cycles are chosen as our computational metric because they remain consistent regardless of the frequency of the processor (assuming identical system architecture). Each combination is accessed via the same Application Programming Interface (API) throughout testing to ensure all results are comparable.

6.2 Standalone Cryptographic Operations

This section provides the standalone CPU benchmark results for the Sign and Verify operations of each signature scheme and analyzes the differences between them. The goal is not to compare the performance of one component signature algorithm against another, but to compare and contrast the performance of different hybrid schemes. For clarity, only algorithms with comparable NIST security levels are used as components within the hybrid schemes to increase the readability of the results and to align to the projected use of hybrid schemes in a realistic environment. The only exceptions to this are the RSA and DSA algorithms which use 3072-bit keys throughout the experimentation. This limitation is arbitrary as it is possible to mix the security levels of the component algorithms.

Additionally, all signature algorithms are compiled without any CPU optimizations to include instruction set extensions (e.g., AVX, BMI, etc.). This decision does impact the computational performance of some of the PQ component algorithms; however, the goal of the testing is to measure the performance of the hybrid schemes and not the performance of the individual algorithms.

6.2.1 Setup

The Sign and Verify operations for each signature algorithm is measured using the *hybrid speed* program. This program repeatedly measures the expended clock cycles for both the Sign and Verify operations of each algorithm using inline assembly code to directly access the per-core timestamp register. Both operations are repeated for a total of 100,000 iterations for each test run. The minimum, mean, and standard deviation for each run is then calculated for each signature scheme.

To increase accuracy, the same key pair and a unique randomized 50-character message string is used for each operation. It is critical to generate a new randomized message string for each Sign operation to counteract differences within the results between probabilistic and deterministic signature algorithms. Specifically, the Dilithium signature algorithm that was submitted during Round 1 of NIST's PQC standardization project is deterministic [8], [72]. During signing, the ExpandMask function "deterministically generates the randomness of the Dilithium signature scheme" by mapping a seed and nonce to the sampled vectors [72]. The seed is derived from elements of the secret key and the message that is being signed. This results not only in the same signature being generated for a message which uses the same key pair, but also results in the same number of rejection loops. If the same message is used for each test iteration, the results show minimal deviation in computational performance during signing and do not capture realistic scenarios where the content of messages vary.

For Round 2, the Dilithium team submitted a conceptual design change to the ExpandMask function that allows the seed to be chosen at random; however, this change was not yet integrated into the *liboqs* library at the time of testing.

6.2.2 Results

The first two subsections establish a classical and PQ performance baseline by introducing the individual performance results for each component signature algorithm. The results for each hybrid scheme are then presented by categories that align to each true hybrid scheme (e.g., FS–RSA, FS–DSA, etc.). Each category is further divided into separate tables by the respective NIST security level to improve readability.

Classical Baseline

Table 6.3 shows the individual performance of each classical algorithm used for testing. This data is used as a baseline from which to compare different hybrid combinations. The results from the classical algorithms are derived using low-level function calls in the OpenSSL library. Normally, these function calls should not be accessed directly as they bypass encapsulation and object orientation and expose critical elements of the individual algorithms and structures to manipulation. The direct access is necessary to integrate the newer hash algorithms (e.g., SHAKE128, SHAKE256, etc.) into the classical signature algorithms. This is only necessary because the current version of the library does not yet support using SHA-3 hash algorithms with legacy signature algorithms. Currently, there is a proposed RFC draft that updates the CMS to include support for the SHAKE family of hash algorithms [105]. As such, all RSA messages are manually encoded using the PKCS #1 standard defined in RFC 8017 [64] with custom algorithm identifiers for the newer hash algorithms.

Additionally, the RSA and DSA keys are limited to a length of 3072 bits which is roughly equivalent to the NIST Level 1 security category at 128 bits of security [41], [106]. In order for RSA to be comparable to NIST Level 3 at 192 bits of security, the modulus would need to be at least 7680 bits [106]. This key length is not tested, given the substantial increase in key size and the availability of classical and PQ alternatives.

The NIST P-256 elliptic curve (i.e., *prime256v1*) is used in all "Level 1" ECDSA testing to approximately match the same 128-bit security level achieved by the 3072-bit RSA and DSA keys [41], [69]. The NIST P-384 elliptic curve (i.e., *secp384r1*) is used for all "Level 3" ECDSA testing to approximately match the 192 bits of security provided by the Level 3 PQ algorithms. These are not the only elliptic curves that meets this requirement; however, P-256 and P-384 are widely supported due to their inclusion as one of two 256-bit or 384-bit curves approved for use with USG systems [24].

PQ Baseline

Tables 6.4 and 6.5 introduce the individual performance for each PQ signature algorithm used in further testing. As discussed in Chapter 4, these signature algorithms are chosen for their compatibility with the true hybrid schemes introduced in Chapter 3. For readability, the results are divided into two separate tables based on their claimed NIST security

Table 6.3. Classic Signature Algorithm Performance (clock cycles)

Name	Sign			Verify		
	Minimum	Mean	St. Dev.	Minimum	Mean	St. Dev.
RSA 3072	4775952	4878751	428825	101484	103053	3008
DSA 3072	1908506	2052518	43366	1836340	1907853	33743
ECDSA P-256	146478	148717	3997	321660	323800	4774

level (e.g., Level 1, Level 3) and this pattern is repeated for the duration of the paper. These levels were introduced during the NIST PQ standardization project and are based on broad comparison to a "reference primitive offered by the existing NIST standards in symmetric cryptography" [8]. Levels 1, 3, and 5 establish that an attack on a PQ algorithm's security definition must require "computational resources comparable to or greater than those required for key search on a block cipher" with a either a 128-bit, 192-bit, or 256-bit key, respectively. The majority of FS-compatible PQ signature algorithms submitted during the first two rounds support both levels 1 and 3; however, of these, only Dilithium established Level 5 security parameters. As a result, only the security parameters associated with Level 1 and Level 3 are explored in this work.

Table 6.4. Level 1 PQ Signature Algorithm Performance (clock cycles)

Name	Sign			Verify		
	Minimum	Mean	St. Dev.	Minimum	Mean	St. Dev.
MQDSS-31-48	28332168	28475354	154445	20433094	20508699	79665
qTESLA-p-I	769650	6080871	5656566	834702	841373	10674
Dilithium 2	202018	205766	6538	95926	97429	5739
Falcon-512	13883670	14002816	125397	141272	146377	2618

FS–RSA

Tables 6.6, 6.7, and 6.8 list the results of the Sign and Verify operations for the FS–RSA and Falcon–RSA hybrid schemes. The computational difference between the concatenated and true hybrid schemes is minimized by the reduced interaction between the component

Table 6.5. Level 3 PQ Signature Algorithm Performance (clock cycles)

Name	Sign			Verify		
	Minimum	Mean	St. Dev.	Minimum	Mean	St. Dev.
MQDSS-31-64	91877486	92139638	126269	66968172	67396942	741166
qTESLA-p-III	1817536	12271230	11790021	2191800	2203284	11129
Dilithium 3	405644	414253	16545	132936	138787	7141
Falcon-1024	29944920	30396434	157724	292120	293975	3903

algorithms. Unlike other true hybrid schemes (e.g., FS–FS, FS–DSA #1), the entirety of the PQ signature algorithm can be completed with only minor modification prior to initiating signing with the RSA algorithm. This is an expected result as the only difference between the concatenated and true hybrid scheme is that the RSA algorithm must sign a hash of both the message digest and the challenge c in the latter instead of just the message digest as in the former. For the same reason, the true hybrid scheme also produces a smaller signature because c is included in the output of the RSA algorithm. The size difference varies between component algorithms depending on the length of c after the algorithm packs and encodes its individual signature.

FS–DSA

Tables 6.9 and 6.10 depict the performance results for the FS–DSA hybrid schemes and Tables 6.11 and 6.12 depict the performance results for the FS-ECDSA hybrid schemes. The three FS–(EC)DSA true hybrid schemes, FS–(EC)DSA #1, FS–(EC)DSA #2, and FS–(EC)DSA #3 are differentiated by appending either a "_1", "_2" or "_3", respectively.

The results show a significant computational increase during signing for the FS–DSA #1 true hybrid scheme when the component algorithm is either qTESLA or Dilithium. While the minimum required clock cycles for the concatenated and hybrid schemes are negligible (approx. 1-2% difference) across all combinations, the average Sign clock cycles required for qTESLA-p-I_DSA3072_1 and qTESLA-p-III_DSA3072_1 are roughly four times greater than the concatenated equivalent. Similarly, the average Sign clock cycles for Dilithium 2_DSA3072_1 and Dilithium 3_DSA3072_1 are approximately 5.6 times greater than their concatenated equivalents.

Table 6.6. Level 1 FS–RSA Algorithm Performance (clock cycles/100,000 iterations)

Name	Sign			Verify		
	Minimum	Mean	St. Dev.	Minimum	Mean	St. Dev.
MQDSS-31-48_RSA3072	33174256	33426406	431071	20306580	20387172	278981
MQDSS-31-48+RSA3072	33115092	33404479	504067	20319128	20432653	185177
qTESLA-p-I_RSA3072	5524304	7761219	2181460	935986	957015	29502
qTESLA-p-I+RSA3072	5522052	7824139	2398040	937408	961961	33513
Dilithium 2_RSA3072	5081682	5946696	825240	308518	315259	12649
Dilithium 2+RSA3072	5089776	5867469	760034	308564	316811	15755

Table 6.7. Level 3 FS–RSA Algorithm Performance (clock cycles/100,000 iterations)

Name	Sign			Verify		
	Minimum	Mean	St. Dev.	Minimum	Mean	St. Dev.
MQDSS-31-64_RSA3072	96870754	97456064	550850	68288436	68569869	435007
MQDSS-31-64+RSA3072	95567940	96232121	717129	70598242	71028114	449881
qTESLA-p-III_RSA3072	6738706	11058109	4416960	2275136	2307037	33229
qTESLA-p-III+RSA3072	6744158	11576033	5038620	2270330	2314710	42344
Dilithium 3_RSA3072	5253670	6263381	1057898	406032	410517	9310
Dilithium 3+RSA3072	5252904	6304159	1092058	405812	410523	9303

Table 6.8. Level 1 and 3 Falcon–RSA Algorithm Performance (clock cycles/100,000 iterations)

Name	Sign			Verify		
	Minimum	Mean	St. Dev.	Minimum	Mean	St. Dev.
Falcon-512_RSA3072	18717310	19074463	556886	248084	254144	15062
Falcon-512+RSA3072	18435926	18842572	477399	244488	248024	3690
Falcon-1024_RSA3072	35010284	35780684	548477	392828	400322	9798
Falcon-1024+RSA3072	34939216	35547779	493642	391530	395248	4924

The performance gap is caused by how either PQ algorithm creates a signature. Both of these signature algorithms share a similar programmatic flow in which they employ a rejection sampling to generate a random polynomial that is eventually used to generate a signature. The signature must meet certain criteria before it is accepted and, in the event that it is rejected, both algorithms restart the signing process with a new random polynomial. It is inside of this sampling loop that both the ω and challenge c values are derived in part from the random polynomial. As a result, the ω and c values change with every sampling iteration.

Due to the way the FS–DSA #1 scheme is designed, the DSA algorithm requires the ω as input and a portion of its output s is then used as input for c. As such, the entire DSA signing operation is repeated for every iteration of the sampling loop. This results in an exponential increase in computation for every additional iteration. Figure 6.1 captures this trend for the ECDSA #1 and DSA #1 hybrid combinations of Dilithium 2 and qTESLA-p-I. This problem is not present in the other versions of the FS–DSA hybrid schemes because the DSA portion of the algorithm occurs *after* the response z has been validated. In other words, the rejection loop does not include the DSA portion of the hybrid scheme. Instead, the only additional operations occurring within the loop are those used to generate the shared challenge c such as additional hash operations.

FS–FS

Tables 6.13 and 6.14 show the results of the FS–FS true hybrid schemes. As mentioned previously, algorithm combinations were limited to comparing similar NIST security levels

Table 6.9. Level 1 FS–DSA Algorithm Performance (clock cycles/100,000 iterations)

Name	Sign			Verify		
	Minimum	Mean	St. Dev.	Minimum	Mean	St. Dev.
MQDSS-31-48_DSA3072_1	30593510	30911629	190915	22296080	22447616	184779
MQDSS-31-48_DSA3072_2	30286158	30486752	112053	22609498	22735904	202410
MQDSS-31-48_DSA3072_3	28892260	29273585	482051	20782366	20960530	148475
MQDSS-31-48+DSA3072	30445364	30866511	253820	23105798	23336910	165772
qTESLA-p-I_DSA3072_1	2786674	21853655	19252583	2708872	2799670	104610
qTESLA-p-I_DSA3072_2	2783430	4969160	2168691	2725524	2820646	117599
qTESLA-p-I_DSA3072_3	4645166	6803139	2262872	2749568	2797934	33565
qTESLA-p-I+DSA3072	2863320	4929405	2144125	2731348	2832163	274384
Dilithium 2_DSA3072_1	2403506	13570161	11806304	2125466	2201727	198151
Dilithium 2_DSA3072_2	2402442	3230981	793480	2127060	2202829	46512
Dilithium 2_DSA3072_3	4255984	5003315	726336	2163704	2209893	40023
Dilithium 2+DSA3072	2474348	3117772	684389	2099760	2174546	57389

Table 6.10. Level 3 FS–DSA Algorithm Performance (clock cycles/100,000 iterations)

Name	Sign			Verify		
	Minimum	Mean	St. Dev.	Minimum	Mean	St. Dev.
MQDSS-31-64_DSA3072_1	93664324	94480827	731356	68856150	69375085	515562
MQDSS-31-64_DSA3072_2	93126278	93709010	459711	68154172	68566037	365851
MQDSS-31-64_DSA3072_3	86885316	87580475	401333	64790938	65285280	272373
MQDSS-31-64+DSA3072	93146278	94173886	752451	69973912	70939857	509620
qTESLA-p-III_DSA3072_1	3786028	21035678	18137620	4062740	4144590	62881
qTESLA-p-III_DSA3072_2	3797666	8768752	5568558	4031274	4126837	122603
qTESLA-p-III_DSA3072_3	5648058	10363582	4693998	4099288	4185566	91074
qTESLA-p-III+DSA3072	3777176	8394040	4694360	4055854	4138440	55649
Dilithium 3_DSA3072_1	2543280	15804210	13964595	2202754	2259741	47863
Dilithium 3_DSA3072_2	2532608	3626120	1122317	2218780	2276042	41174
Dilithium 3_DSA3072_3	4411104	5442930	1003520	2260910	2310402	42473
Dilithium 3+DSA3072	2501558	3440188	971525	2178644	2224332	37876

