POLITECNICO DI TORINO

Master degree course in Cybersecurity

Master Degree Thesis

A Crypto-Agile Framework for Integrating Post-Quantum Cryptography into Mozilla Firefox and NSS

Supervisor
prof. Antonio Lioy

Candidate
Francesco Rollo

March 2026

Alla mia famiglia

Summary

The rapid advancement of quantum computing poses a critical long-term threat to current public-key cryptography, with cryptographically relevant quantum computers (CRQCs) potentially capable of breaking widely deployed cryptosystems by running Shor's algorithm, undermining the security of major Internet protocols. This threat has prompted urgent calls for transitioning to post-quantum cryptography (PQC), with NIST finalizing its first set of post-quantum standards in August 2024. However, given the unprecedented scope of the PQC transition, affecting all public-key algorithms, this transition presents significant technical and operational challenges. Migrating complex, widely deployed software ecosystems, such as web browsers and their cryptographic libraries, requires not only algorithm replacement but also architectural flexibility to adapt to evolving standards and implementations. This thesis addresses the challenge of transitioning Mozilla Firefox and its Network Security Services (NSS) cryptographic library toward post-quantum cryptography through a modular, crypto-agile framework. Conducted within the European QUBIP project, this work builds upon prior research that established initial support for PQ/T hybrid key exchange, extending the implementation to encompass post-quantum and hybrid digital signatures for comprehensive quantum-resistant TLS 1.3 connections. The methodology leverages NSS's native support of the `PKCS#11` standard to introduce post-quantum capabilities through a shallow-loadable module approach. Rather than directly embedding cryptographic implementations within NSS, an external `PKCS#11` software token named `qryptotoken` serves as a lightweight delegation layer, routing cryptographic operations to third-party post-quantum libraries through a novel Adapter-based architecture. This design achieves practical cryptographic agility by decoupling algorithm implementations from both Firefox/NSS and the module itself, enabling straightforward replacement and evaluation as standards evolve. The implementation encompasses support for NIST-standardized `ML-DSA` and `SLH-DSA` signature schemes across all parameter sets, as well as the composite `MLDSA65-Ed25519` algorithm. Integration testing through the QUBIP Quantum-secure Internet Browsing Pilot successfully demonstrated end-to-end PQ/T hybrid TLS 1.3 connections supporting both server-only and mutual authentication scenarios. Interoperability testing against external platforms including Open Quantum Safe (OQS) and Cloudflare Research's servers validated correct operation across heterogeneous implementations. These results provide empirical evidence that the proposed methodology enables practical post-quantum deployment while preserving backward compatibility and operational continuity. This work contributes to the broader objective of preparing the web browsing ecosystem for a quantum-secure future, demonstrating that modular, crypto-agile architectures can effectively manage the complexity of the post-quantum transition while maintaining the flexibility necessary to adapt to an evolving cryptographic landscape.

Acknowledgements

This thesis, and the research results it reports, would not have been possible without the guidance, support, and generosity of many people to whom I owe my deepest gratitude.

First and foremost, I would like to thank my thesis supervisor, Professor Antonio Lioy at Politecnico di Torino, for his guidance throughout this work and for his patience in reading and revising the drafts of this thesis. I'm also grateful for his lectures during my Master's programme, which played a pivotal role in shaping my professional path and fostering a genuine passion for the field, which ultimately led me to undertake this research.

I also want to thank Doctoral Researcher Nicola Tuveri at Tampere University, who brought me into the QUBIP project and gave this thesis a dimension it wouldn't otherwise have had. His technical expertise and genuine enthusiasm for the field were a constant source of inspiration. Getting to participate at the OpenSSL Conference and DevConf in Czech Republic was something I couldn't have arranged on my own, and those events genuinely changed my view of open-source and cryptographic communities, which I will carry with me throughout my career.

Finally, my heartfelt gratitude goes to my family and the people who stood closest to me throughout these years. Their constant support and encouragement have been my foundation at every step of this academic path.

Contents

1 Introduction 9

1.1 Motivation . . . 10
1.2 Research Objectives . . . 11
1.3 Research Questions . . . 12
1.4 Contributions . . . 12
1.5 Structure of the Thesis . . . 13

2 Background 14

2.1 Overview of Cryptography . . . 14
2.1.1 Terminology . . . 15
2.2 Symmetric Cryptography . . . 15
2.3 Hash Functions . . . 16
2.3.1 Message Authentication Code (MAC) . . . 17
2.4 Authenticated Encryption . . . 18
2.5 Public-Key (Asymmetric) Cryptography . . . 18
2.5.1 Mathematical Foundations of Public-Key Algorithms . . . 19
2.6 Key Exchange . . . 20
2.6.1 Perfect Forward Secrecy . . . 22
2.7 Digital Signatures . . . 22
2.8 Public-Key Infrastructure . . . 23
2.8.1 PKI Components . . . 24
2.8.2 Certificate Revocation . . . 24
2.8.3 Certificate Transparency (CT) . . . 25
2.8.4 Certificate Chains and Path Validation . . . 25
2.9 PKCS . . . 26
2.9.1 PKCS#11: Cryptoki API . . . 26
2.10 The Transport Layer Security (TLS) Protocol . . . 28
2.10.1 TLS 1.3 Handshake Protocol . . . 29
2.11 Quantum Computing . . . 31
2.11.1 Quantum Bit . . . 31
2.11.2 Multiple Qubits and Entangled States . . . 33

2.11.3 Quantum Gates and Quantum Circuits . . . 33
2.11.4 Quantum Parallelism and Interference . . . 34
2.11.5 The Threat of Shor's Algorithm . . . 35
2.11.6 Grover's Algorithm . . . 35
2.11.7 Current State and Challenges in Building Quantum Computers . . . 36
2.12 Post-Quantum Cryptography . . . 36
2.12.1 NIST's Security Levels . . . 39
2.12.2 From Harvest Now, Decrypt Later to Trust Now, Forge Later . . . 39
2.12.3 Post-Quantum Migration Planning of Web PKI . . . 41

3 Related Work 45
3.1 QUBIP Consortium Efforts . . . 45
3.1.1 Public Key Infrastructure . . . 45
3.1.2 Operating Systems Integration . . . 46
3.1.3 Aurora: A Shallow Provider for OpenSSL . . . 47
3.1.4 Prior Work and Contributions to Firefox's Crypto-Agility . . . 48
3.2 Building Blocks: Preliminaries . . . 49
3.2.1 Kryoptic: RedHat's PKCS#11 Software Token . . . 49
3.2.2 Mozilla Firefox . . . 51
3.2.3 Network Security Services (NSS) . . . 52

4 Methodology 54
4.1 QUBIP Module Overview . . . 54
4.2 Next Steps . . . 56
4.2.1 Support of PQ/T Hybrid Digital Signatures in TLS 1.3 . . . 57
4.2.2 Mutual PQ/T Hybrid Authentication in TLS 1.3 . . . 57
4.2.3 Enhancements to `qryptotoken` . . . 58
4.2.4 Integration and Interoperability Testing . . . 59

5 Results 61
5.1 Results of Integrating PQC Support in Firefox and NSS . . . 61
5.1.1 Supported PQ/Hybrid Algorithms . . . 62
5.1.2 Core Infrastructure Extensions . . . 62
5.1.3 PKCS#11 Integration . . . 63
5.1.4 TLS Protocol Integration . . . 64
5.1.5 Certificate Validation Infrastructure . . . 64
5.1.6 Post-Quantum Algorithm Policy Enforcement . . . 65
5.2 Results of Integrating PQC Support in `qryptotoken` . . . 66
5.2.1 Adapter Architecture . . . 66
5.2.2 PKCS#11 Integration . . . 68
5.2.3 Validation and Testing . . . 69
5.2.4 Targeted Post-Quantum and Hybrid Algorithms Supported . . . 70
5.3 Results of Integration and Interoperability Testing . . . 72
5.4 Challenges and Limitations . . . 76

6 Conclusion 79

6.1 Answering the Research Questions 79

6.2 Contributions and Future Directions 84

Bibliography 85

Chapter 1

Introduction

For decades, cryptography has been crucial in establishing information security in the digital era. Cryptography has spread across all technological domains, adequately addressing, in theory and in practice, security objectives such as confidentiality, integrity, authenticity, and non-repudiation. Security mechanisms that rely on cryptographic principles are an integral part of nearly every aspect of modern information systems, ensuring digital trust, data protection, and secure communication.

Classical cryptography consists of different primitives including encryption, key distribution, digital signatures, pseudo-random number generation, zero-knowledge proofs, and one-way functions [1, p. 15]. All these cryptographic primitives are built upon mathematical problems that are believed to be computationally infeasible for conventional computers to solve. In particular, public key (asymmetric) cryptography relies on *trapdoor one-way functions*, mathematical functions easy to compute in one direction, yet difficult to invert without extra information (called the *trapdoor information*). Sources of many trapdoor functions are the **integer factorization problem** (e.g., RSA cryptosystem), and the **discrete logarithm problem** (e.g., Diffie-Hellman key exchange and Elliptic Curve Cryptography), both of which have no known polynomial-time inverse solutions for conventional computers. In contrast, the security of symmetric key cryptography relies on the secrecy and unpredictability of a shared random key between two parties. An adversary attempting to compromise the system must either proceed with cryptanalysis[1] or determine the key by exhaustively searching the entire key space. When the key space is sufficiently large, such brute-force attacks become computationally infeasible within any practical time frame.

However, complexity theory lacks the necessary tools to definitively prove that efficient algorithms don't exist for certain problems. Therefore, decisions about problem hardness rely entirely on empirical evidence. Historically, the security of cryptographic schemes has been assessed under the assumption that adversaries have access to classical computers and are limited to randomized polynomial-time computations. In this model, cryptanalysts attempt to solve the hard mathematical problems upon which these schemes are built. Many cryptosystems have been examined for years without anyone finding more efficient attacks. Over time, the cryptographic community becomes increasingly confident that the best available attack methods are already known, or at least that practical attackers are unlikely to discover better approaches. A cryptographic scheme is therefore considered secure once it has successfully resisted extensive analysis under these classical computing assumptions [1, pp. 2, 15–16].

Nevertheless, the rapid progress of **quantum computing** research poses a significant threat to the classical assumptions underlying the security of current cryptography. Quantum computers operate on the principles of quantum mechanics, exploiting quantum phenomena at the microscopic level, which allows them to perform certain computations exponentially faster than their classical counterparts. Advancements in this field promise a revolutionary leap in computational

[1] Cryptanalysis refers to the process of analysing information systems in order to breach cryptographic security systems and gain access to the contents of encrypted messages, even if the cryptographic key is unknown.

power, with the potential to drive innovation and optimization across a wide range of industries. However, these same capabilities could be exploited by adversaries to compromise existing cryptographic algorithms. The emerging quantum computational paradigm introduces new hypotheses in which the intractability of the aforementioned "hard" problems may be at risk. One of the most significant hypotheses in this context, dates back to 1994 when Peter W. Shor proposed the first example of quantum cryptanalysis [2]. In his paper, Shor demonstrated that future quantum computers could factor integers and compute discrete logarithms in polynomial time, providing the first evidence that well-known and widely used public-key cryptosystems could potentially be broken. Two years later, in 1996, Lov K. Grover introduced a quantum algorithm for unstructured database search, which has important implications for symmetric cryptography [3]. Grover's algorithm can be applied to exhaustively search the key space of symmetric algorithms, achieving at least a quadratic speedup that effectively halves the security provided by a given key length. Currently, no quantum computer capable of performing such computations, commonly referred to as a **cryptographically relevant quantum computer (CRQC)**, exists. But, in future, a successful quantum attack by a CRQC would undoubtedly have a devastating impact on cryptography, severely affecting global digital infrastructure and daily life.

1.1 Motivation

Internet protocols that are widely used today rely on classical cryptographic algorithms to ensure secure communication. In most cases, cryptographic keys are first exchanged using public-key cryptography (e.g., RSA or ECDH), and then are used to encrypt subsequent messages with symmetric algorithms (e.g., AES). Some examples include Transport Layer Security (TLS), Internet Protocol Security (IPsec), and Secure Shell (SSH). Within this context, the threat posed by quantum computing presents a major long-term security concern. While the exact timeline for the development of large-scale, cryptographically relevant quantum computers is still unknown, projections suggest that such capabilities may emerge earlier than expected, even in less than a decade. This scenario exposes current information systems to the **Harvest Now Decrypt Later (HNDL)** attack strategy, whereby any adversary can collect and store encrypted data today, with the intention of decrypting it once a cryptographically relevant quantum computer becomes available, assuming the collected data remains relevant in some years from now.

In response, organizations like the National Institute of Standards and Technology (NIST) have demanded an urgent transition to **post-quantum cryptography (PQC)**. NIST announced the PQC Standardization Process with a public call for algorithms submissions in December 2016 and, as of August 2024, has finalized its principal set of PQC algorithms designed to withstand cyberattacks from a quantum computer [4]. New standards include the Module-Lattice-Based Key-Encapsulation Mechanism (FIPS 203 [5]), the Module-Lattice-Based Digital Signature Algorithm (FIPS 204 [6]), and the Stateless Hash-Based Signature Algorithm (FIPS 205 [7]). In parallel, the U.S. National Security Agency (NSA) has updated its Commercial National Security Algorithm Suite (CNSA) to version 2.0, mandating the adoption of quantum-resistant algorithms for their systems by 2035 to ensure long-term security [8].

History has taught us that cryptographic transitions are complicated and can take several years or even decades to reach an end. For example, when DES was discovered to be insecure in the late 90s, the industry came up with Triple-DES as a temporary solution, but it took over a decade for AES (standardized in 2001) to reach widespread adoption. Both AES and Triple-DES were in use for many years until Triple-DES was finally prohibited in 2024 [9]. This slow transition is dictated by key size limitations, performance concerns, and the need for backward compatibility. The PQC transition intensifies these issues, because unlike previous transitions which aimed at a specific algorithm, PQC involves all the public-key algorithms, hence the largest cryptographic migration in the history. The slow pace of this process is also critical for complex systems like Public Key Infrastructures (PKIs), which ensure the authenticity and trust of all secure communications across the Internet and beyond. Furthermore, the rapid and unpredictable pace of advancements in quantum computing means that new cryptographic solutions may be required on a recurring basis, increasing the risk of not being ready to secure the systems in time.

That said, to limit this occurrence, the cryptographic community has increasingly embraced

the principle of **cryptographic agility**, also referred to as *crypto-agility*. Despite its frequent use, the term remains quite controversial for its ambiguity. Over the years, researchers have explored various interpretations of cryptographic agility, acknowledging the difficulty of providing a unified definition due to the diversity of systems and contexts to which it applies. Recently, the NIST published a white paper outlining different considerations for achieving cryptographic agility [10], and in this document a more general definition that captures the essence of the concept is given:

> *"Cryptographic (crypto) agility describes the capabilities needed to replace and adapt cryptographic algorithms for protocols, applications, software, hardware, firmware, and infrastructures while preserving security and ongoing operations."* [10, p. 2]

Later in the document [10, p. 2], there are several more targeted definitions depending on the technical context explored. Particularly relevant in the present discussion is the definition related to communication protocols:

> *"...In a protocol, crypto agility is the ability to maintain interoperability when introducing new cryptographic algorithms and preventing the use of weak algorithms."*

As standardization bodies work to transition Internet protocols to post-quantum cryptography, including ongoing efforts within the IETF's LAMPS[2] working group, this thesis aims to explore new modular and flexible approaches for integrating post-quantum cryptography, guided by the principle of cryptographic agility. Secure digital communication relies heavily on long-established protocols such as Transport Layer Security (TLS), which is fundamental to most Internet connections and is embedded in countless applications, including web browsers. **Mozilla Firefox** provides an ideal platform for experimental PQC integration due to its open-source nature and reliance on the **Network Security Services (NSS)**, an open-source cryptographic library also used by many other applications, allowing the work to be both accessible and practical to the community. NSS serves as Firefox's cryptographic backbone providing support for the TLS protocol, Public Key Infrastructure X.509 (PKIX) certificate chain validation, and other core cryptographic standards. Transitioning NSS to support the latest standardized post-quantum algorithms would not only strengthen Firefox's own security posture but also enable a reusable and scalable pathway toward quantum-resistant security across a wide range of dependent software.

1.2 Research Objectives

The main research goal of this thesis is to develop a practical and adaptable approach for transitioning Mozilla Firefox and its cryptographic library, NSS, toward post-quantum cryptography. Achieving this goal requires addressing the technical and architectural challenges involved in introducing the newly standardized post-quantum algorithms into such a foundational component.

The NSS codebase is large and complex, which makes a direct transition to PQC far from trivial. However, NSS was designed around the `PKCS#11` standard, a widely adopted interface that defines a platform-independent API for communicating with cryptographic tokens and hardware security modules (HSMs). This modular design enables NSS to dynamically load cryptographic tokens capable of externally handling cryptographic operations requested by any application built on top of it, such as Firefox.

Building on this capability, this thesis focuses on designing and implementing a modular and agile, `PKCS#11`-based approach to introduce post-quantum capabilities into NSS and Firefox's

[2] IETF LAMPS (Limited Additional Mechanisms for PKIX and SMIME) is a working group within the Internet Engineering Task Force (IETF), the organization that develops and maintains internet standards. LAMPS is chartered to update X.509 certificate and email security documents originally produced by the now-closed PKIX and S/MIME working groups, focusing on updates with known constituencies interested in real deployment. The group is currently working on integrating post-quantum cryptography (PQC) algorithms into PKIX certificates and Cryptographic Message Syntax (CMS) to prepare for quantum computing threats [11].

TLS stack, avoiding breaking changes and minimizing the effort required to adapt to emerging PQC solutions. As the post-quantum cryptographic landscape continues to evolve, a loadable and agile module for NSS, hereafter referred to as the **shallow-loadable module**, can provide practical means to test, replace, and upgrade PQC algorithms with relative ease, enabling a flexible, incremental, and future-proof transition toward a quantum-secure Internet.

A preliminary proof of concept in this research direction was previously explored by D. Luoma within the same research environment in which this thesis was developed [12]. A detailed discussion of that prior work, upon which the present study builds, is provided in Chapter 3. The present work seeks to extend cryptographic agility within NSS by integrating support for post-quantum signature schemes and hybrid approaches currently under active investigation by the research and standardization community. In addition, this thesis focuses on strengthening Firefox's interoperability with modern Public Key Infrastructures (PKIs) from a post-quantum perspective. Ensuring interoperability with evolving PKI components, such as certification authorities, trust anchors, and hybrid certificate chains, is critical, as Firefox relies on these mechanisms for authentication and secure communication. Without such adaptations, Firefox would risk losing interoperability as the Internet ecosystem transitions toward quantum-resistant standards.

1.3 Research Questions

To accomplish the objectives introduced earlier, this thesis defines a set of research questions, which covers the full scope of this work. Accordingly, the research is guided by the following questions:

- **RQ1:** What is the methodology required to effectively guide the transition of Mozilla Firefox and its cryptographic library, NSS, toward post-quantum cryptography to enable a quantum-secure Internet browsing?
- **RQ2:** How can the proposed shallow-loadable module approach enhance cryptographic agility and facilitate the integration of post-quantum algorithms within NSS?
- **RQ3:** What are the critical challenges in adapting PKI components to support post-quantum algorithms, ensuring interoperability within Firefox and NSS?

These questions shape the scope and direction of this thesis. Their resolution provides a solid basis for the design, implementation, and evaluation presented in the following chapters.

1.4 Contributions

The work presented in this thesis is conducted within the **QUBIP project** [13], which is funded by the European Union under the Horizon Europe framework programme [Grant Agreement No. 101119746].

QUBIP (**Qu**antum oriented update to **B**rowsers and **I**nfrastructure for the **PQ** transition) is designed to contribute to the EU transition to PQC with the aim of streamlining the process and creating a replicable transition model. In particular, this work contributes, among others, to the third and fifth strategic objectives of the project, which are reported below:

3. *"Explore the transition of OpenSSL and NSS cryptographic libraries to PQC through loadable modules with the main goal of plugging (i) PQ algorithms and schemes in the existing software ecosystem and (ii) PQ hardware implementations transparently."*

5. *"Experiment with the transition of Firefox browser toward a post-quantum security state through adoption of PQ/T Hybrid TLS 1.3 for key-exchange and authentication, and PQ zero-knowledge Verifiable Credentials (PQ zero-knowledge VCs) for authentication and authorization."*

This thesis was carried out in 2025 as part of the author's employment at the Network and Information Security Group (NISEC) within the Unit of Computing Sciences at **Tampere University**, while being enrolled at **Politecnico di Torino** for the Master's Degree in Cybersecurity. Both institutions are partners of the QUBIP project. The author carried out this work under the joint supervision of Professor Antonio Lioy, the thesis advisor at Politecnico di Torino, and Doctoral Researcher Nicola Tuveri, supervisor at Tampere University. The author also collaborated closely with his colleagues Akif Mehmood, Alex Shaindlin, and Nouman Ali Khan.

1.5 Structure of the Thesis

This thesis is organized into the following distinct chapters. Chapter 2 establishes the theoretical background, covering fundamental concepts of symmetric and asymmetric cryptography, quantum computing principles, post-quantum cryptography, and the migration planning for Web Public Key Infrastructures (PKIs). Chapter 3 reviews related work within the QUBIP consortium and introduces the core building blocks that underpin this research. Chapter 4 presents the methodology adopted for transitioning Firefox and NSS toward post-quantum cryptography through a shallow-loadable module approach. Chapter 5 details the implementation results, including modifications to Firefox/NSS, the Adapter-based architecture within `qryptotoken`, and integration and interoperability testing outcomes, while also examining the limitations encountered during development and deployment. Finally, Chapter 6 concludes by answering the research questions, summarizing contributions, and identifying directions for future work.

Chapter 2

Background

To address the topic of this thesis effectively, it is first necessary to establish a solid theoretical foundation. This chapter provides an overview of the essential background information, focusing on the aspects most pertinent to this study. The discussion encompasses fundamental concepts of symmetric and asymmetric cryptography, including key exchange protocols, digital signatures, and Public Key Infrastructure (PKI), concluding with an analysis of the TLS 1.3 protocol. Crucially, the chapter introduces quantum computing fundamentals, the field of post-quantum cryptography, and the intersection of quantum algorithms with current cryptography. For simplicity and to maintain focus on concepts relevant to this study, detailed mathematical derivations and specific algorithm implementations are intentionally omitted. Readers seeking comprehensive technical treatment of these topics are strongly encouraged to consult the literature referenced throughout the chapter.