Table 6.11. Level 1 FS-ECDSA Algorithm Performance (clock cycles/100,000 iterations)

Name	Sign			Verify		
	Minimum	Mean	St. Dev.	Minimum	Mean	St. Dev.
MQDSS-31-48_P-256_1	29673944	30010283	239413	21589426	21875800	248057
MQDSS-31-48_P-256_2	29667024	29985660	248567	21911948	22125701	177884
MQDSS-31-48_P-256_3	27302484	27540771	246822	19923908	20092525	161472
MQDSS-31-48+P-256	29504468	29910481	272541	21815036	22015379	131894
qTESLA-p-I_P-256_1	1981182	14297937	12485330	1907740	1958294	52662
qTESLA-p-I_P-256_2	1977910	4155831	2371222	1893860	1942864	54994
qTESLA-p-I_P-256_3	2995506	5133583	2305432	1959238	1995222	41050
qTESLA-p-I+P-256	1979672	4069353	2226078	1889212	1934356	48721
Dilithium 2_P-256_1	1612978	8281891	7439924	1279940	1314887	37692
Dilithium 2_P-256_2	1613566	2360574	799932	1287770	1316937	33755
Dilithium 2_P-256_3	2627620	3346614	742938	1344868	1368064	24996
Dilithium 2+P-256	1584504	2230755	670711	1279476	1306615	33586

Table 6.12. Level 3 FS-ECDSA Algorithm Performance (clock cycles/100,000 iterations)

Name	Sign			Verify		
	Minimum	Mean	St. Dev.	Minimum	Mean	St. Dev.
MQDSS-31-64_P-384_1	93331220	94462822	618842	69668524	70346020	406472
MQDSS-31-64_P-384_2	92562708	93734508	521095	68811476	69637445	433858
MQDSS-31-64_P-384_3	87660710	88325308	465132	64962174	65420061	305014
MQDSS-31-64+P-384	92592892	93396275	496419	67445270	68229051	467418
qTESLA-p-III_P-384_1	2991960	14862288	12310802	3269906	3327888	72241
qTESLA-p-III_P-384_2	2985150	7869324	5205150	3237836	3310353	52980
qTESLA-p-III_P-384_3	6442510	10970763	4418290	4342570	4414645	61690
qTESLA-p-III+P-384	2988522	7788849	5286906	3242010	3299152	66254
Dilithium 3_P-384_1	1729576	9951745	8875587	1392140	1426187	34431
Dilithium 3_P-384_2	1734598	2835995	1208457	1387014	1426976	55507
Dilithium 3_P-384_3	5184894	6269566	1064462	2485112	2531574	43676
Dilithium 3+P-384	1706448	2631422	1025969	1349078	1368455	31114

for clarity. While it is possible to combine two PQ algorithms with dissimilar security levels, the practicality of doing so is unclear given the increased key and signature size of PQ signature algorithms, in general.

The same pattern of results found in the FS–DSA #1 hybrid scheme is also present in certain combinations of the FS–FS hybrid scheme. Specifically, combining both Dilithium and qTESLA results in another exponential increase in the number of CPU cycles needed for the Sign operation. On average, both Dilithium 2_qTESLA-p-I and Dilithium 3_qTESLA-p-III require over 82 percent more cycles than their concatenated counterparts.

This effect is again caused by the rejection sampling present in both qTESLA and Dilithium. Both signature algorithms require the signature derived from this value to meet strict criteria before it is accepted. For the FS–FS scheme, if either algorithm rejects the shared challenge c, both algorithms must restart at the beginning of their rejection sampling and the algorithm that rejected c increments their internal nonce. Essentially, a shared c value that satisfies *both* algorithm's selection criteria is required in order to complete the signing process. Table 6.15 and Table 6.16 list the number of iterations of each true and concatenated hybrid scheme for both Dilithium and qTESLA, respectively, by recording changes in the nonce values of both algorithms during testing. As shown, only the FS–FS hybrid scheme between Dilithium and qTESLSA results in a substantial increase in the number of rejection sampling iterations.

6.2.3 Discussion

The primary goal of testing the true hybrid schemes presented in Chapter 3 is to determine if they introduced significant computational overhead outside of the component algorithms. Figures 6.2–6.4 show the average difference between executing the Sign and Verify operations for each component algorithm consecutively or in the corresponding true hybrid scheme. Table 6.17 lists the percentage difference of both the Sign and Verify operations for each hybrid combination. When comparing the results, there is very little difference between the true hybrid schemes and the concatenated hybrid schemes with the exception of FS–DSA #1 and certain FS–FS combinations.

The similarity of the performance between the concatenated and hybrid schemes highlights that the true hybrid schemes do not add significant computational overhead in most combinations. This is explained by the fact that the Sign and Verify operations of the individual

Table 6.13. Level 1 FS–FS Algorithm Performance (clock cycles/100,000 iterations)

Name	Sign			Verify		
	Minimum	Mean	St. Dev.	Minimum	Mean	St. Dev.
MQDSS-31-48_qTESLA-p-I	25914630	28343915	2266728	19477260	19756312	228915
MQDSS-31-48+qTESLA-p-I	25766862	28300895	2359434	19263604	19549071	182367
Dilithium 2_qTESLA-p-I	1108922	20373856	20022781	1057124	1083495	45853
Dilithium 2+qTESLA-p-I	1084436	3600091	2147510	1048870	1079041	44559
Dilithium 2_MQDSS-31-48	25474576	26440020	806066	18541908	18763211	157209
Dilithium 2+MQDSS-31-48	25510494	26356854	682342	18541332	18729667	239650

Table 6.14. Level 3 FS–FS Algorithm Performance (clock cycles/100,000 iterations)

Name	Sign			Verify		
	Minimum	Mean	St. Dev.	Minimum	Mean	St. Dev.
MQDSS-31-64_qTESLA-p-III	85385952	90730989	4695612	61427524	62149326	490615
MQDSS-31-64+qTESLA-p-III	85717446	91018767	4913508	62951274	63813032	609661
Dilithium 3_qTESLA-p-III	2246790	44295900	41017452	2497288	2560986	85820
Dilithium 3+qTESLA-p-III	2231714	7691898	5031825	2477974	2519027	43158
Dilithium 3_MQDSS-31-64	84708782	87481815	2534362	64275024	64958704	414343
Dilithium 3+MQDSS-31-64	84322128	85787056	1039509	61285770	61707011	559371

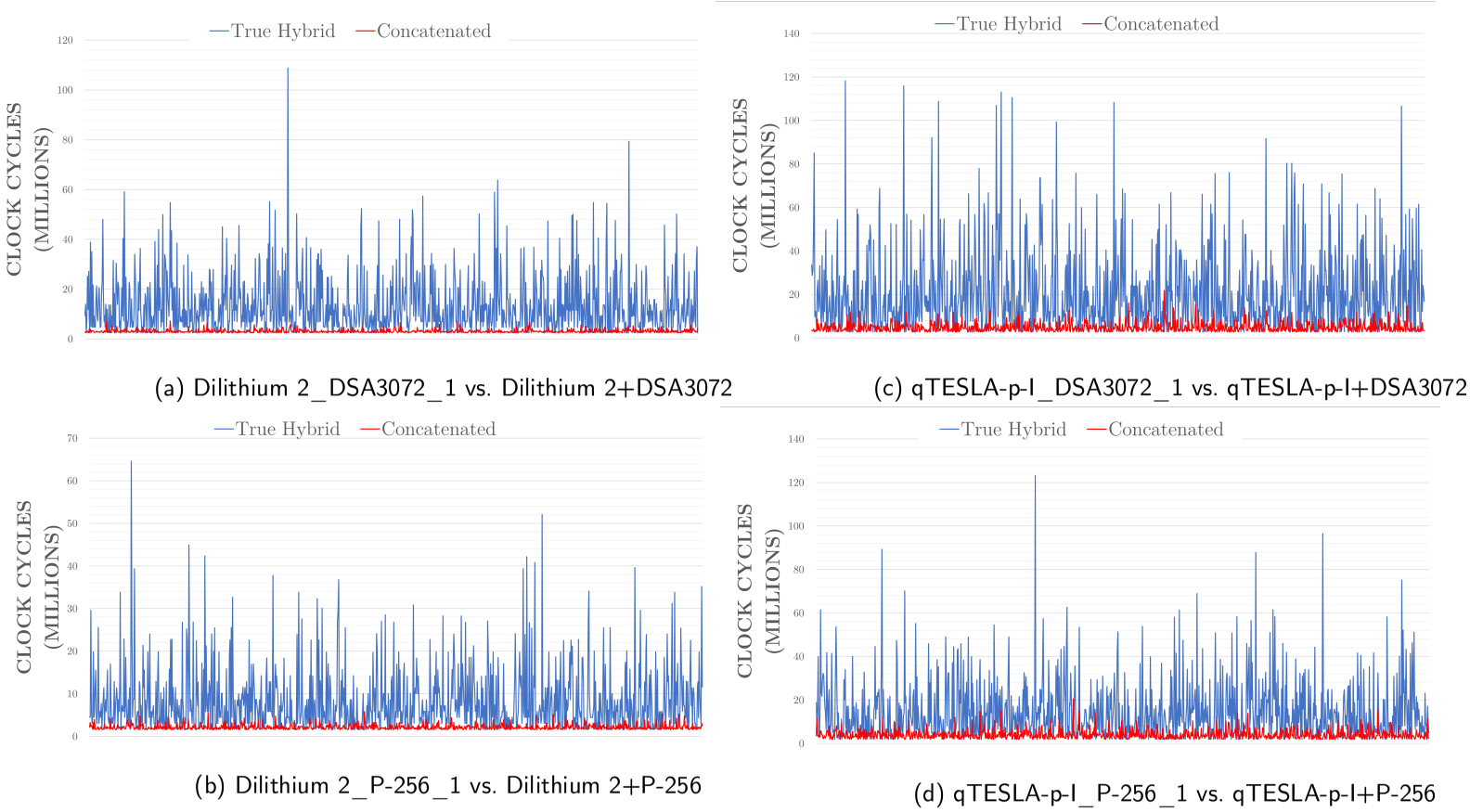


Figure 6.1. Performance Gap between Certain FS–(EC)DSA #1 Combinations and their Concatenated Counterparts

Table 6.15. Rejection Sampling Iterations for the Dilithium Signature Algorithm

Name	Min	Max	Mean	St. Dev.
Dilithium 2	3	111	16	14
Dilithium 2_RSA3072	3	132	17	15
Dilithium 2_DSA3072_1	3	123	16	15
Dilithium 2_DSA3072_2	3	102	16	15
Dilithium 2_DSA3072_3	3	120	17	14
Dilithium 2_P-256_1	3	111	17	15
Dilithium 2_P-256_2	3	108	17	15
Dilithium 2_P-256_3	3	123	17	15
Dilithium 2_MQDSS-31-48	3	120	16	14
Dilithium 2_qTESLA-p-I	3	**963**	153	148
Dilithium 2+qTESLA-p-I	3	108	17	15
Dilithium 3	4	180	26	24
Dilithium 3_RSA3072	4	152	25	23
Dilithium 3_DSA3072_1	4	176	26	24
Dilithium 3_DSA3072_2	4	156	26	23
Dilithium 3_DSA3072_3	4	144	26	24
Dilithium 3_P-384_1	4	236	27	26
Dilithium 3_P-384_2	4	172	26	23
Dilithium 3_P-384_3	4	160	25	23
Dilithium 3_MQDSS-31-64	4	176	26	24
Dilithium 3_qTESLA-p-III	4	**2416**	194	191
Dilithium 3+qTESLA-p-III	4	204	26	24

component algorithms are computationally more significant (i.e., "cost" more) than any of the additional steps introduced by the true hybrid schemes. Since both the concatenated and true hybrid schemes must complete the same Sign and Verify operations, the end result is a very similar performance in most cases. The only exception to this is when the hybrid scheme introduces computationally expensive code inside of either qTESLA's or Dilithium's rejection sampling during signing. This occurs in both the FS–DSA #1 and FS–FS hybrid schemes which results in a significant decrease to the average signing efficiency.