2.1 Overview of Cryptography

Understanding cryptography as it exists today requires a brief introduction to the fundamental issues of information security. As discussed in the Handbook of Applied Cryptography by Menezes, van Oorschot, and Vanstone [14, p. 2-4], in any system where information is exchanged or stored, all parties involved must have confidence that certain objectives associated with information security are met. Historically, information security relied on physical protections to prevent unauthorized access or tampering.

Today, the majority of information is stored and transmitted in a electronic form via telecommunication systems, and the ability to copy, alter, or intercept information has increased dramatically and with unprecedented ease. This shift has made necessary the adoption of new security methods that ensure the core objectives of information security, such as confidentiality, integrity, and authenticity, independent of the physical medium, such that they rely solely on the digital information itself. Modern cryptography provides these technical means. Until late twentieth century, cryptography was largely regarded as an art: designing secure codes or breaking existing ones depended heavily on intuition and creativity. However, today, the field has evolved into a rigorous science, grounded in mathematics and formal analysis.

Menezes et al. describe modern cryptography as *"[...] the study of mathematical techniques related to aspects of information security such as confidentiality, data integrity, entity authentication, and data origin authentication"* [14, p. 4]. The primary goals of cryptography can be organized around four main properties:

1. **Confidentiality:** ensures that information is accessible only to authorized parties. Also referred to as secrecy or privacy, confidentiality can be enforced through mathematical algorithms that render data unintelligible (i.e., encryption).

2. **Data integrity:** provides assurance that information has not been altered by unauthorized parties. This includes detecting any unauthorized insertion, deletion, or modification of data. Ensuring data integrity is essential for maintaining the trustworthiness of information.

3. **Authentication:** refers to the verification of the identity of an entity or the origin of information. It applies both to entities and to the data being exchanged, enabling parties to confirm the identity of the communicants and the integrity of transmitted information. Authentication is typically divided into *entity authentication* and *data origin authentication*, the latter implicitly provides data integrity.

4. **Non-repudiation:** prevents an entity from denying previously performed actions or commitments. Mechanisms like digital signatures provide evidence linking an action, such as a transaction, to its originator. However, true non-repudiation requires a trusted-third party to verify the authenticity of digital signature and resolve potential disputes.

By addressing these four properties, cryptography helps to prevent and detect malicious activity in digital systems, providing information security.

2.1.1 Terminology

A message to be protected is referred to as **plaintext**. The process of disguising a message so as to conceal its content is called **encryption**, and the resulting encrypted data is known as **ciphertext**. The reverse process, by which the ciphertext is transformed back into a readable plaintext, is called **decryption**.

The mathematical function used to perform encryption and decryption is called **cryptographic algorithm**, or **cipher**. In most systems, the cipher defines two closely related functions, one for encryption and another for decryption, typically parameterized by a **key**. The key represents the secret piece of information that determines the specific transformation applied by the cipher. The key may take on any one of a large number of possible values. The set of all possible key values is known as the **key space**.

Some algorithms use different keys for encryption and decryption, but in all such systems, the security depends entirely on the key (or keys), not on the secrecy of the algorithm itself (Kerckhoffs's principle). This principle implies that the algorithm can be publicly disclosed and subjected to analysis without compromising its security [15, p. 1-4]. Relying on the secrecy of the algorithm, a concept known as *security through obscurity (STO)*, is considered poor practice because it provides only a false sense of protection: once the algorithm is discovered or reverse-engineered, the entire system is compromised.

A **cryptosystem** is a general term referring to a collection of cryptographic primitives designed to provide various information security services. Each primitive provides a specific security property or function. While the term cryptosystem is most commonly associated with primitives that ensure confidentiality (i.e., encryption), it can also refer to systems that achieve other security objectives described earlier [14, p. 15].

Cryptographic algorithms are generally divided into two categories: *symmetric-key* and *public-key* algorithms. The following sections provide a brief introduction to each type.

2.2 Symmetric Cryptography

Symmetric cryptographic schemes, also referred to as *symmetric-key*, *secret-key*, or *single-key* algorithms, use the **same secret key** for both encryption and decryption of a message. These algorithms require that the sender and the receiver agree on a key in order to communicate securely. The key may be established prior to communication, out of band (OOB), or negotiated through a key-exchange protocol, which will be discussed later in Section 2.6.

The defining characteristic of symmetric primitives, in contrast to asymmetric ones, is the absence of a "public" operation. To achieve confidentiality using a symmetric algorithm, both

parties must use the same key as input to the cryptographic functions. The security of a symmetric algorithm relies entirely on the secrecy of this key. If the key is disclosed, anyone can both encrypt and decrypt messages.

Symmetric algorithms can be further divided into **block ciphers** and **stream ciphers**. A *block cipher* is an encryption scheme which breaks up the plaintext message to be transmitted into blocks of a fixed length, and encrypts one block at a time [14, p. 16]. Because a block cipher works on fixed-length blocks, it is suited for situations where data can be segmented into chunks of this size (or padded to that size). When plaintext messages exceed one block, various modes of operation for block ciphers are used to securely process long messages. A widely used example of a block cipher is the **Advanced Encryption Standard (AES)**, which operates on a block size of 128-bit and supports key sizes of 128, 192, or 256 bits. AES is efficient in both software and hardware, and is commonly used with modes such as *Cipher Block Chaining (CBC)* or *Galois Counter Mode (GCM)* for secure data transmission and storage.

On the other hand, *stream ciphers* encrypt data one unit at a time, typically a bit or a byte, using a keystream that may be truly random or generated deterministically from a seed by a *keystream generator*. The keystream is combined with the plaintext, usually via XOR, to produce the ciphertext. Stream ciphers are suitable in contexts where memory and buffering of data are limited or latency is critical [14, p. 20-21]. They tend to have little or no error-propagation: if one ciphertext unit is corrupted, only the corresponding plaintext unit is wrong (unlike some old block-modes where subsequent blocks get corrupted).

Common examples include the **one-time pad (OTP)**, which is theoretically unbreakable when the key is truly random, of the same size of the message and used only once. However, this cipher is impractical for most modern systems due to key distribution challenges. More practical stream ciphers used today are **Salsa20** and **ChaCha20**, which overcome these limitations by using pseudorandom keystream generators that provide strong security while offering high performances.

2.3 Hash Functions

Cryptographic hash functions play a fundamental role in modern cryptography. A hash function takes a message as input and produces a fixed-length output known as the *hash-value* or simply *hash*. The hash value serves as a compact and unique representation of the input message, often referred to as a *message digest*, and can be treated as a digital fingerprint of the original data. Hash functions are commonly used to ensure data integrity in conjuction with digital signatures, following the so-called *hash-and-sign* paradigm, which will be examined in Section 2.7.

Several standardized hash algorithms are widely used today. Among these, relevant are the SHA-2 family (e.g., `SHA-256` and `SHA-512`) and the more recent SHA-3 family based on the *Keccak* algorithm, standardized by NIST in FIPS 202 [16]. The SHA-3 family also includes extendable-output functions (XOFs) such as `SHAKE128`, `SHAKE192`, and `SHAKE256`.

At the highest level, hash functions can be divided into two main categories: *unkeyed* and *keyed* hash functions. An *unkeyed hash function* accepts a single input parameter (a message), while a *keyed hash function* takes two distinct inputs: a message and a secret key. According to [14, Chapter 9], a hash function can be informally defined as follows.

Definition 2.3.1. A hash function h is a deterministic function that satisfies at least the following two properties:

1. **Compression:** h maps an input x of arbitrary finite length to an output $h(x)$ of fixed bit length n.
2. **Ease of computation:** for a given input x, the value $h(x)$ is easy to compute (polynomial in time).

For an unkeyed hash function h with inputs x, x' and outputs y, y', three additional security properties are typically required:

1. **Preimage resistance:** Given a hash output y, it should be computationally infeasible to find any input x' such that $h(x') = y$. This ensures that, without prior knowledge of x, an adversary cannot efficiently invert the hash function. This property is typically referred to as *one-wayness*.

2. **Second-preimage resistance:** Given an input x, it should be computationally infeasible to find another distinct input $x' \neq x$ such that $h(x) = h(x')$. This property prevents an adversary from finding another message that produces the same hash as a given message. This property is typically referred to as *weak collision resistance*.

3. **Collision resistance:** It should be computationally infeasible to find any two distinct inputs x and x' such that $h(x) = h(x')$. In this case, the adversary is free to choose both x and x'. This property is typically referred to as *strong collision resistance*.

Definition 2.3.2. A one-way function is a function f such that, for every x in the domain of f, it is easy to compute $f(x)$, but for essentially all y in the range of f, it is computationally infeasible to find any x such that $f(x) = y$.

Intuitively, cryptographic hash functions can be viewed as practical constructions of one-way functions that also ensure collision and second-preimage resistance, making them indispensable in data integrity mechanisms. Unkeyed hash functions can be used to provide *data integrity*, but they cannot provide *data origin authentication*, since anyone can generate an hash. To provide both integrity and authenticity, keyed hash functions are used. These are commonly known as *Message Authentication Codes* (MACs).

2.3.1 Message Authentication Code (MAC)

A Message Authentication Code (MAC), also called *keyed-digest*, is a category of keyed hash functions that allows to generate an *authentication tag* derived from a shared secret key and a message. The recipient, using the same key, can recompute the tag and verify that it matches the one attached to the message, ensuring that the message is authentic and has not been altered in transit. Following [14, Chapter 9], a MAC is defined as follows.

Definition 2.3.3. A message authentication code (MAC) algorithm is a family of functions h_k, each parameterized by a secret key k, and satisfies the following properties:

1. **Ease of computation:** For a known function h_k, given the key k and an input message x, the value $h_k(x)$ can be efficiently computed. The resulting value $h_k(x)$ is referred to as the *MAC*.

2. **Compression:** Each function h_k maps an input x of arbitrary finite length to an output $h_k(x)$ of fixed bit length n.

3. **Computation resistance:** Given the description of the function family h and zero or more known message-MAC pairs $(x_i, h_k(x_i))$, it is computationally infeasible for an adversary (who does not know k) to compute a valid pair $(x, h_k(x))$ for any new input $x \neq x_i$. This includes producing a new message x that yields the same MAC value as an existing one.

The most common instantiation of a MAC algorithm is based on a block cipher operating in Cipher Block Chaining (CBC) mode, called CBC-MAC.

Another very common MAC construction used in cryptographic protocols nowadays is the **Hashed Message Authentication Code (HMAC)**. The HMAC construction uses a cryptographic hash function as its underlying primitive, combines it with a secret key, and applies two rounds of hashing with dedicated inner and outer padding. This design offers strong security properties even if the underlying hash function is subject to certain weaknesses, such as collision attacks.

2.4 Authenticated Encryption

Authenticated encryption (AE) is a cryptographic approach that simultaneously provides confidentiality, data integrity, and data origin authentication.

A straightforward, yet conceptually transparent scheme, standardized by ISO/IEC 19772:2009 [17], is *encrypt-then-MAC*. According to this method, the plaintext is first encrypted using a symmetric cipher, and subsequently a Message Authentication Code (MAC) is computed over the ciphertext. This sequence ensures that any alteration of the ciphertext can be detected before decryption, providing robust protection against active adversaries.