Table 6.16. Rejection Sampling Iterations for the qTESLA Signature Algorithm

Name	Min	Max	Mean	St. Dev.
qTESLA-p-I	1	88	9	8
qTESLA-p-I_RSA3072	1	85	8	8
qTESLA-p-I_DSA3072_1	1	99	8	8
qTESLA-p-I_DSA3072_2	1	81	8	8
qTESLA-p-I_DSA3072_3	1	55	8	8
qTESLA-p-I_P-256_1	1	38	7	8
qTESLA-p-I_P-256_2	1	89	7	8
qTESLA-p-III_P-256_2	1	64	9	8
MQDSS-31-48_qTESLA-p-I	1	74	9	8
Dilithium 2_qTESLA-p-I	1	**321**	51	49
Dilithium 2+qTESLA-p-I	1	64	9	8
qTESLA-p-III	1	59	7	7
qTESLA-p-III_RSA3072	1	75	7	7
qTESLA-p-III_DSA3072_1	1	76	7	7
qTESLA-p-III_DSA3072_2	1	75	7	6
qTESLA-p-III_DSA3072_3	1	43	7	7
qTESLA-p-III_P-384_1	1	25	6	5
qTESLA-p-III_P-384_2	1	69	6	7
qTESLA-p-III_P-384_3	1	50	7	6
MQDSS-31-64_qTESLA-p-III	1	41	7	6
Dilithium 3_qTESLA-p-III	1	**604**	48	47
Dilithium 3+qTESLA-p-III	1	51	7	6

Table 6.17. Percentage Difference of True Hybrid Minimum and Mean Performance from Concatenated Hybrid Schemes

Name	Sign		Verify	
	Minimum	Mean	Minimum	Mean
Dilithium 2_RSA3072	-0.16%	1.33%	-0.01%	-0.49%
Dilithium 2_DSA3072_1	-2.95%	77.02%	1.21%	1.23%
Dilithium 2_DSA3072_2	-2.99%	3.50%	1.28%	1.28%

Dilithium 2_DSA3072_3	41.86%	37.69%	2.96%	1.60%
Dilithium 2_P256_1	1.77%	73.06%	0.04%	0.63%
Dilithium 2_P256_2	1.80%	5.50%	0.64%	0.78%
Dilithium 2_P256_3	39.70%	33.34%	9.72%	4.86%
Dilithium 2_MQDSS-31-48	-0.14%	0.31%	0.00%	0.18%
Dilithium 2_qTESLA-p-I	2.21%	82.33%	0.78%	0.41%
qTESLA-p-I_RSA3072	0.04%	-0.81%	-0.15%	-0.52%
qTESLA-p-I_DSA3072_1	-2.75%	77.44%	-0.83%	-1.16%
qTESLA-p-I_DSA3072_2	-2.87%	0.80%	-0.21%	-0.41%
qTESLA-p-I_DSA3072_3	38.36%	27.54%	0.66%	-1.22%
qTESLA-p-I_P256_1	0.08%	71.54%	0.97%	1.22%
qTESLA-p-I_P256_2	-0.09%	2.08%	0.25%	0.44%
qTESLA-p-I_P256_3	33.91%	20.73%	3.57%	3.05%
MQDSS-31-48_RSA3072	0.44%	0.40%	-2.32%	-2.15%
MQDSS-31-48_DSA3072_1	0.06%	-0.06%	-1.27%	-1.30%
MQDSS-31-48_DSA3072_2	0.08%	0.12%	0.53%	0.36%
MQDSS-31-48_DSA3072_3	6.11%	6.47%	1.78%	1.63%
MQDSS-31-48_P256_1	-0.41%	-0.51%	2.28%	2.36%
MQDSS-31-48_P256_2	-0.11%	-0.24%	1.57%	1.46%
MQDSS-31-48_P256_3	3.52%	3.48%	1.52%	1.48%
MQDSS-31-48_qTESLA-p-I	0.57%	0.15%	1.10%	1.05%
Dilithium 3_RSA3072	0.01%	-0.65%	0.05%	0.00%
Dilithium 3_DSA3072_1	1.64%	78.23%	1.09%	1.57%
Dilithium 3_DSA3072_2	1.23%	5.13%	1.81%	2.27%
Dilithium 3_DSA3072_3	43.29%	36.80%	3.19%	3.64%
Dilithium 3_P384_1	1.34%	73.56%	3.09%	4.05%
Dilithium 3_P384_2	1.62%	7.21%	2.74%	4.10%
Dilithium 3_P384_3	67.09%	58.03%	45.71%	45.94%
Dilithium 3_MQDSS-31-64	0.46%	1.94%	4.65%	5.01%
Dilithium 3_qTESLA-p-III	0.67%	82.64%	0.77%	1.64%
qTESLA-p-III_RSA3072	-0.08%	-4.68%	0.21%	-0.33%
qTESLA-p-III_DSA3072_1	0.23%	60.10%	0.17%	0.15%

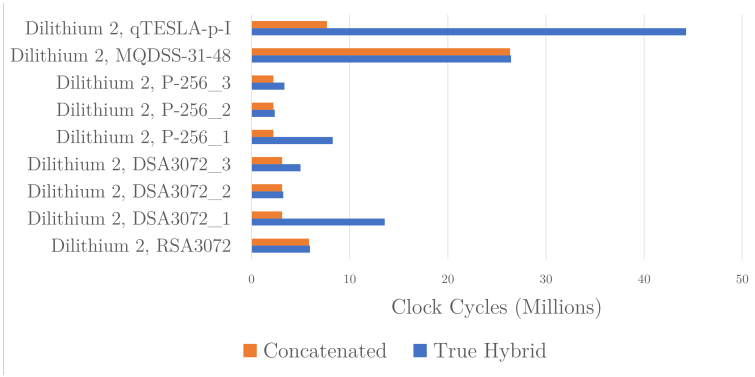
qTESLA-p-III_DSA3072_2	0.54%	4.27%	-0.61%	-0.28%
qTESLA-p-III_DSA3072_3	1.55%	1.75%	1.58%	1.66%
qTESLA-p-III_P384_1	0.11%	47.59%	0.85%	0.86%
qTESLA-p-III_P384_2	-0.11%	1.02%	-0.13%	0.34%
qTESLA-p-III_P384_3	1.07%	1.23%	0.91%	0.87%
MQDSS-31-64_RSA3072	0.12%	-0.02%	2.23%	2.03%
MQDSS-31-64_DSA3072_1	-0.03%	0.06%	-0.34%	-0.20%
MQDSS-31-64_DSA3072_2	-0.49%	-0.14%	-0.32%	-0.35%
MQDSS-31-64_DSA3072_3	1.55%	1.75%	1.58%	1.66%
MQDSS-31-64_P384_1	-0.19%	0.01%	0.59%	0.63%
MQDSS-31-64_P384_2	-0.36%	-0.20%	0.47%	0.39%
MQDSS-31-64_P384_3	1.07%	1.23%	0.91%	0.87%
MQDSS-31-64_qTESLA-p-III	-0.39%	-0.32%	-2.48%	-2.68%

Selection Priorities

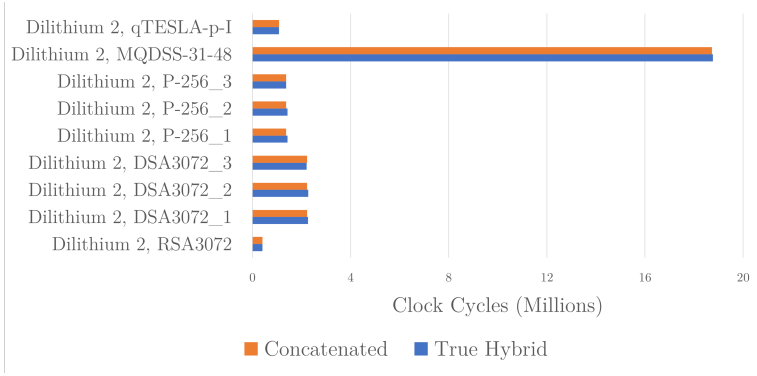
Based on the results in Table 6.17, the decision of choosing between a concatenated and true hybrid design should not be a question of performance, but one of security properties. A concatenated design only achieves the security properties of the component algorithms *if* both are verified. A signer has no guarantee that the verifier will choose to verify both signatures. In fact, most backwards-compatible systems rely on this concept as legacy systems may not support one or more of the component signature algorithms. Using a true hybrid scheme, it is impossible for an *honest* verifier to validate a signature without verifying all component algorithms (i.e., *hybrid verification*). As such, a signer achieves a guarantee that would not be possible in a purely concatenated system.

Code Design and Optimizations

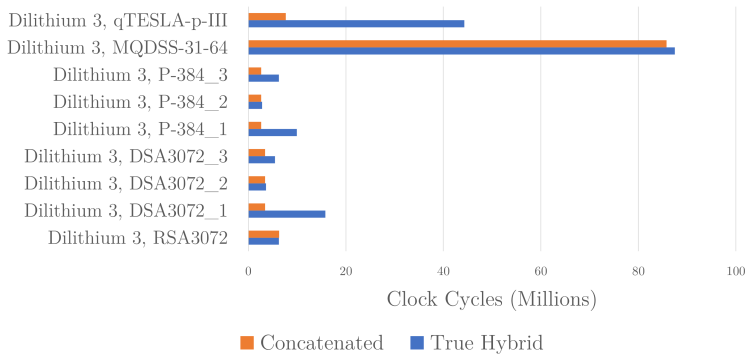
Our approach for implementation focuses on minimizing as much change to the original signature algorithms as possible. As a result, optimization of different hybrid combinations was not considered. To minimize the effect of additional function calls and other dissimilarities between combinations, we use the same API for all true and concatenated hybrid schemes. The API prioritizes the same depth of function calls and consistent parameters



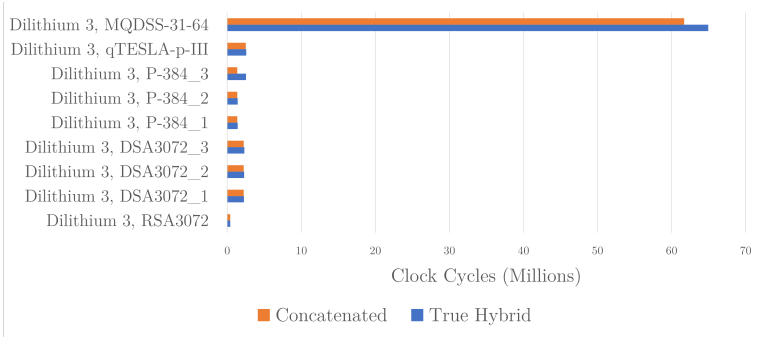
(a) Dilithium 2 Mean Sign Performance



(b) Dilithium 2 Mean Verify Performance

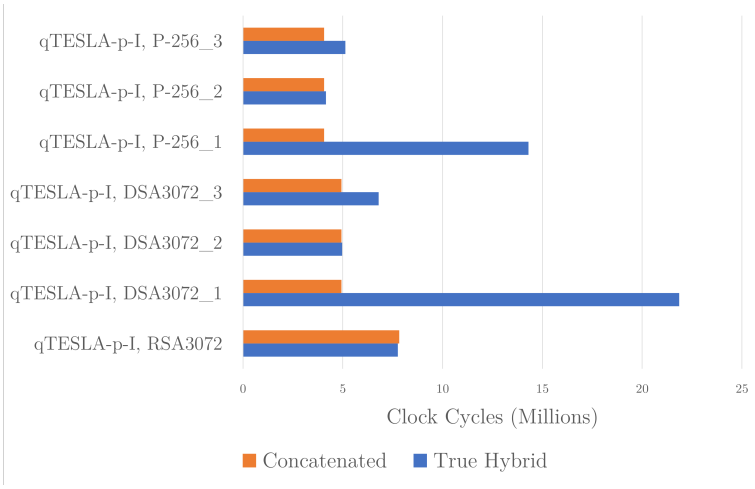


(c) Dilithium 3 Mean Sign Performance

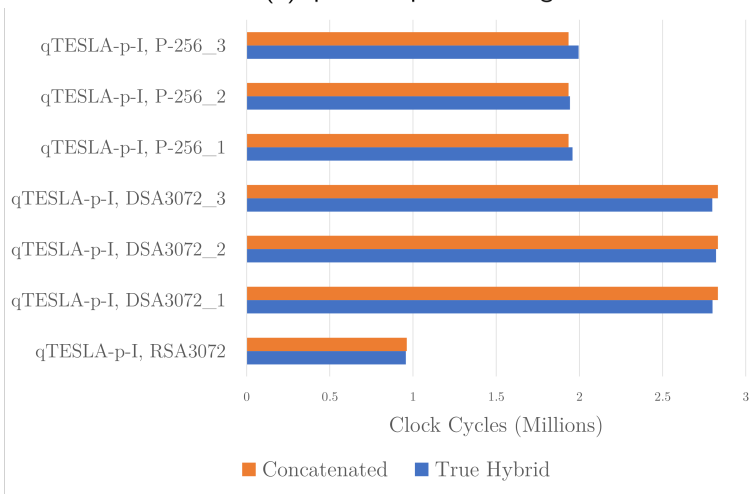


(d) Dilithium 3 Mean Verify Performance

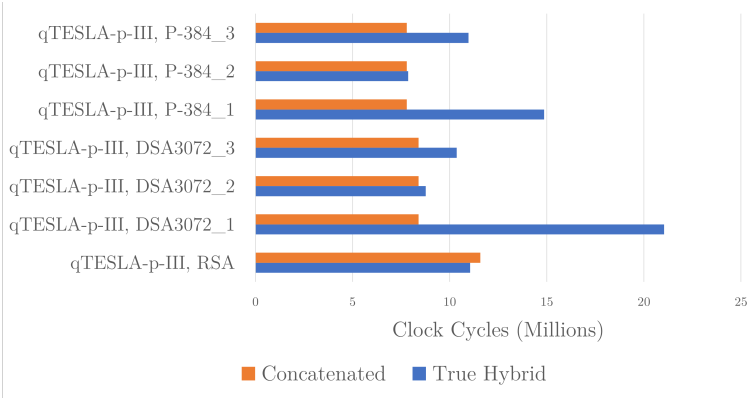
Figure 6.2. Dilithium 2 & 3 Mean Sign and Verify Operation Performance for True and Concatenated Hybrid Schemes



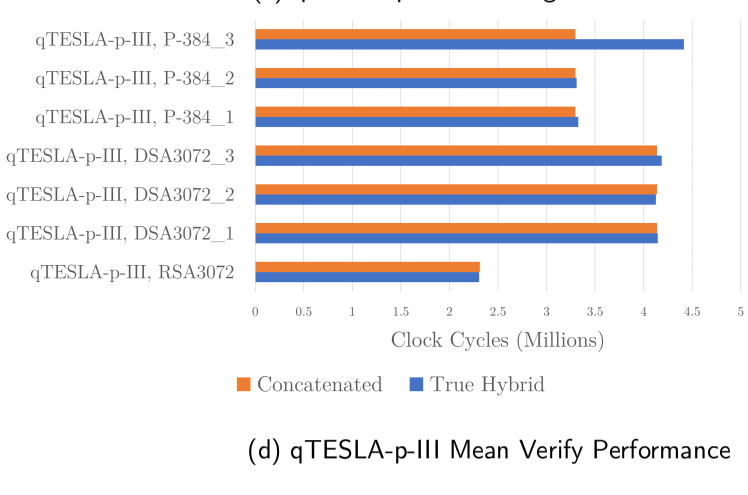
(a) qTESLA-p-I Mean Sign Performance



(b) qTESLA-p-I Mean Verify Performance

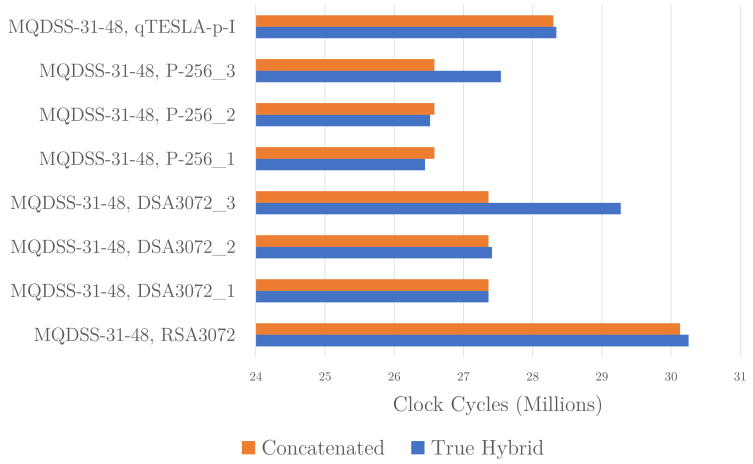


(c) qTESLA-p-III Mean Sign Performance

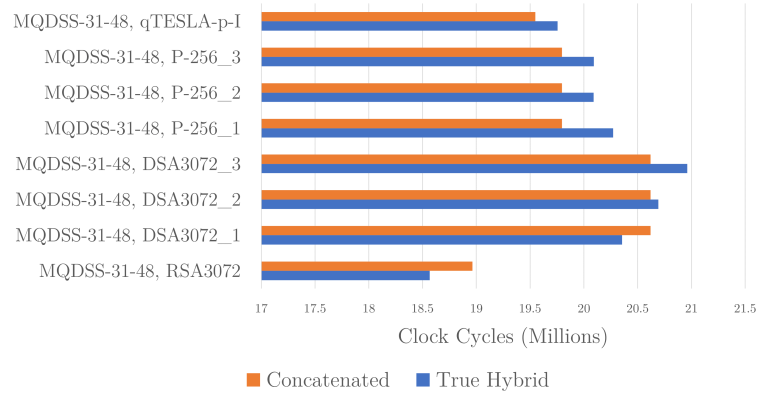


(d) qTESLA-p-III Mean Verify Performance

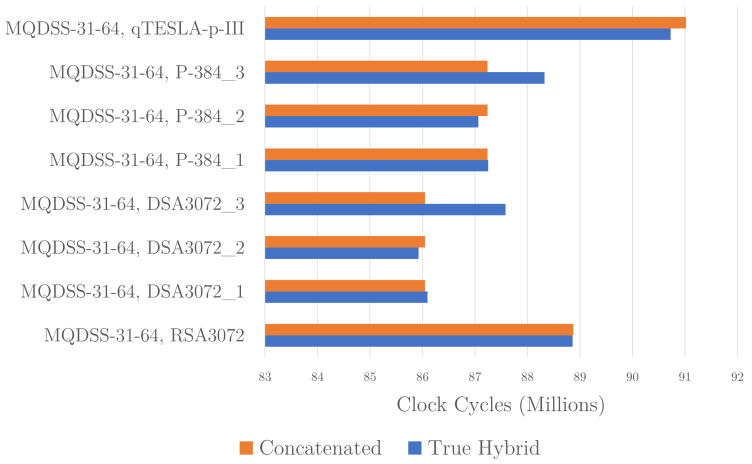
Figure 6.3. qTESLA-p-I & qTESLA-p-III Mean Sign and Verify Operation Performance for True and Concatenated Hybrid Schemes



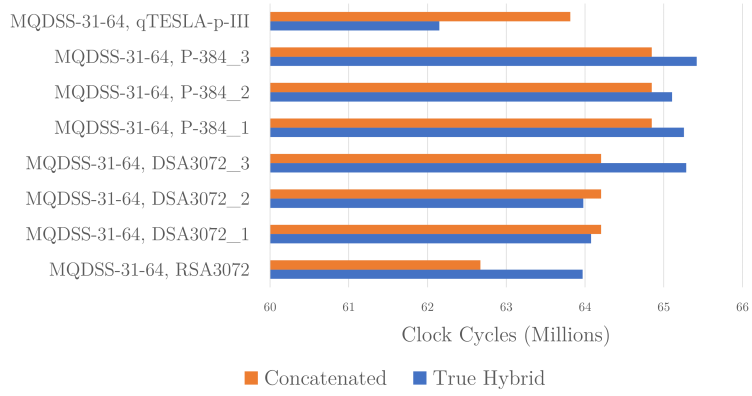
(a) MQDSS-31-48 Mean Sign Performance



(b) MQDSS-31-48 Mean Verify Performance



(c) MQDSS-31-64 Mean Sign Performance



(d) MQDSS-31-64 Mean Verify Performance

Figure 6.4. MQDSS-31-48 & MQDSS-31-64 Mean Sign and Verify Operation Performance for True and Concatenated Hybrid Schemes

over traditional clean coding principles for each true and concatenated hybrid combination. This approach is taken to minimize differences in the language and operating system specific overhead between schemes for our performance testing; however, this design decision leads to an explosion of duplicate code with subtle modifications as required by the hybrid scheme. This design artifact makes integration into existing cryptographic libraries both difficult and expensive. For example, any minor update to a component signature algorithm necessitates that the same change is applied across every hybrid combination without error.

Additionally, no efforts are made to protect against side-channel attacks or to safeguard critical areas of memory for any of the hybrid schemes tested in this work as this is considered outside the scope of this work. As a result, our implementations should not be considered safe for any production environment.

CHAPTER 7:
Results on X.509 Certificate Sampling and Hybrid Certificate Use in TLS

This chapter examines server-based X.509 certificates associated with a ranked list of active domain names and the performance of hybrid schemes in TLS 1.3 authentication. Through these experiments, this chapter provides a representative snapshot of the signature algorithms currently employed as part of PKI on the Internet and highlights the impact hybrid certificates have on the TLS 1.3 protocol.

7.1 X.509 Certificate Sampling

The section examines a collection of X.509 certificates recorded using a list of one million domains with the "most referring subnets" provided by Majestic-12 Ltd [107]. Majestic-12 Ltd. uses the total number of backlinks, a uni-directional HTML link to another website, contained within a web page to determine a relative score for each domain. This ranking system is not considered for the experiment; however, it is important to note that the results may be impacted.

7.1.1 Setup

In order to collect the X.509 certificates, a connection to Transmission Control Protocol (TCP) port 443 is attempted for every listed domain using OpenSSL's *s_client* program. If a TLS session is successfully established, all session metadata produced by the program is recorded. By default, this information includes the TLS version, selected ciphersuite, the entire certificate chain with Privacy Enhanced Mail (PEM) encoded certificates, and the total handshake size. The server certificate is then parsed and various attributes such as the certificate length, signature algorithm identifier, public key information, and any extensions are ingested into a local database from which the results are derived.

7.1.2 Results

Of the one million domains, only 683,289 responded with an active TLS session. This is an expected result as the ranked list does not specify or require the domains to support TLS on TCP 443. Table 7.1 provides a list of all signature algorithms encountered, based on the name associated with the *Signature Algorithm Identifier* field in the X.509 certificate. Table 7.2 uses the same sample but depicts the numbers based on the OID identified in the *Signature Algorithm* field in the X.509 certificate.

Table 7.1. Majestic-12 Breakout by Signature Algorithm Identifier

Signature Algorithm	Total	Percentage (%)
RSA	530972	77.70
ECDSA	152317	22.30
	683289	100

Table 7.2. Majestic-12 Breakout by Signature Algorithm OID

Signature OID Name	Total	Percentage (%)
md5WithRSAEncryption	1425	0.21
sha1WithRSAEncryption	9663	1.40
sha256WithRSAEncryption	517738	75.77
sha384WithRSAEncryption	1715	0.25
sha512WithRSAEncryption	427	0.06
sha1WithRSA *(deprecated)*	4	–
ecdsa-with-SHA256	152188	22.27
ecdsa-with-SHA1	2	–
ecdsa-with-SHA384	127	0.03
	683289	100

Using additional metadata recorded by the *s_client* command, we also extracted the TLS version and the key exchange/agreement protocols used during the handshake. Table 7.3 contains a breakdown of the TLS versions in our sample. Excluding failed connections or instances where certificates are expired, the majority of connections are divided between TLS 1.2 and TLS 1.3. For key exchanges in TLS versions up to 1.2, approximately 94

percent of the TLS sessions use Elliptic Curve Diffie-Hellman Ephemeral (ECDHE), 3.6 percent used RSA, and 2.6 percent use Diffie-Hellman Ephemeral (DHE) for exchanging symmetric keys.

Table 7.3. Majestic-12 Breakout by TLS version

TLS version	Total	Percentage (%)
TLSv1.0	464	0.99
TLSv1.1	9	0.02
TLSv1.2	27018	57.89
TLSv1.3	19184	41.10

For TLS 1.3, changes to the protocol removed support for legacy ciphersuites in favor of Authenticated Encryption with Associated Data (AEAD) algorithms [2]. Currently, NIST recommends four AEAD ciphersuites for use with TLS 1.3: TLS_AES_128_GCM_SHA256, TLS_AES_256_GCM_SHA384, TLS_AES_128_CCM_SHA256, and TLS_AES_128_CCM_8_SHA256 [108]. These suites can be paired with either RSA or ECDSA server certificates; however, DSA certificates are no longer supported. Of the collected samples, over 85 percent use TLS_AES_256_GCM_SHA384 as shown in Table 7.4.

Table 7.4. Majestic-12 Breakout of TLS 1.3 Ciphersuites

ciphersuite	Total	Percentage (%)
TLS_AES_256_GCM_SHA384	16374	85.35
TLS_AES_128_GCM_SHA256	1293	6.74
TLS_CHACHA20_POLY1305_SHA256	93	0.48
NONE (failed connections)	1424	7.42

7.1.3 Discussion

Based on the results in Table 7.1, the majority of X.509 certificates used to authenticate web servers in our data set still rely on the RSA signature algorithm. In the sample, no examples of the standard DSA algorithm were found in use; however, approximately 22 percent of the certificates used its successor, ECDSA.

Additionally, the average distance from the server to its root CA (e.g., depth of the certificate chain) in our sample is only two certificates. Based on the recorded metadata and the information contained within the X.509 certificates, the server certificate is typically signed by an intermediate CA which is itself signed by a root CA. The longest certificate chain in our data set contains 14 certificates; however, the server certificate is signed by an intermediate CA that is directly signed by a root CA. The additional certificates appear to be for related domains that are not specifically tied to the server certificate.

Signature Algorithm Transition

The fact that the majority of the certificates in our sample use the RSA signature algorithm reinforces the idea that digital signature schemes transition at a slower pace compared to other cryptographic primitives. Comparatively, the key exchange/agreement options in our sample show that the selected ciphersuite is likely not to use legacy algorithms like RSA and static Diffie–Hellman (DH) for exchanging keys. Table 7.5 provides a prioritized list of TLS ciphersuites for our version of the OpenSSL s_client program. This list is obtained by using a simple Python script that captures the TLS handshake metadata.

Table 7.5. List of TLS Ciphersuites for OpenSSL 1.1.1i *s_client* Program in Priority Order

Ciphersuite	Description
0x1302	TLS_AES_256_GCM_SHA384
0x1303	TLS_CHACHA20_POLY1305_SHA256
0x1301	TLS_AES_128_GCM_SHA256
0xc02c	TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384
0xc030	TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384
0x009f	TLS_DHE_RSA_WITH_AES_256_GCM_SHA384
0xcca9	TLS_ECDHE_ECDSA_WITH_CHACHA20_POLY1305_SHA256
0xcca8	TLS_ECDHE_RSA_WITH_CHACHA20_POLY1305_SHA256
0xccaa	TLS_DHE_RSA_WITH_CHACHA20_POLY1305_SHA256
0xc02b	TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256
0xc02f	TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256

0x009e	TLS_DHE_RSA_WITH_AES_128_GCM_SHA256
0xc024	TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA384
0xc028	TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA384
0x006b	TLS_DHE_RSA_WITH_AES_256_CBC_SHA256
0xc023	TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA256
0xc027	TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA256
0x0067	TLS_DHE_RSA_WITH_AES_128_CBC_SHA256
0xc00a	TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA
0xc014	TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA
0x0039	TLS_DHE_RSA_WITH_AES_256_CBC_SHA
0xc009	TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA
0xc013	TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA
0x0033	TLS_DHE_RSA_WITH_AES_128_CBC_SHA
0x009d	TLS_RSA_WITH_AES_256_GCM_SHA384
0x009c	TLS_RSA_WITH_AES_128_GCM_SHA256
0x003d	TLS_RSA_WITH_AES_256_CBC_SHA256
0x003c	TLS_RSA_WITH_AES_128_CBC_SHA256
0x0035	TLS_RSA_WITH_AES_256_CBC_SHA
0x002f	TLS_RSA_WITH_AES_128_CBC_SHA
0x00ff	TLS_EMPTY_RENEGOTIATION_INFO_SCSV

Table 7.5 shows that even when excluding TLS 1.3, which removes ciphersuite negotiation and mandates a small subset of ciphersuites, the *s_client* utility prefers "newer" key exchange algorithms. In fact, OpenSSL 1.1.1i lists over 21 different ciphersuites featuring either the ECDHE or DHE key exchange options before any that use the RSA algorithm. As a result, if a server running TLS 1.2 or older supports any of these ciphersuites, an application using the OpenSSL library for TLS support will not use RSA for key exchanges.

The prioritization of ciphersuites is a conscious decision made by standards organizations and system engineers to efficiently provide the highest level of security required for the situation at hand. While NIST does not provide a priority order of TLS ciphersuites for USG systems, it does limit ciphersuites to those that use "approved" algorithms [108]. Additionally, guidance is also updated if the risks in a design or implementation outweigh

perceived benefits. For example, NIST no longer recommends that RSA be used as a key transport mechanism and that any government agencies that rely on it move to a new method as soon as possible [108].

Public Key Length and Certificate Sizes

Upon examining the recorded X.509 certificates, the average size for a DER-encoded, server-based X.509 certificate is 1654 bytes. This analysis does not include any certificates beyond the server certificate that may be included as part of a certificate chain such as intermediate and root CA certificates. When categorized by signature algorithm, the average certificate size for RSA signed certificates is 1750 bytes and the average size for ECDSA is 1262 bytes.