A subset of AE algorithms includes *Authenticated Encryption with Associated Data (AEAD)* schemes, which distinguish between data requiring confidentiality and data requiring only integrity and authentication, known as *associated data*. An example is the Galois Counter Mode (GCM), which encrypts data using the Counter Mode (CTR) followed by the computation of a MAC, allowing the authentication of unencrypted associated data, such as headers or metadata, without including it in the ciphertext. This property is particularly useful in network protocols, where certain fields (e.g., packet headers) must remain visible to intermediaries while still being protected against tampering.

AEAD schemes are widely adopted in practice. A prominent example is AES-GCM [18], or the well-known ChaCha20-Poly1305 [19], which integrates the ChaCha20 stream cipher with the Poly1305 MAC. Regarding the latter, both components were designed by Daniel J. Bernstein to achieve high performance in software environments, particularly where hardware acceleration for AES-GCM is unavailable.

Modern cryptographic protocols, such as TLS 1.3, mandate the use of AEAD ciphersuites to simultaneously ensure confidentiality, integrity, and authenticity of transmitted application data. The two AEAD algorithms currently standardized and negotiated in TLS 1.3 are AES-GCM [20] and ChaCha20-Poly1305 [19].

2.5 Public-Key (Asymmetric) Cryptography

The terms *public-key cryptography* and *asymmetric cryptography* denote the same thing and are used interchangeably. Public-key cryptography was first introduced by Whitfield Diffie, Martin Hellman, and Ralph Merkle in 1976. Since their contribution, many public-key algorithms have been proposed, but only few of them are both secure and practical, and widely used today.

As the name implies, asymmetric algorithms introduce an asymmetry in key usage, unlike symmetric algorithms where both parties share the same secret key and perform identical operations. In asymmetric cryptography, two mathematically related keys are used: a **public key** and a **private key**. The public key is widely known and can be freely distributed, while the private key must remain secret. Operations performed with one key can only be reversed using the corresponding counterpart. Most importantly, knowledge of the public key alone does not give an adversary the ability to deduce the private key or perform the private operation.

For example, in a basic public-key encryption scheme, when a sender wishes to securely share a message with a recipient, the sender obtains the recipient's public key and uses it to encrypt the message. The recipient then uses their corresponding private key to decrypt the message and recover the original plaintext.

A primary advantage of public-key systems is that distributing public keys is generally easier than distributing secret keys securely, as required by symmetric-key systems. Sharing a public key can occur over insecure channels, whereas symmetric-key systems require a secure channel to exchange the shared secret key in advance. Another problem solved by public-key cryptography is the **key distribution problem**. In systems based on symmetric-key cryptography with n participants, each pair of users requires a unique shared key, resulting in a total of $\frac{n \cdot (n-1)}{2}$ distinct keys. Managing and securely distributing such a number of keys quickly becomes impractical as the number of participants grows. In contrast, asymmetric cryptography requires only n key pairs, to allow any two participants to communicate securely.

There are several public-key algorithms, some of them are suitable for encryption, others for key exchange or digital signatures. Since public-key encryption algorithms are substantially slower than symmetric-key encryption algorithms, they are commonly used for securely exchanging keys that are subsequently used for bulk data encryption with faster symmetric ciphers [14, p. 283].

The most prominent asymmetric primitives are key exchange and digital signature algorithms, which will be described in the following sections.

2.5.1 Mathematical Foundations of Public-Key Algorithms

Public-key cryptography is grounded in mathematical problems that are believed to be computationally infeasible to solve with conventional computers. In practice, there are three major families of public-key algorithms that can be classified based on their underlying computational problem [21, p. 155]. The mathematical details of these algorithms will not be examined here as they fall out of the scope of this thesis. Instead, this section will focus on describing and formally defining the underlying computational problems on which these cryptographic schemes are based.

Integer-Factorization Schemes

Several public-key cryptosystems rely on the intractability of the **integer factorization problem**. The most prominent example is the RSA cryptosystem, named after its inventors R. Rivest, A. Shamir, and L. Adleman. RSA may be used both for encryption and digital signatures, and its security is based on the assumption that no efficient algorithm exists to factor large integers (typically 2048 bits or more).

The underlying problem can be described as follows. Given a large integer

$$n = p \cdot q \tag{2.1}$$

which is the product of two large prime numbers p and q of approximately equal bit length, it is computationally infeasible to recover p and q given only the value of n. Formally:

Definition 2.5.1. Given a positive integer $n \in \mathbb{N}$, find its prime factorization; that is, determine the set of distinct prime numbers $p_1, p_2, \ldots, p_k$ and corresponding positive integers $e_1, e_2, \ldots, e_k$ such that

$$n = p_1^{e_1} p_2^{e_2} \cdots p_k^{e_k}, \tag{2.2}$$

where each p_i is a prime number, $e_i \geq 1$, and the p_i are pairwise distinct [14, p. 89].

In other words, the goal is to decompose a composite number n into its prime factors. While this is computationally easy for small integers, it becomes infeasible for large values of n, particularly when n is the product of two large primes of roughly equal size.

Discrete Logarithm Schemes

There are several algorithms which are based on what is known as the **discrete logarithm problem** in finite fields. A partial list of these includes Diffie-Hellman key agreement, ElGamal encryption, and the ElGamal signature scheme or the Digital Signature Algorithm (DSA). The underlying problem is defined as follows.

Definition 2.5.2. Given the finite cyclic group $\mathbb{Z}_p^*$ of order $p-1$, let $\alpha \in \mathbb{Z}_p^*$ be a primitive element and $\beta \in \mathbb{Z}_p^*$. The discrete logarithm problem (DLP) is defined as the task of determining the integer x, where $1 \leq x \leq p-1$, such that:

$$\alpha^x \equiv \beta \pmod{p}. \tag{2.3}$$

This integer x is called the *discrete logarithm* of β to the base α, and can be written formally as:

$$x = \log_\alpha \beta \pmod{p}. \tag{2.4}$$

Computing discrete logarithms modulo a prime is computationally hard when the parameters are sufficiently large [21, p. 217].

Elliptic Curve (EC) Schemes

Elliptic Curve Cryptography (ECC) generalizes the discrete logarithm problem to the group of points on an elliptic curve over a finite field. The **elliptic curve discrete logarithm problem (ECDLP)** is defined as follows.

Definition 2.5.3. Let E be an elliptic curve defined over a finite field $\mathbb{F}_q$, and let $P \in E(\mathbb{F}_q)$ be a point of order n. Given another point $Q \in \langle P \rangle$ (that is, Q lies in the subgroup generated by P), the ECDLP is the problem of finding the integer $\ell \in [0, n-1]$ such that

$$Q = \ell P. \tag{2.5}$$

The integer ℓ is called the *discrete logarithm* of Q to the base P, denoted

$$\ell = \log_P Q. \tag{2.6}$$

In this setting, *point addition* and *scalar multiplication* on the elliptic curve are well-defined group operations. Computing $Q = \ell P$ (that is, performing scalar multiplication) is computationally efficient, even for large ℓ. However, the inverse operation, determining ℓ given P and Q, is believed to be computationally infeasible for properly chosen elliptic curve parameters.

The security of widely deployed elliptic curve schemes such as Elliptic Curve Diffie–Hellman (ECDH) and the Elliptic Curve Digital Signature Algorithm (ECDSA) directly relies on the hardness of the ECDLP [22, p. 153-155].

Compared to classical discrete logarithm systems (e.g., in modular arithmetic), elliptic curve cryptosystems provide *equivalent security with much smaller key sizes*. For instance, a 256-bit elliptic curve key has the equivalent security of a 3072-bit RSA key, which results in a significant efficiency and performance gain.

Security Level

The presented public-key algorithm families essentially rely on number-theoretic functions and require arithmetic with long operands and keys. The longer the key, the more secure the algorithm. The **security level** of an algorithm (in bits) corresponds to the computational effort needed for the best known attack: a security level of n bits requires roughly 2^n steps. The following table provides an overview of the public-key lengths for different security levels, given a specific algorithm family.

Algorithm family	Typical schemes	128-bit	192-bit	256-bit
Integer factorization	RSA	3072 bits	7680 bits	15360 bits
Discrete logarithm	DH, DSA, ElGamal	3072 bits	7680 bits	15360 bits
Elliptic curves	ECDH, ECDSA	256 bits	384 bits	512 bits

Table 2.1. Public-key key lengths for different security levels [23]

2.6 Key Exchange

Key exchange protocols enable two parties to securely share a secret, typically for subsequent use as symmetric keys for a variety of cryptographic purposes including encryption or message authentication. Key exchange may be further divided into *key transport* and *key agreement*.

Definition 2.6.1. A key transport protocol is a key exchange technique in which one party generates a secret value and securely transfers it to another.

In key transport protocols based on public-key encryption, one party chooses a symmetric key and transmits it to the other using that party's public encryption key. This approach provides key authentication to the originator, as only the intended recipient possesses the corresponding private key required for decryption. However, the originator does not obtain entity authentication, and the recipient, in turn, does not obtain source authentication. These assurances can be achieved with the use of digital signatures and public-key certificates. More information can be found in [14, Chapter 12].

However, the main focus of this section will be shifted on interactive key agreement protocols, which are relevant to widely used cryptographic protocols like TLS, discussed in later sections.

Definition 2.6.2. A key agreement protocol is a key exchange mechanism in which two parties jointly establish a shared secret as a function of information contributed by, or associated with, each of them, in such a way that neither party can unilaterally determine or predict the resulting value in advance.

The **Diffie-Hellman (DH) key agreement** is a fundamental protocol providing unauthenticated key agreement. The Diffie-Hellman key agreement protocol provided the first practical solution to the key distribution problem. It enables two parties, who have never met beforehand and share no prior keying material, to establish a shared secret over an insecure communication channel. The security of the protocol relies on the computational intractability of the discrete logarithm problem (Section 2.5.1).

Protocol description:

1. **One-time setup:** A large prime p and a generator α of the multiplicative group $\mathbb{Z}_p^*$ (with $2 \leq \alpha \leq p-2$) are selected and made public.
2. **Protocol messages:**

$$A \to B: \ \alpha^x \bmod p \tag{2.7}$$
$$A \leftarrow B: \ \alpha^y \bmod p \tag{2.8}$$

3. **Protocol actions:** Each time a shared key is required, the following steps are executed:
   (a) A chooses a random private value x, where $1 \leq x \leq p-2$, and sends B the message in (2.7).
   (b) B chooses a random private value y, where $1 \leq y \leq p-2$, and sends A the message in (2.8).
   (c) Upon receiving α^x, B computes the shared secret as

$$K_{BA} = (\alpha^x)^y \bmod p.$$

   (d) Upon receiving α^y, A computes the shared secret as

$$K_{AB} = (\alpha^y)^x \bmod p.$$

Both parties thus obtain the same shared key $K = K_{BA} = K_{BA}$:

$$K = \alpha^{xy} \bmod p.$$

This shared value can subsequently be used as a symmetric session key for secure communication.

The protocol described above is the classical Diffie-Hellman key exchange protocol, defined over the multiplicative group $\mathbb{Z}_p^*$, where p is a prime number. This protocol can be generalized to other algebraic structures, most notably to the group of points on an elliptic curve, formulating the **Elliptic Curve Diffie-Hellman (ECDH)** protocol.

The basic Diffie–Hellman protocol provides protection in the form of secrecy of the derived key against passive adversaries (i.e., eavesdroppers). However, it offers no defense against active adversaries who can intercept, modify, or inject messages into the communication (*man-in-the-middle*

attack). Consequently, neither party can be certain of the source identity of the incoming message or the identity of the party which may know the resulting key, respectively *entity authentication* and *key authentication*.