Table 7.6 compares the relative size of the hybrid certificate designs discussed in Chapter 5. Each design contains the same metadata (e.g., issuer name, serial number, etc.) and is signed using Dilithium 2 and RSA 3072. The certificates are then encoded using the ASN.1 functions included with OpenSSL. Comparatively, there is little difference in size between the hybrid designs as they contain more or less the same information although at different locations within the certificate. The variance is a product of the ASN.1 structures used to represent the information. For example, the CROSSING and ISARA designs use the *SubjectPublicKeyInfo* structure for the second public key [20], [62]. This field is a sequence of the *AlgorithmIdentifier* and the public key represented as a bit string [90]. The OQS design, on the other hand, concatenates the second public key with the first public key and stores both in the original *SubjectPublicKeyInfo* field [87]. As a result, the second *AlgorithmIdentifier* is not needed.

Table 7.7 displays the relative sizes of the hybrid X.509 certificates between each true hybrid scheme. Each certificate uses the OQS design and contains the same metadata. Of note, there is approximately a 200-300 byte difference in certificate sizes between similar certificates in our X.509 sample and the certificates used in our TLS authentication testing (see Section 7.2) when controlling for the signature algorithm and public key length. This is likely due to a difference in the number of X.509v3 extensions between both groups. For example, our testing certificates do not contain any Online Certificate Status Protocol (OCSP), Certificate Revocation List (CRL), or Signed Certificate Timestamp (SCT) extensions whereas over seventy percent of the certificates in our data set do. Additionally, other extensions, found in both groups, are variable in length and are influenced by external factors such as domain name length.

Even with this taken into account, our smallest hybrid certificate, Dilithium 2_RSA3072, is still over twice as large as the average RSA 3072 certificate in our sample. This is not surprising given the relative increase of both public key and signature sizes between PQ and classic signature algorithms. The same trend extends to the difference between the true hybrid schemes and PQ signature algorithms; however, the difference in certificate size is not as stark. As shown in Table 7.7, there is approximately a 700 byte increase between Dilithium 2 and Dilithium 2_RSA3072 and only a 600 byte increase between Dilithium 2 and Dilithium 2_P-256. Regardless of the encoding format, hybrid digital certificates are obviously significantly larger than their single algorithm counterparts because hybrid schemes require at least two public keys. Additionally, the combined signature length, while varied, will also be larger.

Table 7.6. Comparison of Hybrid Certificate Sizes (bytes)

Name	Size (DER)	Size (PEM)
ISARA	4833	6603
CROSSING	4835	6603
OQS	4786	6538

While outside the scope of this paper, this experiment also highlighted several non-standard RSA key lengths in use. NIST recommends a minimum of 2048 bits for digital signature keys [109]; however, several keys were below this threshold. Other key sizes (e.g., 2024, 2255, 4086) were between standardized sizes. In these instances, the server certificates were either self-signed or part of default web server installations.

7.2 Authentication in TLS

This section explores the impact hybrid signature schemes have on the TLS 1.3 protocol handshake. The objective is to demonstrate how the increased certificate sizes and computational complexity caused by hybrid signature schemes can affect current standardized communication protocols. TLS 1.3 is chosen due to its ubiquitous use throughout Internet-based applications and existing research. To control the number of variables, all tests only observe server authentication using a digital certificate signed by an intermediate CA. OCSP responses, client certificate authentication, 0-RTT handshakes, and SCT extensions are considered beyond the scope of testing for this paper.

7.2.1 Setup

The experiment is designed to test the performance impact of hybrid digital certificates on a TLS 1.3 handshake by capturing timing variances between different hybrid signature schemes. As noted in Section 7.1.3, the average depth of a certificate chain is only two certificates: the intermediate CA and root CA. Based on this, our experiment examines two scenarios: a hybrid server certificate signed by an all-hybrid CA chain and a non-hybrid server certificate signed by a non-hybrid intermediate CA that is signed by a hybrid root CA. These scenarios represent the two extremes that are likely to be encountered in a realistic environment without substantial modifications to either the underlying protocol or PKI. For our control, we measure the performance of certificate chains composed entirely of the component algorithms for each scheme. For example, the control for the Dilithium 2_RSA3072 scheme would be one test where everything is signed by Dilithium 2 and one test where everything is signed by RSA3072. The only exception to this is the DSA algorithm. While we test all true hybrid combinations that use DSA as a component algorithm, OpenSSL no longer supports DSA certificates with TLS 1.3 [101]. In their guidelines, NIST also only mandates that TLS servers support RSA or ECDSA; however, DSA is included

"for completeness and to cover edge cases" [108]. As a result, we do not use DSA-only certificates in any of our tests.

For this experiment, all hybrid X.509 certificates are based on the OQS design presented in Chapter 5. This decision is based on the relative similarities in size of each hybrid certificate design (see Section 7.1.3) and the difficulty of integrating hybrid signatures into the OpenSSL library. Additionally, this experiment only examines the performance of the "Level 1" true hybrid schemes.

On the client side, OpenSSL's *s_time* program is modified to capture the elapsed time of each TLS handshake. By default, *s_time* creates a TLS session by establishing multiple, sequential TLS connections to a server over a set period provided by the user. The total number of connections and the average time spent for one connection is then provided as output. For our purposes, *s_time* is modified to measure and record the elapsed monotonic time in nanoseconds of each TLS handshake from the client's perspective using the *clock_gettime()* system call. We use the "CLOCK_MONOTONIC_RAW" clock as our time source because it is not subject to incremental changes like Network Time Protocol (NTP) adjustments that may affect accuracy when measuring the elapsed wall clock time [110]. We also modified the program to operate with a fixed number of iterations (i.e., 100,000) instead of using an interval of time.

On the server side, we use the default *s_server* program built from the same modified OpenSSL library. This program is designed as a generic TLS server and is used for all incoming connections created by the *s_time* program. The server uses a digital certificate directly signed by an intermediate CA.

All connections occurred over the default virtual interface on the host machine. This decision is made to minimize latency and other network-related variables during testing. Additionally, the modified *s_time* program does not request any content from the web server. Consequently, the results of this test represent a best case scenario for a generic TLS handshake and is not intended to reflect real-world or application-specific conditions.

7.2.2 Results

Table 7.8 provides the minimum, maximum, and mean handshake completion times in nanoseconds for the single signature algorithm control. These results are used to examine their performance of the true hybrid counterparts and should not be used to compare the performance between individual algorithms.

Table 7.8. Single Algorithm TLS 1.3 Performance (μs/100,000 iterations)

Name	Non-Hybrid Chain			
	Min.	Max.	Mean	St. Dev.
RSA3072	2665	6078	2763	170
ECDSA P-256	1010	3049	1052	28
Dilithium 2	1124	3532	1376	246
qTESLA-p-I	2272	8764	3045	804
MQDSS-31-48	30461	41629	30981	397

Table 7.9 provides the minimum, maximum, and mean handshake completion times in nanoseconds for an all-hybrid CA certificate chain and partially hybrid certificate chains using the specified signature algorithms. As described in Section 7.2.1, the experiment scenarios are separated as follows: in the all-hybrid certificate chain every certificate is hybrid while in the partially hybrid certificate chain only the root CA certificate is hybrid. In the second scenario, all other certificates in the chain (i.e., intermediate CA and server) are non-hybrid. Since the true hybrid schemes use two component algorithms, we separately examine the performance when each component type is used for the non-hybrid certificates. Using the Dilithium 2_RSA3072 combination as an example, we examine both a partially hybrid certificate chain where the non-hybrid certificates are signed with Dilithium 2 and a partially hybrid certificate chain where the non-hybrid certificates are signed with RSA3072. In both cases, the root CA is signed with the true hybrid scheme: Dilithium 2_RSA3072.

The performance results of all three certificate chains is presented in Table 7.9. The results for the all-hybrid certificate chain are under the *Hybrid Chain* column. The results for the partially hybrid chain where the non-hybrid certificates are signed with the first component

algorithm in the true hybrid scheme (e.g., Dilithium 2 in Dilithium 2_RSA3072) are under the *Hybrid Root CA (primary)* column. Finally, the results for partially hybrid certificate chain where the non-hybrid certificates are signed using the second component algorithm (e.g., RSA3072 in Dilithium 2_RSA3072) are under the *Hybrid Root CA (secondary)* column.

As expected, there is a significant difference in the time required to complete the TLS handshake between a completely hybrid certificate chain and a completely non-hybrid certificate chain. Table 7.10 shows the percentage difference between these two scenarios by comparing the all-hybrid chain to a completely non-hybrid chain using only one signature algorithm. While the difference varies depending on the algorithm, a handshake that uses a completely hybrid chain takes significantly longer due to both the size of the hybrid certificate and the increase in the time needed to complete the hybrid scheme's Sign and Verify operations.

Table 7.11 shows the percentage difference in the time required to complete the TLS handshake between a certificate chain where only the root CA uses a true hybrid scheme and a completely non-hybrid certificate chain. Of the combinations tested, only Dilithium 2_RSA3072 yielded results similar to the non-hybrid alternative (e.g., either Dilithium 2 *or* RSA3072) when only the root CA uses a hybrid scheme. The other true hybrid combinations show that there is a significant overhead associated with using hybrid schemes when compared with their solo counterparts. The only exceptions are when the performance of one component algorithm is significantly worse than the second component algorithm. This can be seen with the MQDSS-31-48_RSA3072 combination where there is little gap between the completely hybrid certificate chain and a partially hybrid certificate chain that uses the MQDSS-31-48 significant algorithm for the non-hybrid certificates.

7.2.3 Discussion

It is evident based on the results of this experiment that hybrid certificates and their underlying schemes significantly impact the performance of a TLS handshake. While it is possible to optimize the performance of the signature algorithms and adjust the TLS protocol to handle the increased certificate sizes, the fact is that hybrid signature schemes will always be slower than either of their component algorithms. As such, it is important to under-

stand both the advantages and disadvantages of hybrid signature schemes when modifying existing protocols or designing new ones.

From our observations, two factors are identified as acutely impacting the TLS handshake: the total amount of signature related data sent during the handshake and the performance of both the Sign and Verify operations of the hybrid signature algorithms. As show in Table 7.7, the true hybrid schemes produce a significantly larger certificate. This is caused by the need to encapsulate two separate public keys and combined hybrid signature. This, in turn, negatively influences the transmission time between the server and client.

Another impacting factor are the Sign and Verify operations that are part of the TLS handshake. In our setup, the client verifies the certificates of the server, intermediate CA, and the root CA. Additionally, the client must also verify the signature embedded in the server's *CertificateVerify* message. During the handshake, the server only signs the CertificateVerify message as all of the X.509 certificates are generated and signed beforehand. As shown by the verification performance of the individual signature algorithms in Chapter 6, different hybrid algorithm combinations can influence verification times. Even in examples where the individual algorithm's average verification performance is roughly similar (i.e., Dilithium 2 and RSA3072), the corresponding hybrid scheme requires at least twice the time to verify a message.

As with size, the overhead associated with a hybrid scheme's verification performance can be partially mitigated. If the only hybrid certificate in the chain is the root CA, only a single verification operation is influenced by the hybrid signature scheme. This is highlighted by comparing the results between Tables 7.11 and 7.10. Depending on the algorithm used for the rest of the signature operations, the average handshake time can be reduced significantly. When excluding the worst performing true hybrid schemes (i.e., FS–FS and FS–(EC)DSA #1) and all MQDSS-31-48 combinations, the collective average handshake completion time with a hybrid root CA is 2607 microseconds which is approximately a twenty percent increase over the collective average handshake time with a completely non-hybrid certificate chain. This could potentially be improved by using existing methods such as caching the root CA certificate locally on the client [90]. This would allow the server to send only the intermediate and server certificates in the TLS handshake thereby reducing the overall amount of data sent during the handshake.

Table 7.7. OQS X.509 Certificate Sizes for True Hybrid Schemes (bytes)

Name	Size (DER)	Size (PEM)
RSA3072	1548	2151
ECDSA P-256	898	1269
Dilithium 2	4000	5474
qTESLA-p-I	18242	24759
MQDSS-31-48	29211	39611
Dilithium 2_RSA3072	4786	6538
Dilithium 2_DSA3072 #1	5286	7213
Dilithium 2_DSA3072 #2	5285	7213
Dilithium 2_DSA3072 #3	5333	7278
Dilithium 2_P-256 #1	4603	6290
Dilithium 2_P-256 #2	4601	6286
Dilithium 2_P-256 #3	4650	6351
Dilithium 2_qTESLA-p-I	21477	29138
Dilithium 2_MQDSS-31-48	32450	43999
qTESLA-p-I_RSA3072	19028	25823
qTESLA-p-I_DSA3072 #1	19528	26501
qTESLA-p-I_DSA3072 #2	19528	26501
qTESLA-p-I_DSA3072 #3	19560	26542
qTESLA-p-I_P-256 #1	18843	25571
qTESLA-p-I_P-256 #2	18844	25575
qTESLA-p-I_P-256 #3	18875	25616
MQDSS-31-48_RSA3072	30001	40684
MQDSS-31-48_DSA3072 #1	30500	41358
MQDSS-31-48_DSA3072 #2	30501	41358
MQDSS-31-48_DSA3072 #3	30532	41403
MQDSS-31-48_P-256 #1	29817	40432
MQDSS-31-48_P-256 #2	29817	40432
MQDSS-31-48_P-256 #3	29850	40476
MQDSS-31-48_qTESLA-p-I	46692	63283

Table 7.9. TLS 1.3 Performance for Specified True Hybrid Schemes (μs/100,000 iterations)

Name	Hybrid Chain				Hybrid Root CA (primary)				Hybrid Root CA (secondary)			
	Min.	Max.	Mean	SD	Min.	Max.	Mean	SD	Min.	Max.	Mean	SD
Dilithium 2_RSA3072	3892	6996	4274	283	1184	4875	1435	242	2745	13819	2900	226
Dilithium 2_DSA3072_1	4132	49165	8266	4588	1882	5388	2112	214	-	-	-	-
Dilithium 2_DSA3072_2	4102	6814	4375	236	1875	3791	2106	204	-	-	-	-
Dilithium 2_DSA3072_3	4819	7786	5122	264	1936	4492	2156	205	-	-	-	-
Dilithium 2_P-256_1	2971	25897	5418	2571	1598	4364	1828	216	1493	2143	1548	35
Dilithium 2_P-256_2	2963	5280	3234	246	1597	4968	1822	225	1494	5945	1547	43
Dilithium 2_P-256_3	3416	5953	3703	250	1627	4529	1829	202	1526	2861	1578	38
Dilithium 2_MQDSS-31-48	30722	33573	31337	274	8703	11301	9048	259	30753	46210	31471	599
Dilithium 2_qTESLA-p-I	3005	57788	9578	6528	1585	3969	1795	210	2512	10662	3282	802
qTESLA-p-I_RSA3072	5080	13336	6010	836	2330	10917	3110	836	3050	14052	3197	260
qTESLA-p-I_DSA3072_1	5358	61427	12009	7025	3058	11462	3813	783	-	-	-	-
qTESLA-p-I_DSA3072_2	5294	12663	6081	782	3065	12538	3834	799	-	-	-	-
qTESLA-p-I_DSA3072_3	6021	15214	6850	842	3108	10210	3876	767	-	-	-	-
qTESLA-p-I_P-256_1	4195	41947	8437	4464	2772	18086	3543	828	1821	8937	1882	104
qTESLA-p-I_P-256_2	4158	11360	4928	797	2756	9995	3515	788	1812	6152	1875	65
qTESLA-p-I_P-256_3	4637	12918	5440	820	2803	10797	3572	800	1850	3884	1976	110
MQDSS-31-48_RSA3072	32608	40712	33155	351	30721	35442	31253	225	9493	18777	9643	480
MQDSS-31-48_DSA3072_1	32873	35365	33379	197	30666	53965	31410	1634	-	-	-	-
MQDSS-31-48_DSA3072_2	33009	34258	33551	177	31422	59931	32130	2143	-	-	-	-
MQDSS-31-48_DSA3072_3	33574	36513	34169	206	30947	32563	31518	180	-	-	-	-
MQDSS-31-48_P-256_1	31404	33536	31935	182	30956	32127	31493	170	8729	23709	8880	456
MQDSS-31-48_P-256_2	31512	32740	32106	173	30502	42325	30965	455	8672	19093	8839	413
MQDSS-31-48_P-256_3	31896	33466	32453	174	32004	35898	33325	526	8519	9678	8648	72
MQDSS-31-48_qTESLA-p-I_2	31939	36698	32943	674	31161	35902	31682	287	9366	15707	10058	590

Table 7.10. Percentage Difference of Minimum and Mean TLS Handshake Completion Times between a Completely Hybrid Certificate Chain and Completely Non-Hybrid Certificate Chain. The percentages are a computational increase of a completely hybrid certificate chain over that of a certificate chain composed entirely of the component algorithm.

Name	Entire chain composed of 1st component alg.		Entire chain composed of 2nd component alg.	
	Min.	Mean	Min.	Mean
Dilithium 2_RSA3072	71.12%	67.81%	31.53%	35.35%
Dilithium 2_DSA3072_1	72.80%	83.35%	-	-
Dilithium 2_DSA3072_2	72.60%	68.55%	-	-
Dilithium 2_DSA3072_3	76.68%	73.14%	-	-
Dilithium 2_P-256_1	62.17%	74.60%	66.00%	80.58%
Dilithium 2_P-256_2	62.07%	57.45%	65.91%	67.47%
Dilithium 2_P-256_3	67.10%	62.84%	70.43%	71.59%
Dilithium 2_MQDSS-31-48	90.36%	89.68%	0.85%	1.14%
Dilithium 2_qTESLA-p-I	62.60%	85.63%	24.39%	68.21%
qTESLA-p-I_RSA3072	55.28%	49.33%	47.54%	54.03%
qTESLA-p-I_DSA3072_1	57.60%	74.64%	-	-
qTESLA-p-I_DSA3072_2	57.08%	49.93%	-	-
qTESLA-p-I_DSA3072_3	62.27%	55.55%	-	-
qTESLA-p-I_P-256_1	45.84%	63.91%	75.92%	87.53%
qTESLA-p-I_P-256_2	45.36%	38.21%	75.71%	78.65%
qTESLA-p-I_P-256_3	51.00%	44.03%	78.22%	80.66%
MQDSS-31-48_RSA3072	6.58%	6.56%	91.67%	91.67%
MQDSS-31-48_DSA3072_1	7.34%	7.18%	-	-
MQDSS-31-48_DSA3072_2	7.72%	7.66%	-	-
MQDSS-31-48_DSA3072_3	9.27%	9.33%	-	-
MQDSS-31-48_P-256_1	3.00%	2.99%	96.71%	96.71%
MQDSS-31-48_P-256_2	3.34%	3.50%	96.72%	96.72%
MQDSS-31-48_P-256_3	4.50%	4.54%	96.76%	96.76%
MQDSS-31-48_qTESLA-p-I	4.63%	5.96%	90.76%	90.76%

Table 7.11. Percentage Difference of Minimum and Mean TLS Handshake Completion Times between a Partially Hybrid Certificate Chain and a Completely Non-Hybrid Certificate Chain. The percentages are a computational increase of a partially hybrid certificate chain over that of a certificate chain composed entirely of the component algorithm.

Name	Entire chain composed of 1st component alg.		Entire chain composed of 2nd component alg.	
	Min.	Mean	Min.	Mean
Dilithium 2_RSA3072	5.07%	4.11%	2.91%	4.72%
Dilithium 2_DSA3072_1	40.28%	34.85%	-	-
Dilithium 2_DSA3072_2	40.05%	34.66%	-	-
Dilithium 2_DSA3072_3	41.94%	36.18%	-	-
Dilithium 2_P-256_1	29.66%	24.73%	32.35%	32.04%
Dilithium 2_P-256_2	29.62%	24.48%	32.40%	32.00%
Dilithium 2_P-256_3	30.92%	24.77%	33.81%	33.33%
Dilithium 2_MQDSS-31-48	87.08%	84.79%	0.95%	1.56%
Dilithium 2_qTESLA-p-I	29.09%	23.34%	9.55%	7.22%
qTESLA-p-I_RSA3072	2.49%	2.09%	12.62%	13.58%
qTESLA-p-I_DSA3072_1	25.70%	20.14%	-	-
qTESLA-p-I_DSA3072_2	25.87%	20.58%	-	-
qTESLA-p-I_DSA3072_3	26.90%	21.44%	-	-
qTESLA-p-I_P-256_1	18.04%	14.06%	44.54%	44.10%
qTESLA-p-I_P-256_2	17.56%	13.37%	44.26%	43.89%
qTESLA-p-I_P-256_3	18.94%	14.75%	45.41%	46.76%
MQDSS-31-48_RSA3072	0.85%	0.87%	71.93%	71.35%
MQDSS-31-48_DSA3072_1	0.67%	1.37%	-	-
MQDSS-31-48_DSA3072_2	3.06%	3.58%	-	-
MQDSS-31-48_DSA3072_3	1.57%	1.70%	-	-
MQDSS-31-48_P-256_1	1.60%	1.63%	88.43%	88.15%
MQDSS-31-48_P-256_2	0.13%	-0.05%	88.35%	88.10%
MQDSS-31-48_P-256_3	4.82%	7.03%	88.14%	87.84%
MQDSS-31-48_qTESLA-p-I	2.25%	2.21%	75.74%	69.73%

CHAPTER 8:
Conclusion

Any uncertainty surrounding the PQ transition, to include the increased complexity of PQ algorithms, will likely manifest in a reluctance to accept unknown risk by broadly adopting newer digital signature algorithms. This further reinforces the current fragile, monolithic ecosystem on which many fundamental Internet applications are based. Hybrid signature schemes offer a flexible approach to ensuring uninterrupted cryptographic security by combining PQ signature algorithms with well-accepted classical algorithms, thus offsetting the risk associated with adopting newer algorithms.

In this work, we evaluated the viability and performance of several hybrid signature schemes at both the algorithmic and protocol levels. Traditional hybridization techniques were examined to include concatenation and nesting, and a new type of true hybrid scheme was also introduced. To compare and contrast performance, we implemented the true hybrid digital signature schemes within a common cryptographic framework and evaluated their performance against traditional hybrid techniques via standalone cryptographic operations. Our results show that specific true hybrid signature schemes introduce negligible overhead when compared to concatenated hybrid schemes using the same component algorithms. The results also show that certain true hybrid combinations add additional computational overhead. In these examples, the efficiency decrease is directly influenced by how the true hybrid scheme interacts with the component algorithms. It is important to note that our implementation is not fully optimized for performance nor should it be considered secure without external review.

We also explored how hybrid digital signatures could be integrated into existing X.509 certificates and examined their performance by integrating both into the TLS 1.3 protocol. As first observed in [18], our work confirmed that the larger size of hybrid digital certificates and the increase in computational processing required to run two digital signature algorithms within a hybrid scheme have a significant impact on the total handshake time. Additionally, we found that integrating hybrid signatures into existing protocols within cryptographic libraries is a non-trivial task. Implementation requires an in-depth knowledge of the existing protocol standards, the cryptographic library internals, and the security features

of the programming language it is written in. While these impediments can be overcome individually, it highlights the role standardization plays in ensuring interoperability between different hardware and software builds.

8.1 Recommendations

Based on the results in Chapter 6 and the security properties detailed in Chapter 3, we recommend that true hybrid schemes should be used over traditional hybridization techniques, when possible. Out of the schemes compared in this paper, the true hybrid schemes presented in Section 3.3 are the only ones that provide additional security properties, such as hybrid verification, outside of what is offered by the individual component signature algorithms. The hybrid verification security property is essential in ensuring that both component signature algorithms are verified which directly supports the primary reason to use a hybrid signature scheme.

True hybrid signature schemes also provide developers and system engineers with options to solve unique cryptographic challenges. As an example, digital signatures can be used to sign official documents that are later part of the public record or of historical significance. In this situation, the security longevity of the signature algorithm is vitally important. An adversary could alter the document and retroactively apply a signature if the scheme used to originally sign is later broken. This can change the historical perspective of significant events with little to no recourse for stakeholders to prove otherwise. To combat this, a true hybrid signature scheme can be used to sign official documents with multiple signature algorithms. This solution is optimal because it does not rely on the longevity of a single signature algorithm and guarantees that both signature algorithms are used if the original signature is verified. Additionally, since signing documents is unlikely to be extremely time-sensitive, the increase in security outweighs the performance cost.

Given the comparable performance in our results between most component algorithm combinations, we recommend that the FS–RSA, FS–DSA and FS–ECDSA true hybrid schemes should be promoted for broader adoption. As shown by the certificate sampling in Section 7.1, RSA and ECDSA are the most commonly used signature algorithms in X.509 certificates. Additionally, one of the greater use cases for hybrid signature schemes is to provide a bridge between classical and PQ signature algorithms. By combining a FS-based

PQ signature algorithm with well-accepted algorithms like RSA and ECDSA, both of these true hybrid schemes deliver classic *and* PQ security.

Additionally, FS–RSA, FS–DSA and FS–ECDSA avoid some of the challenges associated with implementing true hybrid schemes. As discussed in Chapters 4 and 6, true hybrid schemes pose several challenges that can make it difficult for them to be securely implemented. For example, component signature algorithms are likely to have differing hash algorithms and challenge lengths which need to be reconciled before they can be securely used in a true hybrid scheme. This necessary modification impacts how the component algorithms function and can have unintended consequences on the overall security of the true hybrid scheme. FS–RSA, FS–DSA, and FS–ECDSA avoid this complication because only the FS component algorithm provides input into the challenge and the non-FS component algorithms in these hybrid schemes (i.e., RSA, DSA and ECDSA) can support all of the FS component hash algorithms.

In addition to reconciling differences between component signature algorithms, the mechanisms of each component algorithm must be considered during implementation. The interaction between simple design choices and these internal mechanisms can drastically impact the efficiency of the true hybrid scheme. For example, both qTESLA and Dilithium use a simple programmatic loop during signing to determine if a generated signature meets specific criteria. If the signature is invalid, the Sign operation restarts. This continues until a valid signature is found. As shown in Chapter 6, we found that separating this loop into two sub-functions introduced roughly a twenty percent increase in computation required to complete the Sign operation. This inefficiency is solely caused by the overhead associated with a function call and can be mitigated when the sub-functions are replaced with a single function that includes all of the components of the sub-function.

As a result of the complexities associated with implementing true hybrid schemes, we recommend that they should be integrated into existing cryptographic libraries in order to support protocols that can handle both their increased signature and key sizes and the overall performance cost. Cryptographic libraries represent the best mechanism for consistently ensuring general users and developers have standardized access to secure hybrid signature schemes. Assuming the library is well-supported and peer reviewed, the risk of incorrect implementation is shifted to an experienced team and away from a single

developer who may not be experienced with the intricacies of implementing cryptographic code. Cryptographic libraries are also centralized, which can be advantageous for both optimizations and security. For example, any potential vulnerabilities in either the hybrid scheme or individual component algorithms that are found after adoption can be fixed in a single library vice in every single application that is using the hybrid scheme.

It is important to note that true hybrid schemes are not a "one size fits all" solution for every use case. For example, true hybrid schemes do not innately support backwards compatibility with legacy systems. As with any cryptographic use case, developers must identify the security requirements of a system *prior* to choosing a cryptographic solution. If backwards compatibility is valued more than hybrid verification, other solutions like nested hybrid schemes may be a better fit.

8.2 Future Work

While true hybrid signature schemes provide flexible solutions and can combat uncertainty during the PQ transition, their wide-spread adoption requires significant work beyond what is presented in this paper. Interoperability between software and hardware implementations is essential for almost every digital signature use case and standards are what provide implementations with a common foundation from which to build. Even if hybrid schemes are implemented independently within cryptographic libraries, there is no guarantee that software compiled with one will work with software compiled with another. As such, future work should focus on identifying and optimizing component algorithm combinations for use with true hybrid schemes. These efforts can then be used to guide future standardization efforts and are essential to the widespread adoption of true hybrid signature schemes.