To address these limitations, **Authenticated Key Exchange (AKE)** is used. In this particular case, it's referred to as **Authenticated Key Agreement (AKA)** and it's achieved by integrating mechanism previously mentioned: digital signatures and public-key certificates, which will be explored in detail in the following sections. In a typical implementation of authenticated DH, each party digitally signs their respective DH public key share before sending it. For example, party A would generate its public value $\alpha^x \pmod p$, compute a digital signature over this value using its long-term static private key, and then send both α^x and the signature to party B. Party B would use A's public key (obtained from a trusted public-key certificate) to verify the signature, thereby implictly authenticating that the value α^x genuinely originated from A and was not tampered with.

2.6.1 Perfect Forward Secrecy

Key exchange protocols face a significant vulnerability when used for long-term secure communication. The compromise of a single static long-term private key risks confidentiality of all past communication sessions. An attacker can patiently record key exchanges and all encrypted traffic protected by the resulting session keys. If the static long-term private key is eventually compromised, the attacker can retroactively decrypt (in case of *key transport*), or recompute (in case of *key agreement*), all the historical session keys and decrypt the corresponding encrypted data.

Definition 2.6.3. Perfect Forward Secrecy (PFS) is a critical security property in key agreement protocols that ensures the confidentiality of past session keys even if the long-term private key of one or both parties is later compromised.

Example protocols that achieve PFS are **Ephemeral Diffie-Hellman (DHE)** or **Elliptic Curve Ephemeral Diffie-Hellman (ECDHE)**. Taking these in consideration, to achieve PFS, for each execution of the protocol, short-term **ephemeral key exchange keys** are computed, and the static long-term private keys are used only to authenticate the exchange, via digital signatures, and verify the identity of the communicating peers. Immediately after the shared session secret is computed, the **ephemeral key-exchange keys are thrown away**. If a short-term private key is compromised is an issue only for the confidentiality of the current encrypted traffic, instead if a long-term private key is compromised is an issue only for the authentication since they are used only for signing. This ensures the secrecy of past communications is "forwarded" in time.

2.7 Digital Signatures

Digital signatures are one of the most relevant cryptographic primitives widely used today. They provide **data-origin authentication** and **data integrity**. One of the most significant applications of digital signatures is the certification of public keys in large networks. Certification is a means for a trusted third party (TTP) to bind the identity of a user to a public key. This concept forms the basis of a more complex framework called Public-Key Infrastructure (PKI), that will be explored in Section 2.8.

While symmetric cryptography can offer message authentication and integrity through message authentication codes (MACs), such methods require the communicating parties to share a secret key, so it's not possible to say which of the two parties has computed the keyed-digest. In contrast, digital signatures use asymmetric cryptography, where each user possesses a pair of cryptographic keys: a private key (used to sign) and a corresponding public key (used to verify).

A digital signature scheme is a cryptographic primitive consisting of three algorithms [14, Chapter 11]:

1. **Key generation** (KeyGen): produces a key pair (pk, sk), where sk is the private signing key and pk is the public verification key.

2. **Signing** (Sign): given a private key sk and a message m, produces a signature $\sigma = \mathsf{Sign}(sk, m)$.
3. **Verification** (Verify): given a public key pk, a message m, and a signature σ, returns `true` (valid) or `false` (invalid).

For correctness, the following must hold:

$$\mathsf{Verify}(pk, m, \mathsf{Sign}(sk, m)) = \texttt{true}.$$

A secure digital signature scheme must fulfill the following essential properties:

- **Authenticity:** Only the holder of the private key can produce valid signatures. Therefore the source of data received is as claimed (data-origin authentication).
- **Integrity:** The signature is invalidated by any alteration of the message. During verification there will be a signature mismatch.
- **Non-repudiation:** The signer cannot deny having signed a message that verifies correctly under their public key. Only certification of a public-key by a trusted third party allows digital signatures to achieve also *non-repudiation*.
- **Unforgeability:** It is computationally infeasible for an adversary to generate a valid signature on a new message without knowledge of the private key.

In practice, digital signatures are not computed directly on large messages. Instead, a cryptographic hash function is first used to compute a compact message digest:

$$h = H(m).$$

The digest h is then signed using the private key. This approach is called **hash-and-sign** paradigm. Therefore, signing and verification process typically proceeds as follows:

1. The signer computes the message digest $h = H(m)$.
2. The signer applies the private signing function to h to produce the signature σ.
3. The verifier recomputes the digest $h' = H(m)$ and verifies that σ is valid for h' with the public key.

The National Institute of Standards and Technology (NIST) has standardized and approved several digital signature algorithms in FIPS 186-5 [24]. These include the Elliptic Curve Digital Signature Algorithm (ECDSA), the Edwards-Curve Digital Signature Algorithm (EdDSA), and the RSA signature scheme. Notably, EdDSA operates differently from traditional schemes and does not rely on the conventional hash-and-sign approach. Moreover, NIST has recently standardized the Module-Lattice-Based Digital Signature Algorithm (FIPS 204 [6]), and the Stateless Hash-Based Signature Algorithm (FIPS 205 [7]) as part of the ongoing transition toward post-quantum cryptography.

2.8 Public-Key Infrastructure

In modern cryptography, users of a public key require confidence that the associated private key is owned and controlled by the correct remote subject (be it a person, server, or system), with which an encryption or digital signature mechanism will be used. This confidence is obtained through the use of public-key certificates.

Definition 2.8.1. A **public-key certificate (PKC)**, also known as *digital certificate*, is a data structure that cryptographically binds a public key to an identity (the "subject"). The certificate contains the subject's public key along with information about their identity and other relevant attributes, and its digitally signed by a **Certification Authority (CA)**, a trusted third party (TTP), which vouches for the authenticity of this binding.

One of the most important purposes of a PKC is to achieve **non-repudiation** in digital signatures. Non-repudiation ensures that a digital signature is legally valid and the signer cannot later deny of having performed such signature over a blob of arbitrary data representing, for example, a transaction or a commitment.

However, a certificate alone does not explain how trust in its issuer is established, how identities are verified before issuance, or how certificates are revoked when keys are compromised. To address these and other challenges, information systems rely on what is known as **Public Key Infrastructure (PKI)**. A PKI is a framework that is established to issue, maintain, distribute and revoke public-key certificates. The most widely adopted standard for this framework is the Internet's PKI, specified in RFC 5280 [25], and commonly referred to as **PKIX** (Public-Key Infrastructure X.509), which is based on the **X.509v3** certificate format.

2.8.1 PKI Components

A complete public key infrastructure is composed of several components. As already anticipated, the **Certification Authority (CA)** is the entity that serves as the trusted third party responsible for issuing and revoking certificates. To manage the identity verification process, a CA may delegate this function to a **Registration Authority (RA)**. The RA is the entity responsible for verifying the claimed identity of the requesting entity before submitting the certification request to the CA. In some cases, the RA may live inside the CA itself.

Before issuing a certificate, the CA must also receive proper assurance that the subject actually possesses the corresponding private key. This is known as **Proof-of-Possession (PoP)**, which can be enforced using asymmetric challenge-response protocols, where the subject proves control of the private key by responding to a challenge with a digital signature. In practice, however, the most common method of demonstrating PoP is by generating a **Certificate Signing Request (CSR)**, which is defined in `PKCS#10` [26]. In this protocol, the applicant generates their key pair, creates a CSR containing their public key and identity information, and then digitally signs the entire request using their new private key. When the CA receives the CSR, it verifies the signature using the public key within the request. A valid signature provides cryptographic proof of the possession of the corresponding private key.

Once issued, certificates and revocation information (CRLs) are published to a **Certificate Repository**, a public directory (often based on LDAP or HTTP) that allows relying parties (i.e., clients) to retrieve them.

2.8.2 Certificate Revocation

A certificate is issued with a fixed validity period, but circumstances may require it to be invalidated before its expiration. This action, called **revocation**, is necessary for several reasons such as loss or compromise of the subject's private key, changes to the subject's information, or certificate issuance mistakes. PKIX defines two primary mechanisms for distributing revocation information.

The original X.509 method is the **Certificate Revocation List (CRL)**. A CRL is a time-stamped list, digitally signed by the CA (or a delegated **CRL Issuer**), that contains the serial numbers of all certificates it has revoked that are not yet expired. CAs issue new CRLs on a regular periodic basis (e.g., hourly, daily, or weekly). A relying party must download the most recent CRL from the repository and check if the certificate's serial number is on that list. The main limitation of this method is latency. A revocation reported now will not be known to users until the next CRL is published, leaving a window of vulnerability.

To address this latency, the **Online Certificate Status Protocol (OCSP)** was developed. OCSP is a client-server protocol that provides real-time revocation status of certificates. A client sends a query for a specific certificate's serial number to a **Validation Authority (VA)**, also known as **OCSP Responder**. The VA, which maintains an up-to-date database of revocation status, returns a digitally signed response indicating the certificate's current status, which can be *good*, *revoked*, or *unknown*.

2.8.3 Certificate Transparency (CT)

While CRL and OCSP provide mechanisms to check for known revocations, they do not address the threat of a CA mis-issuing a certificate, either maliciously or through a security breach, that the rightful owner does not know exists. **Certificate Transparency (CT)**, standardized in RFC 9162 [27], was developed to address this gap. The goal of CT is to make the issuance and existence of TLS certificates publicly auditable thereby providing "evidence of mis-issuance" to interested parties (particularly those named in certificates).

Certificate Transparency mitigates this problem by providing public, cryptographically assured, **append-only logs**. This cryptographic assurance is accomplished using Merkle Hash Trees, that store accepted certificates. **Submitters**, which are often the CAs themselves as part of their issuance process, present certificates to logs for public recording.

Upon accepting a certificate, the log returns a **Signed Certificate Timestamp (SCT)**. This SCT is a promise from a log to incorporate an accepted certificate into its Merkle tree within a specific maximum time. This proof of logging enables two primary functions for improving the trustworthiness of the TLS certificates presented by web servers on the Internet:

1. **Monitors**, publicly accessible servers, can continuously watch logs for certificates of interest. This permits any domain owner, or any interested party, to inspect public logs for any certificates issued for their domains, enabling the timely detection of unauthorized or fraudulent issuances.
2. **Clients**, typically web browsers, can establish policies that require that all certificates they accept are accompanied by one or more SCTs. These SCTs are typically delivered to the client during the TLS handshake, in different ways. This client-side enforcement makes it infeasible for a CA to secretly mis-issue a certificate, as any certificate lacking the required SCTs would be rejected by clients. Furthermore, any certificate that is logged is permanently and publicly recorded.

2.8.4 Certificate Chains and Path Validation

A relying party does not trust every CA in the world individually. Instead, its application software, such as an operating system or a browser, is pre-configured with a small set of trusted CA certificates called **Trust Anchors**. A trust anchor is typically a self-signed certificate, a certificate whose subject field and issuer field overlap, and whose signature can be verified using the public key contained in the certificate itself.

To authenticate an **end-entity (EE)** certificate (e.g., a web server), a relying party must build and validate a *certification path*, also called *certificate chain*. A certification path is a sequence of certificates, starting at the trust anchor, which bridges the trust anchor to the end-entity certificate. Such a chain can consist of one or more intermediate CA certificates. In a common three-level hierarchy, the trust anchor (called **Root CA**), signs the certificate of an Intermediate CA, which in turn signs the certificate of the end-entity.

The primary goal of path validation is to verify the binding between the subject's name and public key in the target certificate, using the public key of a trusted anchor as the ultimate basis of trust. This process requires verifying that the entire certification path satisfies certain conditions. First, the validation algorithm checks that each certificate's `subject` field matches the `issuer` field of the certificate following it in the chain. The first certificate in the chain must be issued by a trust anchor, whose self-signed certificate is not part of the transmitted chain but is pre-installed in the relying party's trust store.

The validation algorithm then processes the entire chain, recursively verifying each certificate in the path up to a trusted root. For each certificate, it confirms that the digital signature is valid, that the certificate was valid at the time of verification, that it has not been revoked (via CRL or OCSP), and that it adheres to all applicable constraints. This includes verifying that any CA certificate is authorized to sign other certificates (via its Basic Constraints extension) and that its public key is being used for its designated purpose (via its Key Usage extension). Only if all these steps succeed does the client accept the end-entity certificate as authentic and trustworthy.

2.9 PKCS

Public Key Cryptography Standards (PKCS) are a set of public-key cryptography standards that were developed and published by RSA Laboratories (a subsidiary of RSA Security Inc.) in conjunction with system developers worldwide to promote the deployment of Public Key Cryptography.

Since the first release of the PKCS#1 standard back in 1991, and through years of subsequent updates and publications of new standards, PKCS became ubiquitous in the cryptographic landscape. Over the last 30 years, the strict adherence to PKCS specifications, ranging from PKCS#1 through PKCS#15, became fundamental in enabling seamless interoperability between different applications. Moreover, several PKCS standards have recently become integral part of the standardization procedures of important standardization bodies such as the **IETF** and **PKIX** working group.

The following table lists all the PKCS standards:

Standard	Name	Comments
PKCS#1	RSA Cryptography Standard	
PKCS#2	Withdrawn	Merged into PKCS#1
PKCS#3	Diffie-Hellman Key Agreement Standard	
PKCS#4	Withdrawn	Merged into PKCS#1
PKCS#5	Password-Based Cryptography Standard	
PKCS#6	Extended-Certificate Syntax Standard	Deprecated in favor of X.509 v3
PKCS#7	Cryptographic Message Syntax Standard	Replaced by RFC 3369 (CMS)
PKCS#8	Private-Key Information Syntax Standard	
PKCS#9	Selected Object Classes and Attribute Types	
PKCS#10	Certification Request Syntax Standard	
PKCS#11	Cryptographic Token Interface Standard	Also known as “Cryptoki”
PKCS#12	Personal Information Exchange Syntax Standard	
PKCS#13	Elliptic-Curve Cryptography Standard	Abandoned
PKCS#14	Pseudo-Random Number Generation	Abandoned
PKCS#15	Cryptographic Token Information Format Standard	

Table 2.2. Summary of PKCS Standards

2.9.1 PKCS#11: Cryptoki API

The PKCS#11 standard was first introduced in 1994 by RSA Security, which continued to develop and maintain it until draft version 2.30, which was published for review but never officially released. In late 2012, RSA Security turned the standard over to the newly formed OASIS PKCS#11 Technical Committee [28], which published the first updated version (2.40) in 2015.

This section provides a high-level overview of the PKCS#11 standard, focusing on fundamental concepts and interactions. The objective is to establish the foundational understanding necessary to comprehend one of the core building blocks utilized in the research work presented in this thesis.

To understand its scope, this study refers to the definition provided by the most recent PKCS#11 Specification Version 3.2 [29], available at the time of writing:

> The PKCS#11 standard specifies an application programming interface (API), called “**Cryptoki**” for devices that hold cryptographic information and perform cryptographic functions. Cryptoki follows a simple object based approach, addressing the

goals of technology independence (any kind of device) and resource sharing (multiple applications accessing multiple devices), presenting to applications a common, logical view of the device called a "**cryptographic token**".

Cryptoki, short for **Cry**ptographic **Tok**en **I**nterface, is essentially a standard programming interface that enables applications to perform cryptographic operations using tokens, such as Hardware Security Modules (HSM), USB tokens, or smart-cards, while abstracting the details of the underlying device implementation maintaining portability across different cryptographic tokens. The general model for Cryptoki is illustrated in the diagram below.

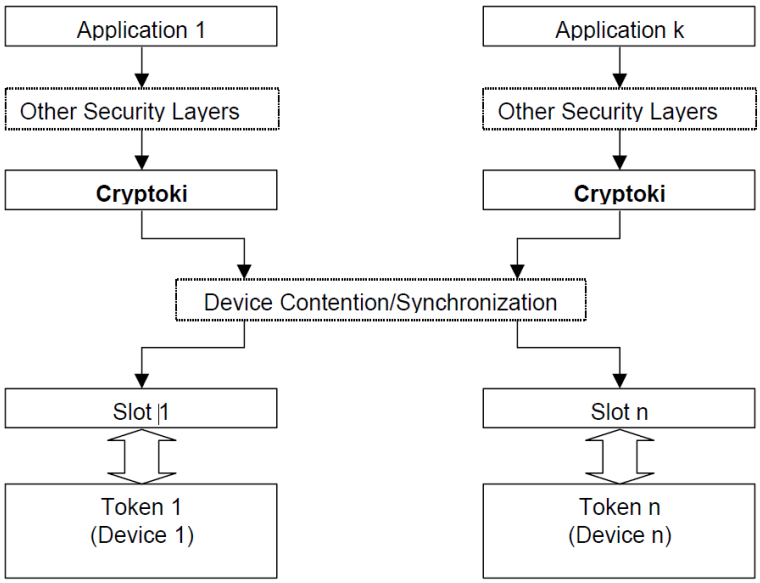

Figure 2.1. General Cryptoki Model

Cryptoki allows to interact with one or more cryptographic tokens present in a system, structured via a set of **slots**. Each slot usually corresponds to a physical reader or the equivalent device interface and can hold a token. Because Cryptoki defines a logical abstraction of slots and tokens, the physical structure can be different. For instance, several logical slots might share the same physical reader. To the application, the system is composed of a number of slots, each potentially associated with a token. To interact with a token in any or all of these slots, the application establishes a **session**, which serves as a logical connection between the application and the token. A cryptographic device provides a number of cryptographic operations, which are invoked through a set of predefined commands. These commands are typically processed through standard device drivers, for example PCI Express services or socket services. Cryptoki abstracts these implementation details managing this complexity on behalf of the application. Interestingly, the "device" itself can be **purely software-based** without requiring any specialized hardware. In practice, Cryptoki is often realized as a **library** that supplies the functions specified by its interface, to which applications link at execution time. Linking can occur either statically, typically more secure, in which case the application includes the Cryptoki library directly, or dynamically, less secure but code signing techniques can prevent potential security threats, through a shared library (e.g., Dynamic Link Library).

The standard defines a list of **general data types, objects and functions**. These include general information types, such as `CK_INFO`, which provides details about Cryptoki, including its version, manufacturer, and other flags. Slot and token types, such as `CK_SLOT_ID` and `CK_TOKEN_INFO`, identify slots and provide token-specific information. Session types, such as `CK_SESSION_HANDLE`, identify a session, and `CK_USER_TYPE` define user roles. In the `PKCS#11` standard's logical view, a token is a device that stores various object types and performs cryptographic operations. Object types, such as `CK_OBJECT_CLASS`, specify an object's class. Cryptoki

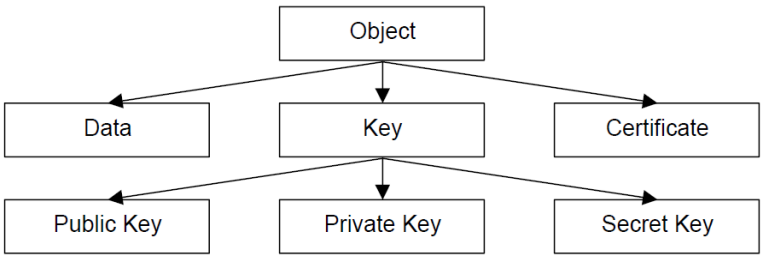

Figure 2.2. Object Hierarchy

recognizes three classes: data, keys, and certificates. As depicted in Figure 2.2 a key object stores a cryptographic key. The key may be a public key, a private key, or a secret key.

An **object** is defined by several attributes contained in a structure called `CK_ATTRIBUTE`. An array of attributes defines a **template** and is used for creating, manipulating and searching for objects. According to their lifetime and visibility, objects are classified as "token objects" and "session objects". Further classification defines access requirements. For example only a normal user can access a "private object", but it must be authenticated to the token by a **PIN** or some other token-dependent method. The PIN is set by the role of the Security Officer (SO), which is also in charge of initializing the token. Currently, Cryptoki only recognizes the aforementioned user types. Cryptographic operations are executed using defined **mechanisms**, which are requested through a standardized set of functions. Mechanism-related types, such as `CK_MECHANISM_INFO`, describe supported cryptographic algorithms and capabilities. Additionally, function list types, such as `CK_FUNCTION_LIST`, provide pointers to the available Cryptoki API functions. The standard also includes concurrency control types, such as `CK_CREATEMUTEX`, which enable multi-threaded access and synchronization [30].

2.10 The Transport Layer Security (TLS) Protocol

The Transport Layer Security (TLS) protocol is the most widely deployed cryptographic protocol that provides a secure channel between two communicating peers over untrusted networks. TLS is the *de facto* standard for securing Internet communications, protecting access to web pages, connections to email servers, or the setup of VPN connections.

TLS is a client-server protocol that should not be viewed as a single protocol, but rather as the combination of two layers of protocols. The TLS Record Protocol processes outgoing messages by fragmenting tha data into manageable blocks, protecting each record, and transmitting the result. Each TLS record is typed, allowing multiple higher-level protocols to be multiplexed over the same record layer, specifically the Handshake protocol, the Change Cipher Spec protocol, the Alert protocol, and the Application Data protocol. In particular, the Hypertext Transfer Protocol (HTTP), which provides the transfer service for web client-server interaction, can operate on top of TLS, commonly known as HTTPS. Among these, the TLS *handshake protocol* allows a client and a server to unilaterally, or mutually, authenticate each other (server authentication is mandatory, while client authentication is optional), negotiate a set of protocol parameters and cryptographic algorithms, and establish a shared key material. Subsequently, the TLS *record protocol* uses the parameters established by the handshake protocol to provide confidentiality, integrity, and authenticity of the exchanged application data.

TLS evolved from its predecessor, the *Secure Socket Layer (SSL)* version 3.0, proposed by Netscape Communications in 1996. The Internet Engineering Task Force (IETF) standardized the protocol as TLS 1.0 in 1998 [31], followed by many revisions that led to version 1.1 in 2006 [32] and version 1.2 in 2008 [33]. In 2014, the IETF initiated the development of **TLS 1.3** in response to several security concerns with version 1.2. This effort resulted in the publication of RFC 8446 [34] in August 2018, which has since seen widespread adoption making TLS 1.3 the major version in use today. The primary goals of TLS 1.3 were to enhance security and improve performance. These goals were achieved by introducing a one-round-trip time (1-RTT) handshake,

encrypting more of the handshake protocol, signing the entire handshake transcript to provide explicit authentication, and introducing a new key derivation scheme for generating separate keys to encrypt handshake messages and application data. TLS 1.3 also deprecated a wide variety of cryptographic algorithms, and mandated the use of Authenticated Encryption with Associated Data (AEAD) schemes to protect application data.