Additionally, protocol standards may need to be updated to reflect any modifications required for their use with hybrid signature schemes. This work only examined true hybrid signature schemes in the context of the TLS 1.3 protocol; however, this may not represent the best use case for hybrid signatures. Several other protocols that also use digital signatures and certificates may benefit from hybridization, such as SSH [111] and Domain Name System Security Extensions (DNSSEC) [112]. More realistic and robust network testing is also needed to further highlight the impact of true hybrid schemes on current and future protocols beyond what is covered in this work.

List of References

[1] B. Schneier, *Applied Cryptography: Protocols, Algorithms, and Source Code in C*, 2nd ed. Somerset, New Jersey, USA: Wiley, 1996.

[2] E. Rescorla, "The transport layer security (TLS) protocol version 1.3," RFC Editor, Fremont, CA, USA, RFC 8446, 2018. [Online]. Available: https://tools.ietf.org/html/rfc8446

[3] T. Ylonen and C. Lonvick, "The secure shell (SSH) authentication protocol," RFC Editor, Fremont, CA, USA, RFC 4252, 2006. Available: https://tools.ietf.org/html/rfc4252

[4] M. Naor, "On cryptographic assumptions and challenges," in *Advances in Cryptology*, 2003. [Online]. doi: 10.1007/978-3-540-45146-4_6.

[5] S. Aaronson, "P=?NP," Weizmann Institute of Science, Rehovot, Israel, Tech. Rep. 004, 2017. [Online]. Available: https://eccc.weizmann.ac.il/report/2017/004/

[6] J. Katz, "Cryptographic hardness assumptions," in *Digital Signatures*, J. Katz, Ed. Boston, MA, USA: Springer US, 2010, pp. 35–66.

[7] P. Shor, "Algorithms for quantum computation: Discrete logarithms and factoring," in *Proceedings 35th Annual Symposium on Foundations of Computer Science*, 1994. [Online]. doi: 10.1109/SFCS.1994.365700.

[8] L. Chen, S. Jordan, Y.-K. Liu, D. Moody, R. Peralta, R. Perlner, and D. Smith-Tone, "Report on post-quantum cryptography," National Institute of Standards and Technology, Tech. Rep. NIST IR 8105, 2016. [Online]. doi: 10.6028/NIST.IR.8105.

[9] National Institute of Standards and Technology. "Post-quantum cryptography," Jan. 3, 2017. [Online]. Available: https://csrc.nist.gov/Projects/post-quantum-cryptography

[10] W. Diffie and M. Hellman, "New directions in cryptography," *IEEE Transactions on Information Theory*, vol. 22, no. 6, pp. 644–654, Nov. 1976. [Online]. doi: https://doi.org/10.1109/TIT.1976.1055638.

[11] N. Ramlee and E. S. Ismail, "A new directed signature scheme with hybrid problems," *AIP Conference Proceedings*, vol. 1571, no. 1, pp. 994–998, Nov. 2013. [Online]. doi: https://doi.org/10.1063/1.4858783.

[12] H. M. Elkamchouchi, A. E. Takieldeen, and M. A. Shawky, "An advanced hybrid technique for digital signature scheme," in *2018 5th International Conference on Electrical and Electronic Engineering (ICEEE)*, 2018. [Online]. doi: 10.1109/ICEEE2.2018.8391365.

[13] S. A. Vanstone, R. Gallant, R. J. Lambert, L. A. Pintsov, F. W. R. Jr, and A. Singer, "Hybrid signature scheme," U.S. Patent 7 877 610B2, Jan. 25, 2011. Available: https://patents.google.com/patent/US7877610B2/en

[14] A. Sarkar and S. Tripathi, "Design of a dual signature scheme using ECDSA in set protocol," *International Journal of Computer Applications*, vol. 88, no. 11, pp. 1–5, Feb. 2014. [Online]. doi: https://doi.org/10.5120/15393-3565.

[15] N. Bindel, U. Herath, M. McKague, and D. Stebila, "Transitioning to a quantum-resistant public key infrastructure," Cryptology ePrint Archive, Tech. Rep. 460, 2017. [Online]. Available: http://eprint.iacr.org/2017/460

[16] E. Crockett, C. Paquin, and D. Stebila, "Prototyping post-quantum and hybrid key exchange and authentication in TLS and SSH," Cryptology ePrint Archive, Bellevue, WA, USA, Tech. Rep. 858, 2019. [Online]. Available: https://eprint.iacr.org/2019/858

[17] P. Kampanakis and D. Sikeridis, "Two PQ signature use-cases: Non-issues, challenges and potential solutions," Cryptology ePrint Archive, Bellevue, WA, USA, Tech. Rep. 1276, 2019. [Online]. Available: http://eprint.iacr.org/2019/1276

[18] D. Sikeridis, P. Kampanakis, and M. Devetsikiotis, "Post-quantum authentication in TLS 1.3: A performance study," Cryptology ePrint Archive, Bellevue, WA, USA, Tech. Rep. 071, 2020. [Online]. Available: http://eprint.iacr.org/2020/071

[19] P. Kampanakis, P. Panburana, E. Daw, and D. V. Geest, "The viability of post-quantum X.509 certificates," Cryptology ePrint Archive, Bellevue, WA, USA, Tech. Rep. 063, 2018. [Online]. Available: https://eprint.iacr.org/2018/063

[20] L. Gladiator, T. Stockert, and J. Wirth, *OpenSSL Hybrid Certificates*, ver. v.5.11-rc7, Darmstadt, Germany, 2021. [Online]. Available: https://github.com/CROSSINGTUD/openssl-hybrid-certificates

[21] D. Skatz, J. Braun, and J. Wirth, "BC Hybrid Certificates," ver. 1.0.1, Darmstadt, Germany, 2019 [Online]. Available: https://github.com/CROSSINGTUD/bc-hybrid-certificates

[22] Open Quantum Safe Project, "OpenSSL," ver. 2020-08, Waterloo, Ontario, 2020. [Online]. Available: https://github.com/open-quantum-safe/openssl

[23] B. Hale and N. Bindel, private communication, Mar. 2020.

[24] National Institute of Standards and Technology, "Digital signature standard (DSS)," Gaithersburg, MD, USA, Tech. Rep. NIST FIPS 186-4, 2013. [Online]. doi: 10.6028/NIST.FIPS.186-4.

[25] F. Vercauteren, "Final report on main computational assumptions in cryptography," European Network of Excellence in Cryptology II, Leuven, Belgium, Tech. Rep. ICT-2007-216676, 2008. [Online]. Available: https://www.ecrypt.eu.org/ecrypt2/documents/D.MAYA.6.pdf

[26] D. Aggarwal and U. Maurer, "Breaking RSA generically is equivalent to factoring," Cryptology ePrint Archive, Tech. Rep. 260, 2008. [Online]. Available: http://eprint.iacr.org/2008/260

[27] D. Brown, "Breaking RSA may be as difficult as factoring," *Journal of Cryptology*, vol. 29, no. 1, pp. 220–241, Jan. 2016. [Online]. doi: https://doi.org/10.1007/s00145-014-9192-y.

[28] S. Vaudenay, "The security of DSA and ECDSA," in *Public Key Cryptography — PKC 2003*, 2002. [Online]. doi: 10.1007/3-540-36288-6_23.

[29] D. Johnson, A. Menezes, and S. Vanstone, "The elliptic curve digital signature algorithm (ECDSA)," *International Journal of Information Security*, vol. 1, no. 1, pp. 36–63, Aug. 2001. [Online]. doi: https://doi.org/10.1007/s102070100002.

[30] C. Gidney and M. Ekerå, "How to factor 2048 bit RSA integers in 8 hours using 20 million noisy qubits," 2019. [Online]. Available: http://arxiv.org/abs/1905.09749

[31] A. Dang, C. D. Hill, and L. C. L. Hollenberg, "Optimising matrix product state simulations of Shor's algorithm," *Quantum*, vol. 3, p. 116, Jan. 2019. [Online]. doi: https://doi.org/10.22331/q-2019-01-25-116.

[32] J. Sevilla and C. J. Riedel, "Forecasting timelines of quantum computing," 2020.

[33] J. Ding and D. Schmidt, "Rainbow, a new multivariable polynomial signature scheme," in *Applied Cryptography and Network Security*, 2005. [Online]. doi: 10.1007/11496137_12.

[34] L. Goubin and N. T. Courtois, "Cryptanalysis of the TTM cryptosystem," in *Advances in Cryptology*, 2000. [Online]. doi: 10.1007/3-540-44448-3_4.

[35] B.-Y. Yang and J.-M. Chen, "All in the XL family: Theory and practice," in *Information Security and Cryptology – ICISC 2004*, 2005. [Online]. doi: 10.1007/11496618_7.

[36] J. Ding, B.-Y. Yang, C.-H. O. Chen, M.-S. Chen, and C.-M. Cheng, "New differential-algebraic attacks and reparametrization of rainbow," in *Applied Cryptography and Network Security*, Berlin, Heidelberg, 2008. [Online]. doi: 10.1007/978-3-540-68914-0_15.

[37] A. Petzoldt, S. Bulygin, and J. Buchmann, "Selecting parameters for the rainbow signature scheme," in *Post-Quantum Cryptography*. Springer, 2010. [Online]. doi: 10.1007/978-3-642-12929-2_16.

[38] S. Goldwasser, S. Micali, and R. L. Rivest, "A digital signature scheme secure against adaptive chosen-message attacks," *SIAM Journal on Computing*, vol. 17, no. 2, pp. 281–308, Apr. 1988. [Online]. doi: https://doi.org/10.1137/0217017.

[39] D. Hofheinz and T. Jager, "Tightly secure signatures and public-key encryption," in *Advances in Cryptology*, 2012. [Online]. doi: 10.1007/978-3-642-32009-5_35.

[40] J. Bos and D. Chaum, "Provably unforgeable signatures," in *Advances in Cryptology*, 1993. [Online]. doi: 10.1007/3-540-48071-4_1.

[41] G. Alagic, J. Alperin-Sheriff, D. Apon, D. Cooper, Q. Dang, Y.-K. Liu, C. Miller, D. Moody, R. Peralta, R. Perlner, A. Robinson, and D. Smith-Tone, "Status report on the first round of the NIST post-quantum cryptography standardization process," National Institute of Standards and Technology, Gaithersburg, MD, USA, Tech. Rep. NIST IR 8240, 2019. [Online]. doi: 10.6028/NIST.IR.8240.

[42] A. Fiat and A. Shamir, "How to prove yourself: practical solutions to identification and signature problems," in *Advances in Cryptology*. Springer, 1987. [Online]. doi: 10.1007/3-540-47721-7_12.

[43] S. Shahandashti, "Contributions to secure and privacy-preserving use of electronic credentials," Ph.D. dissertation, University of Wollongong, Wollongong, New South Wales, 2009. [Online]. Available: https://ro.uow.edu.au/theses/3036

[44] H. Ong and C. P. Schnorr, "Fast signature generation with a Fiat Shamir-like scheme," in *Advances in Cryptology*, 1991. [Online]. doi: 10.1007/3-540-46877-3_38.

[45] L. Guillou and J.-J. Quisquater, "A "paradoxical" identity-based signature scheme resulting from zero-knowledge," in *Advances in Cryptology*, 1990. [Online]. doi: 10.1007/0-387-34799-2_16.

[46] Y. Desmedt, "Fiat–Shamir identification protocol and the Feige–Fiat–Shamir signature scheme," in *Encyclopedia of Cryptography and Security*, H. C. A. van Tilborg and S. Jajodia, Eds. Boston, MA, USA: Springer US, 2011, pp. 457–458. Available: https://doi.org/10.1007/978-1-4419-5906-5_319

[47] M. Abdalla, J. H. An, M. Bellare, and C. Namprempre, "From identification to signatures via the Fiat-Shamir transform: Minimizing assumptions for security and forward-security," in *Advances in Cryptology*, 2002. [Online]. doi: 10.1007/3-540-46035-7_28.

[48] D. Pointcheval and J. Stern, "Security proofs for signature schemes," in *Advances in Cryptology*, 1996. [Online]. doi: 10.1007/3-540-68339-9_33.

[49] M.-S. Chen, A. Hülsing, J. Rijneveld, S. Samardjiska, and P. Schwabe, "MQDSS specifications," Radboud University, Nijmegen, Netherlands, Tech. Rep., 2020. [Online]. Available: http://mqdss.org/files/mqdssVer2point1.pdf

[50] C. Schnorr, "Efficient signature generation by smart cards," *Journal of Cryptology*, vol. 4, no. 3, pp. 161–174, Jan. 1991. [Online]. doi: https://doi.org/10.1007/BF00196725.

[51] E. Alkim, P. S. L. M. Barreto, N. Bindel, J. Kramer, P. Longa, and J. E. Ricardini, "The lattice-based digital signature scheme qTESLA," Cryptology ePrint Archive, Tech. Rep. 085, 2019. [Online]. Available: http://eprint.iacr.org/2019/085

[52] R. Rivest, A. Shamir, and L. Adleman, "A method for obtaining digital signatures and public-key cryptosystems," *Communications of the ACM*, vol. 21, no. 2, pp. 120–126, 1978. [Online]. doi: https://doi-org.libproxy.nps.edu/10.1145/359340.359342.

[53] R. Rivest, A. Shamir, and L. Adleman, "Cryptographic communications system and method," U.S. Patent 4 405 829A, Sep. 20, 1983. Available: https://patents.google.com/patent/US4405829/en

[54] T. Elgamal, "A public key cryptosystem and a signature scheme based on discrete logarithms," *IEEE Transactions on Information Theory*, vol. 31, no. 4, pp. 469–472, July 1985. [Online] doi: 10.1109/TIT.1985.1057074.

[55] L. Guillou and J.-J. Quisquater, "A practical zero-knowledge protocol fitted to security microprocessor minimizing both transmission and memory," in *Advances in Cryptology*. Springer, 1988. [Online]. doi: 10.1007/3-540-45961-8_11.

[56] C. Schnorr, "Method for identifying subscribers and for generating and verifying electronic signatures in a data exchange system," U.S. Patent 4 995 082A, Feb. 19, 1991. Available: https://patents.google.com/patent/US4995082A/en

[57] National Institute of Standards and Technology, "Digital signature standard (DSS)," Gaithersburg, MD, USA, Tech. Rep. NIST FIPS 186-5 (draft), 2019. [Online]. doi: 10.6028/NIST.FIPS.186-5.

[58] M. Suárez-Albela, P. Fraga-Lamas, and T. Fernández-Caramés, "A practical evaluation on RSA and ECC-based cipher suites for IOT high-security energy-efficient fog and mist computing devices," *Sensors*, vol. 18, no. 11, Nov. 2018. [Online]. doi: http://dx.doi.org/10.3390/s18113868.

[59] K. Sakumoto, T. Shirai, and H. Hiwatari, "Authentication device, authentication method, program, and signature generation device," U.S. Patent 8 522 033B2, Aug. 27, 2013. Available: https://patents.google.com/patent/US8522033B2/ko

[60] P. Gaborit and C. Melchor, "Cryptographic method for communicating confidential information," European Union Patent 2 537 284B1, Apr. 20, 2016. Available: https://patents.google.com/patent/EP2537284B1/en

[61] J. Ding, "Method to produce new multivariate public key cryptosystems," U.S. Patent 7 961 876B2, June 30, 2011. Available: https://patents.google.com/patent/US7961876B2/en

[62] A. Truskovsky, A. Yamada, M. Brown, and G. Gutoski, "Using a digital certificate with multiple cryptosystems," U.S. Patent 9 660 978B1, May 23, 2017. Available: https://patents.google.com/patent/US9660978B1/en

[63] D. Boneh, "Twenty years of attacks on the RSA cryptosystem," *Notices of the American Mathematical Society*, vol. 46, no. 2, pp. 203–213, Feb. 1999. [Online]. Available: https://www.ams.org/journals/notices/199902/boneh.pdf

[64] J. Jonsson, K. Moriarty, B. Kaliski, and A. Rusch, "PKCS #1: RSA cryptography specifications," RFC Editor, Fremont, CA, USA, RFC 8017, 2016. [Online]. Available: https://tools.ietf.org/html/rfc8017

[65] S. Kakvi and E. Kiltz, "Optimal security proofs for full domain hash, revisited," *Journal of Cryptology*, vol. 31, no. 1, pp. 276–306, Jan. 2018. [Online]. doi: https://doi.org/10.1007/s00145-017-9257-9.

[66] T. Jager, S. Kakvi, and A. May, "On the security of the PKCS#1 v1.5 signature scheme," Cryptology ePrint Archive, Bellevue, WA, USA, Tech. Rep. 855, 2018. [Online]. Available: http://eprint.iacr.org/2018/855

[67] S. Kakvi, "On the security of RSA-PSS in the wild," Cryptology ePrint Archive, Fremont, CA, USA, Tech. Rep. 1268, 2019. [Online]. Available: http://eprint.iacr.org/2019/1268

[68] C. Lindenberg, K. Wirt, and J. Buchmann, "Formal proof for the correctness of RSA-PSS," Cryptology ePrint Archive, Bellevue, WA, USA, Tech. Rep. 011, 2006. [Online]. Available: http://eprint.iacr.org/2006/011

[69] A. Regenscheid, "Digital signature standard (DSS): Elliptic curve domain parameters," National Institute of Standards and Technology, Tech. Rep., 2019. [Online]. doi: 10.6028/NIST.FIPS.186-5-draft.

[70] M. Fersch, E. Kiltz, and B. Poettering, "On the provable security of (EC)DSA signatures," in *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*. Association for Computing Machinery, 2016. [Online]. doi: 10.1145/2976749.2978413.

[71] J. Buchmann, D. Butin, F. Göpfert, and A. Petzoldt, "Post-quantum cryptography: State of the art," in *The New Codebreakers: Essays Dedicated to David Kahn on the Occasion of His 85th Birthday*, P. Ryan, D. Naccache, and J.-J. Quisquater, Eds. Berlin, Heidelberg, Germany: Springer, 2016, pp. 88–108.

[72] L. Ducas, T. Lepoint, V. Lyubashevsky, P. Schwabe, G. Seiler, and D. Stehle, "CRYSTALS-Dilithium: Digital signatures from module lattices," Cryptology ePrint Archive, Bellevue, WA, USA, Tech. Rep. 633, 2017. [Online]. Available: http://eprint.iacr.org/2017/633

[73] T. Pornin, "New efficient, constant-time implementations of falcon," Cryptology ePrint Archive, Bellevue, WA, USA, Tech. Rep. 893, 2019. [Online]. Available: https://eprint.iacr.org/2019/893

[74] J. Hoffstein, N. Howgrave-Graham, J. Pipher, J. H. Silverman, and W. Whyte, "Ntrusign: Digital signatures using the NTRU lattice," in *Topics in Cryptology*, 2003. [Online]. doi: https://doi-org.libproxy.nps.edu/10.1007/3-540-36563-X_9.

[75] C. Gentry, C. Peikert, and V. Vaikuntanathan, "Trapdoors for hard lattices and new cryptographic constructions," Cryptology ePrint Archive, Tech. Rep. 432, 2007. [Online]. Available: http://eprint.iacr.org/2007/432

[76] J. Patarin and L. Goubin, "Trapdoor one-way permutations and multivariate polynomials," in *Information and Communications Security*, 1997. [Online]. doi: 10.1007/BFb0028491.

[77] K. Sakumoto, T. Shirai, and H. Hiwatari, "Public-key identification schemes based on multivariate quadratic polynomials," in *Advances in Cryptology*, 2011. [Online]. doi: 10.1007/978-3-642-22792-9_40.

[78] M. Garey and D. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*, 1st ed. San Francisco, CA, USA: W. H. Freeman, 1979.

[79] T. Matsumoto and H. Imai, "Public quadratic polynomial-tuples for efficient signature-verification and message-encryption," in *Advances in Cryptology*. Springer, 1988. [Online]. doi: 10.1007/3-540-45961-8_39.

[80] L. Xiaoyu, S. Tang, J. Chen, and L. Xu, "MQ signature and proxy signature schemes with exact security based on UOV signature," Cryptology ePrint Archive, Bellevue, WA, USA, Tech. Rep. 877, 2013. [Online]. Available: https://eprint.iacr.org/2013/877

[81] J. Patarin, "Hidden fields equations and isomorphisms of polynomials: Two new families of asymmetric algorithms," in *Advances in Cryptology*, 1996. [Online]. doi: 10.1007/3-540-68339-9_4.

[82] A. Kipnis, J. Patarin, and L. Goubin, "Unbalanced oil and vinegar signature schemes," in *Advances in Cryptology*, 1999. [Online]. doi: 10.1007/3-540-48910-X_15.

[83] K. Sakumoto, T. Shirai, and H. Hiwatari, "On provable security of UOV and HFE signature schemes against chosen-message attack," in *Post-Quantum Cryptography*. Springer, 2011. [Online]. doi: 10.1007/978-3-642-25405-5_5.

[84] A. Casanova, J. Faugère, G. Macario-Rat, J. Patarin, L. Perret, and J. Ryckeghem, "GeMSS: Agreat multivariate short signature," Universite Pierre et Marie Curie, Paris, France, Tech. Rep., 2017. [Online]. Available: https://www-polsys.lip6.fr/Links/NIST/GeMSS_specification.pdf

[85] J. Patarin, N. Courtois, and L. Goubin, "QUARTZ, 128-Bit Long Digital Signatures," in *Topics in Cryptology*, 2001. [Online]. doi: 10.1007/3-540-45353-9_21.

[86] National Institute of Standards and Technology, "Security requirements for cryptographic modules," Gaithersburg, MD, USA, Tech. Rep. NIST FIPS 140-3, 2019. [Online]. doi: 10.6028/NIST.FIPS.140-3.

[87] Open Quantum Safe Project, "Liboqs," ver. 0.3.0-rc1, Waterloo, Ontario, 2020. [Online]. Available: https://github.com/open-quantum-safe/liboqs

[88] P. Kampanakis and Q. Dang, "Internet X.509 public key infrastructure: Additional algorithm identifiers for RSASSA-PSS and ECDSA using SHAKEs," RFC Editor, Fremont, CA, USA, Internet Draft 8692, 2019. [Online]. Available: https://tools.ietf.org/id/draft-ietf-lamps-pkix-shake-07.html

[89] National Institute of Standards and Technology, "Secure hash standard (SHS)," Gaithersburg, MD, USA, Tech. Rep. NIST FIPS 180-4, 2015. [Online]. doi: 10.6028/NIST.FIPS.180-4.

[90] D. Cooper, "Internet X.509 public key infrastructure certificate and certificate revocation list (CRL) profile," RFC Editor, Fremont, CA, USA, RFC 5280, 2008. [Online]. Available: https://tools.ietf.org/html/rfc5280

[91] C. Adams and M. Blinov, “Alternative certificate formats for the public-key infrastructure using X.509 (PKIX) certificate management protocols,” Internet Requests for Comments, RFC Editor, RFC 4212, 10 2005. Available: https://tools.ietf.org/html/rfc4212

[92] J. Linn, “Privacy enhancement for internet electronic mail: Part 1: Message encryption and authentication procedures,” RFC Editor, Fremont, CA, USA, RFC 1421, 1993. [Online]. Available: https://tools.ietf.org/html/rfc1421

[93] S. Leonard and S. Josefsson, “Textual encodings of PKIX, PKCS, and CMS structures,” RFC Editor, Fremont, CA, USA, RFC 7468, 2015. [Online]. Available: https://tools.ietf.org/html/rfc7468

[94] B. Kaliski, “PKCS #7: Cryptographic message syntax,” RFC Editor, Fremont, CA, USA, RFC 2315, 1998. [Online]. Available: https://tools.ietf.org/html/rfc2315

[95] *Information technology – ASN.1 encoding rules: Specification of basic encoding rules (BER), canonical encoding rules (CER) and distinguished encoding rules (DER)*, ITU-T X.690, 2015.

[96] The Open Quantum Safe Project. "Open quantum safe," Accessed Mar. 19, 2020. [Online]. Available: https://openquantumsafe.org/

[97] S. Fluhrer, S. Mister, A. Truskovsky, M. Ounsworth, P. Kampanakis, and D. Geest, “Multiple public-key algorithm X.509 certificates,” RFC Editor, Fremont, CA, USA, Internet Draft, 2018. [Online]. Available: https://tools.ietf.org/html/draft-truskovsky-lamps-pq-hybrid-x509-00

[98] J. Contreras and S. Bradner, “Intellectual property rights in IETF technology,” RFC Editor, Fremont, CA, USA, RFC 8179, 2017. [Online]. Available: https://tools.ietf.org/html/rfc8179

[99] S. Garfinkel and G. Spafford. *Web Security, Privacy & Commerce*. O’Reilly, Sebastopol, CA, USA, 2011. [Online]. Available: https://learning.oreilly.com/library/view/web-security-privacy/0596000456/

[100] B. Dowling, M. Fischlin, F. Günther, and D. Stebila, “A cryptographic analysis of the TLS 1.3 handshake protocol,” New York, NY, USA, 2020. [Online]. doi: 10.3929/ETHZ-B-000438744.

[101] OpenSSL Software Foundation, “OpenSSL,” ver. 1.1.1, Newark, DE, USA, 2018. [Online]. Available: https://github.com/openssl/openssl

[102] L. Torvalds, *Linux*, ver. 2018-11, Portland, OR, USA, 2019. [Online]. Available: https://github.com/torvalds/linux

[103] G. Pailoni, “How to benchmark code execution times on intel IA-32 and IA-64 instruction set architectures,” Intel Corporation, Santa Clara, CA, USA, Tech. Rep., 2020. [Online]. Available: https://www.intel.com/content/dam/www/public/us/en/documents/white-papers/ia-32-ia-64-benchmark-code-execution-paper.pdf

[104] SUPERCOP. "eBACS: ECRYPT benchmarking of cryptographic systems," Oct. 18, 2020. [Online]. Available: https://bench.cr.yp.to/supercop.html

[105] P. Kampanakis and Q. Dang, “Use of the SHAKE One-Way Hash Functions in the Cryptographic Message Syntax (CMS),” RFC Editor, Fremont, CA, USA, RFC 8702, 2020. [Online]. Available: https://tools.ietf.org/html/rfc8702

[106] E. Barker, L. Chen, A. Roginsky, A. Vassilev, R. Davis, and S. Simon, “Recommendation for pair-wise key establishment using integer factorization cryptography,” National Institute of Standards and Technology, Gaithersburg, MD, USA, Tech. Rep. NIST SP 800-56Br2, 2019. [Online]. doi: 10.6028/NIST.SP.800-56Br2.

[107] Majestic-12 Ltd. "The majestic million," Nov. 20, 2020. [Online]. Available: https://majestic.com/reports/majestic-million

[108] K. McKay and D. Cooper, “Guidelines for the selection, configuration, and use of transport layer security (TLS) implementations,” National Institute of Standards and Technology, Gaithersburg, MD, USA, Tech. Rep. NIST SP 800-52r2, 2019. [Online]. doi: 10.6028/NIST.SP.800-52r2.

[109] E. Barker and Q. Dang, “Recommendation for key management part 3: Application-specific key management guidance,” National Institute of Standards and Technology, Gaithersburg, MD, USA, Tech. Rep. NIST SP 800-57Pt3r1, 2015. [Online]. doi: 10.6028/NIST.SP.800-57Pt3r1.

[110] M. Kerrisk, “Clock_getres(2),” Linux Programmer’s Manual, Dec. 21, 2020. [Online]. Available: https://man7.org/linux/man-pages/man2/clock_getres.2.html

[111] K. Igoe and D. Stebila, “X.509 certificates for secure shell authentication,” RFC Editor, Fremont, CA, USA, Internet Draft, 2011. [Online]. Available: https://tools.ietf.org/html/rfc6187

[112] R. Arends, R. Austein, M. Larson, D. Massey, and S. Rose, “DNS security introduction and requirements,” RFC Editor, Fremont, CA, USA, RFC, 2005. [Online]. Available: https://tools.ietf.org/html/rfc4033

Initial Distribution List

1. Defense Technical Information Center
   Ft. Belvoir, Virginia

2. Dudley Knox Library
   Naval Postgraduate School
   Monterey, California