The TLS 1.3 protocol defines two primary handshake modes: the full **one-round-trip time (1-RTT)** handshake and the **resumption or pre-shared key (PSK)** mode. In the full 1-RTT handshake, the client initiates a connection to a server without any prior knowledge of the server's preferences or keys. Authentication is achieved using public-key certificates for the server and, optionally, for the client. The key exchange relies on (elliptic-curve) Diffie-Hellman ephemeral ((EC)DHE). In this mode, thanks to the TLS 1.3 *Key Schedule*, several session keys are derived for different purposes: encrypting parts of the handshake, exporting keying material to other applications, facilitating session resumption, and encrypting the application data. The 1-RTT handshake derives its name from the fact that application data is sent by the client immediately upon completion of the handshake. In contrast, the PSK mode relies on a symmetric pre-shared key for authentication, optionally combined with (EC)DHE for forward secrecy. The pre-shared key is usually derived from a **session ticket** issued by the server to the client after a successful full 1-RTT handshake. When reconnecting, the client can use this ticket to resume a previous TLS session, avoiding the cost and latency of the full handshake. TLS 1.3 also introduces an optional **zero-round-trip time (0-RTT)** feature in PSK mode, allowing the client to send application data immediately with its first message, known as *early data*, encrypted with a key derived from the PSK [35].

2.10.1 TLS 1.3 Handshake Protocol

This section examines only the TLS 1.3 full 1-RTT handshake with mutual authentication, intentionally omitting session resumption and PSK mode for simplicity. The general protocol flow also applies to (mandatory) server-only authentication, except for certain messages that are clearly highlighted in the discussion. The scheme presented in Figure 2.10.1 and its description draw mainly from Dowling et al. (Cryptographic Analysis of the TLS 1.3 Handshake Protocol) [35], while maintaining proper references to RFC 8446 [34]. The TLS 1.3 Key Schedule is mentioned here multiple times in support of the discussion. Any detailed information can be found in [34, Section 7.1].

The client initiates the handshake by sending the `ClientHello` (CH) message, as part of the first client flight, which includes a 256-bit random nonce (r_c), the protocol version, a list of supported symmetric cipher options (for AEAD and a hash for HKDF), and a set of extensions. Crucially, the client includes the `key_share` extension, which contains its ephemeral public key share $X \leftarrow g^x$ used for (EC)DHE key exchange, along with the `supported_groups` extension indicating the (EC)DHE groups supported by the client, and the `signature_algorithms` extension specifying the signature algorithms it can accept for authentication. When preparing its key material and assembling the first message, the client optimistically chooses the key exchange algorithm and group supported by the server. If the client has not provided a sufficient `key_share` extension (e.g., it includes only (EC)DHE groups unacceptable to or unsupported by the server), the server corrects the mismatch with a `HelloRetryRequest`, and the client must restart the handshake with a new `ClientHello` containing an appropriate `key_share` extension.

At this point, both peers start the Key Schedule, which is the key derivation process used in TLS 1.3 and relies on the HMAC-based Extract-and-Expand Key Derivation Function (HKDF) primitive [36]. They deterministically generate the Early Secret (ES), which is extracted from a zero-value input (if no PSK is being used), and expand the Derived Early Secret (dES). This secret is used as the HKDF.Extract salt for the next key derivation stage.

The next message is the `ServerHello` (SH) message. The server includes its 256-bit random nonce (r_s), the selected list of supported algorithms and parameters, and generates its own ephemeral public key $Y \leftarrow g^y$. Both peers can now compute the same ephemeral shared secret DHE $\leftarrow X^y = Y^x$. This value is mixed with dES to derive the main Handshake Secret (HS). From the HS, both peers can immediately derive the client handshake traffic secret (CHTS) and the

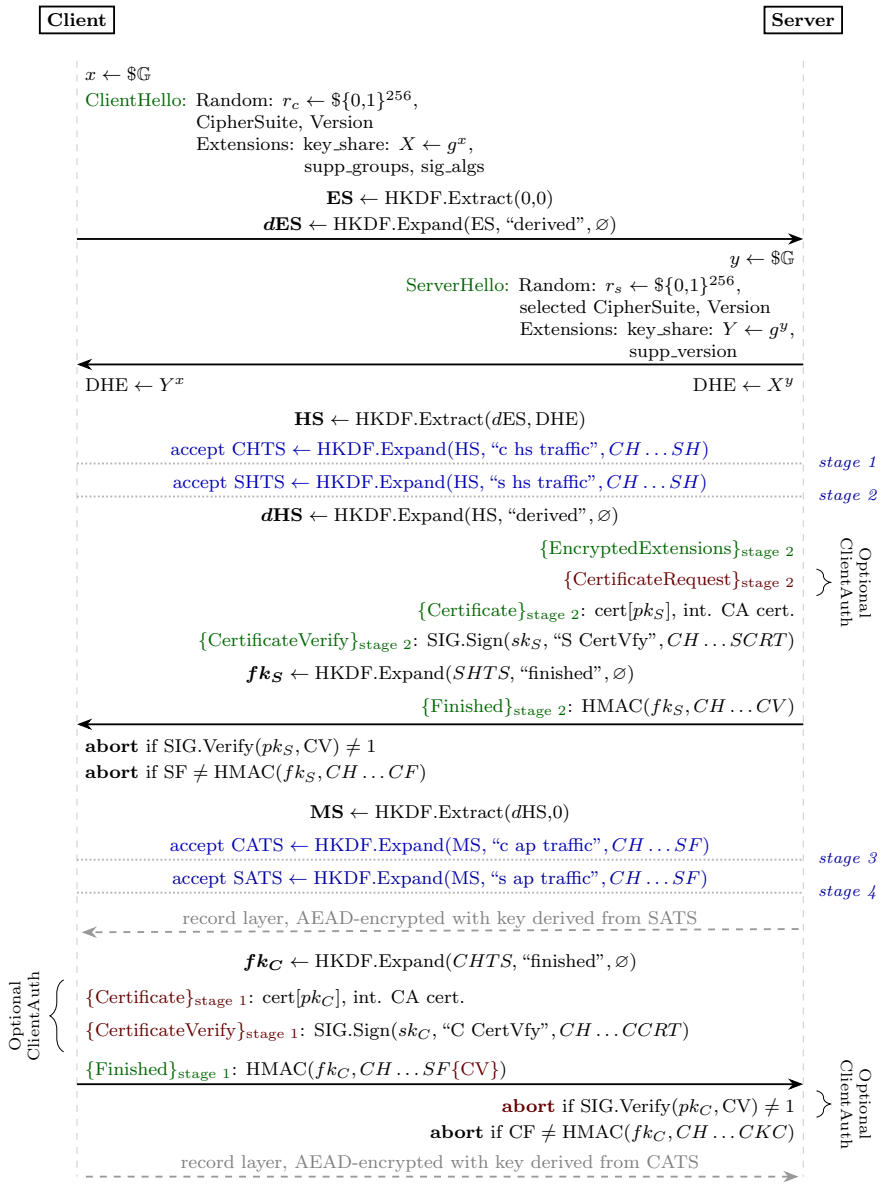

Figure 2.3. TLS 1.3 full 1-RTT Handshake with mutual authentication

server handshake traffic secret (SHTS). These keys are bound to the partial handshake transcript (CH. . . SH) and are used to encrypt the remainder of the handshake messages.

The server then sends the `EncryptedExtensions` message, which contains all the extensions that are not used for cryptographic computations, the `Certificate` message containing the server's public key certificate and its certificate chain (SCRT), and the `CertificateVerify` (CV) message, which contains a digital signature over the entire handshake transcript completed so far. This is the explicit authentication step performed by the server, proving possession of the private key corresponding to its public key certificate. The last message sent by the server is the `Finished` (SF) message, which contains a MAC over the handshake transcript computed with the server's derived finished key (fk_S). This serves as key confirmation, proving that the server correctly derived the Handshake Secret (HS). If the server requests optional client authentication, it sends the `CertificateRequest` message (highlighted in red) immediately after `EncryptedExtensions`. This message tells the client that it must present a certificate. All these messages are sent encrypted using the SHTS key.

After sending its first flight, the server can now derive the Master Secret (MS) from the

Derived Handshake Secret (dHS) and has enough information to compute the client application traffic secret (CATS) and the server application traffic secret (SATS). This allows the server to begin transmitting encrypted application data to the client without waiting for the client's final message flight.

At this point, the client receives all the messages sent by the server flight and prepares to send its second flight. The client verifies the entire server certificate chain up to a trusted root, the server's signature in `CertificateVerify`, and the MAC in the `Finished` message. In case of any verification failure, the connection is aborted. If the client received a `CertificateRequest`, it responds by sending its certificate and certificate chain (CCRT) in the `Certificate` message, and the `CertificateVerify` message containing the signature over the handshake transcript up to that point (both highlighted in red). Finally, the client sends the `Finished` message, which contains a MAC over the entire handshake transcript computed with the client's derived finished key (fk_C). All these messages are sent encrypted using the CHTS key. Only in the case of client authentication does the server now verify the client's certificate chain, the signature in `CertificateVerify`, and the MAC in the `Finished` (CF) message. If any of these verifications fail, the connection is aborted.

In the same way as the server, the client now derives the Master Secret (MS), from which it derives the client application traffic secret (CATS) and the server application traffic secret (SATS), and transmits its first encrypted application data.

2.11 Quantum Computing

The rise of the quantum computing field can be traced to late twentieth-century efforts to combine two distinct scientific theories: information theory and quantum mechanics. Historically, classical information theory, which is the foundation of modern computer science and communications, succeeded by abstracting information away from the physical world. This abstraction allowed for the analysis of algorithms and protocols independent of their physical implementation. This detachment from physics, however, created a bias blind spot: the assumption that computation itself was constrained by the principles of classical physics. While classical computers are typically understood as purely classical systems ruled by deterministic laws, they fundamentally depend on quantum mechanical principles in their physical implementation, for example, in transistors and semiconductors that form the basis of modern processors.

By re-evaluating computation through the lens of quantum mechanics rather than classical mechanics, researcher discovered that the physical laws governing the universe allow for computational processes fundamentally different from those of the classical Turing machine. This discovery led to the idea that quantum phenomena, such as quantum entanglement, might be exploited to accelerate certain types of calculations. Exploring this possibility required revising the traditional, classically based theoretical model of computation. Quantum information science, which encompasses quantum computing, quantum cryptography, and quantum communications, investigates how information behaves when it's described by quantum rather than classical laws. Quantum computing is not about changing the physical substrate on which computation is performed, but rather changing the notion of computation itself.

2.11.1 Quantum Bit

The break from classical computing starts with the physical representation of information. The fundamental concept of classical information theory is the *bit*, a physical system that exists in one of two discrete, deterministic states (e.g., high or low voltage) corresponding to logic 0 or 1. Quantum information is based on an analogous concept, the *quantum bit*, or **qubit** in short. According to the first postulate of quantum mechanics, the state of an isolated physical system is described by a unit vector in a complex vector space (a Hilbert space, $\mathcal{H}$), known as the *state space* of the system.

The *qubit* is a quantum mechanical system which, by definition, has a two-dimensional state space. Just as a classical bit has a state, 0 or 1, a qubit exists as a linear combination of both.

Using the Dirac *bra-ket* notation, the computational basis states are denoted as $|0\rangle$ and $|1\rangle$. The state of a qubit $|\psi\rangle$ is described by the *superposition*:

$$|\psi\rangle = \alpha|0\rangle + \beta|1\rangle \tag{2.9}$$

Here, α and β are complex numbers known as **probability amplitudes**. This equation encloses the phenomenon of **quantum superposition**, where the system explores the vector space spanned by the basis states. The qubit differs from a bit in the superpositions of these two states, in which it is not possible to say that the qubit is definitely in the state $|0\rangle$, or definitely in the state $|1\rangle$, but contains both possibilities simultaneously.

Measurement Problem

Unlike the state of a classical bit that can be determined precisely as either 0 or 1, the quantum state of a qubit cannot be determined with the same precision, that is, the values of α and β. In quantum mechanics, according to the measurement postulate, measuring a qubit $|\psi\rangle = \alpha|0\rangle + \beta|1\rangle$ forces the system to collapse from its superposition into one of the computational basis states. The result $|0\rangle$ is obtained with probability $|\alpha|^2$, and the result $|1\rangle$ with probability $|\beta|^2$. For this reason, α and β are called *probability amplitudes*, and they are physically constrained by the normalization condition:

$$|\alpha|^2 + |\beta|^2 = 1 \tag{2.10}$$

While the measurement of a qubit yields a discrete binary outcome, the qubit itself exists in a continuous state space. Geometrically, this can be visualized as a point on the surface of a unit sphere: the **Bloch Sphere** (Figure 2.4).

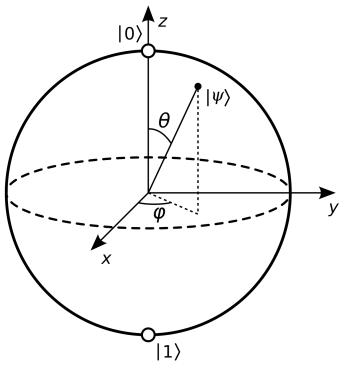

Figure 2.4. Bloch sphere representation of a qubit

- The North Pole represents the state $|0\rangle$.
- The South Pole represents the state $|1\rangle$.
- Any other point on the surface represents a unique superposition state (e.g., the equator represents an equal "50/50" superposition). While a classical bit is constrained to the two poles, a qubit can access the entire continuum of the sphere's surface, encoding information in the continuous variables of phase and angle.

The superposition is rarely an equal distribution. The coefficients α and β can represent any weighted combination of the basis states. This continuum allows the qubit to encode an infinite amount of information in the relative magnitudes and phases of its amplitudes.

2.11.2 Multiple Qubits and Entangled States

With two classic bits, the possible states would be four: 00, 01, 10, 11. Correspondingly, a two-qubit system has four computational basis states: $|00\rangle, |01\rangle, |10\rangle$, and $|11\rangle$. A pair of qubits can also exists in superpositions of these four states, so a generic state of a two-qubit system is described by the linear combination:

$$|\psi\rangle = \alpha_{00}|00\rangle + \alpha_{01}|01\rangle + \alpha_{10}|10\rangle + \alpha_{11}|11\rangle \tag{2.11}$$

where the sum of the squared moduli of the complex coefficients equals 1, satisfying the normalization condition expected for a legitimate quantum state. In general, a group of n qubits, formerly referred to as a *n-qubit quantum register*, is characterized by a set of 2^n complex amplitudes, which must describe every possible combination of bit strings from $|00\ldots0\rangle$ to $|11\ldots1\rangle$. This exponential scaling represents the core reason why a classical computer cannot efficiently simulate a quantum computer, as it would require memory proportional to 2^n to store the same amount of information contained in only n qubits.

An important property of multi-qubit systems is the ability to create superpositions of states that are said to be entangled. An *entangled state* is defined as a global state that cannot be described independently, meaning it cannot be factorized into independent states of individual qubits, but they are perfectly correlated.

A prominent example is the two-qubit state called *Bell state* (or *EPR pair*):

$$|\Phi^+\rangle = \frac{|00\rangle + |11\rangle}{\sqrt{2}} \tag{2.12}$$

In the Bell state, measuring the first qubit yields $|0\rangle$ or $|1\rangle$ with equal probability (1/2). However, the measurement outcomes are perfectly correlated. If the measurement of the first qubit yields $|0\rangle$, the global superposition collapses to the state $|00\rangle$, instantly determining that the second qubit is also in state $|0\rangle$. Symmetrically, if the first qubit is measured as $|1\rangle$, the system collapses to $|11\rangle$, forcing the second qubit to also be in state $|1\rangle$. This correlation ensures that the state of the second qubit is determined by the measurement of the first, regardless of the spatial separation between them. This phenomenon goes under the name of **quantum entanglement**, which has no classical analogue.

2.11.3 Quantum Gates and Quantum Circuits

Just as a classical computer processes information using electrical circuits made of wires and logic gates (such as AND, OR, NOT), a quantum computer uses **quantum circuits** containing "wires", which represent the temporal evolution of qubits, and **quantum gates** to manipulate information carried by these qubits.

Quantum gates act as *unitary operators* on the qubit's state vector. Geometrically they perform rotations of the state vector within the Hilbert space, visualized as rotations on the Bloch sphere. In this way, a quantum computer, deliberately generates superpositions, modifies relative phases, and entangles qubits to perform computation. Quantum circuits are constructed from a set of gates, such as:

- **The Hadamard Gate (H)**: This single-qubit gate creates superposition. By applying H to a basis state, it transforms a definite $|0\rangle$ or $|1\rangle$ into an equal probability superposition:

$$H|0\rangle = \frac{|0\rangle + |1\rangle}{\sqrt{2}} \tag{2.13}$$

- **The Pauli-X Gate**: This is the quantum equivalent of the classical NOT gate, flipping the amplitudes of $|0\rangle$ and $|1\rangle$.

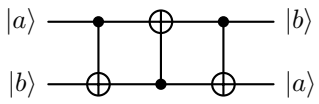

Figure 2.5. Circuit of an equivalent SWAP gate

- **The CNOT (Controlled-NOT) Gate**: This two-qubit gate is essential for generating entanglement. This gate interacts with a pair of qubits: a *control* and a *target*. The CNOT induces a conditional logic operation wherein a Pauli-X transformation is applied to the target qubit only if the control qubit is in the state $|1\rangle$. If the control qubit is in the state $|0\rangle$, the target remains invariant. Mathematically, this operation can be generalized as computing the XOR function.

In the quantum circuit model, an algorithm is essentially a finite sequence of quantum gates. Thus, the terms *quantum circuit* or *quantum gate array* are used interchangeably to refer to a **quantum algorithm**.

Although a comprehensive analysis of quantum circuits falls outside the scope of this chapter, the following example of a **SWAP gate** illustrates a simple composition of elementary gates into functional quantum logic. The circuit in Figure 2.5 swaps the states of two qubits ($|a, b\rangle \rightarrow |b, a\rangle$) by cascading three CNOT gates with alternating control and target assignments. First, the CNOT controlled by a transforms the state to $|a, a \oplus b\rangle$. Next, the second CNOT uses the new second qubit $(a \oplus b)$ as the control, updating the first qubit: $|a \oplus (a \oplus b), a \oplus b\rangle$. This simplifies directly to $|b, a \oplus b\rangle$. Finally, the third CNOT, now controlled by b, completes the exchange: $|b, (a \oplus b) \oplus b\rangle$. This results in the final swapped state, $|b, a\rangle$, since $(a \oplus b) \oplus b = a$.

2.11.4 Quantum Parallelism and Interference

The quantum speed-up achieved by quantum computers is derived from the interplay of quantum parallelism and interference.

Quantum parallelism is a fundamental feature of most quantum algorithms that describes how, by leveraging superposition, quantum computers can evaluate a function $f(x)$ into many different values of x simultaneously. By applying the Hadamard transform[1] to a register of n qubits initialized at $|0\rangle$, a quantum computer can prepare a state that is an equal superposition of all 2^n possible binary strings simultaneously. If a quantum circuit (representing a function) is applied to this register, the quantum logic evaluates the function for every possible input in a single step, storing all the possibilities (in superposition) in the output register.

However, measuring this superposition simply yields a random result. To solve problems, quantum algorithms leverage *interference*. Because probability amplitudes (α, β) are complex numbers, they possess a phase. Therefore, amplitudes with the same phase add together, amplifying the probability of the corresponding outcome (i.e., **constructive interference**). Amplitudes with opposite phases cancel each other out (i.e., **destructive interference**).

A quantum algorithm is typically designed to manipulate these phases. The goal is to exploit a constructive interference pattern for the correct answer, making its probability amplitude large, and a destructive interference pattern for all wrong answers, making their probability amplitudes zero.

[1]The Hadamard transform takes a qubit from a definite state and converts it into a state where it has an equal probability of being measured as either $|0\rangle$ or $|1\rangle$.

2.11.5 The Threat of Shor's Algorithm

In 1994, AT&T researcher Peter Shor published the article "Algorithms for quantum computation: discrete logarithms and factoring" [2], in which he introduced a quantum algorithm, later known as **Shor's algorithm**, capable of solving the integer factorization problem, the discrete logarithm problem (DLP), and the elliptic curve discrete logarithm problem (ECDLP) with exponential speed-up. In practical terms, Shor's algorithm transforms problems that would require thousands of years for a classical supercomputer to solve into ones that a sufficiently powerful quantum computer could complete in hours or days.

The exceptional efficiency of Shor's algorithm derives from its ability to solve a more general class of "hard" problems known as *period finding*. For a function $f(x)$ that is periodic, meaning there exists some period ω such that $f(x + \omega) = f(x)$, Shor's algorithm efficiently identifies ω by applying quantum mechanical principles to number theory. In the specific context of integer factorization, the objective is to factor a large number N into its prime factors. This challenge is reformulated as the task of finding the period ω of the modular exponentiation function $f(x) = a^x \pmod N$, where a is a randomly chosen coprime to N. Finding the period ω of this function is the necessary condition for the completion of the factorization: once the period is known, the prime factors of N can be derived by computing the greatest common divisor (GCD) of $(a^{\omega/2} \pm 1)$ and N.

The process begins by initializing a quantum register into a superposition of all possible integer exponents. This state enables quantum parallelism, wherein the modular exponentiation function $f(x)$, can be evaluated for all inputs simultaneously. However, the resulting quantum state contains a superposition of all possible outcomes, meaning a direct measurement would collapse it yielding a single random value.

To extract the specific periodicity from this superposition, the algorithm applies the Quantum Fourier Transform (QFT). The QFT operates similarly to a classical Fourier Transform but relies on wave interference to manipulate probability amplitudes of the quantum state. The details are complex, but what it is important to grasp is that incorrect solutions are cancelled out through destructive interference, while the states corresponding to the true period ω are amplified by constructive interference. As a consequence, the probability distribution of the system peaks strongly at the value of the period, allowing it to be measured with high probability.

The crucial point is that, in the absence of quantum superposition, each exponent must be evaluated one by one, therefore, even the most efficient classical factoring algorithms require time exponential in the bit length of N (where $n = log_2 N$). By contrast, Shor's algorithm operates in polynomial time, specifically $O(n^2(\log n)(\log\log n))$, placing integer factorization, and similarly the discrete logarithm problem, into tractable problems, therefore threatening the security assumptions upon which modern public-key cryptography relies.

2.11.6 Grover's Algorithm

A completely different class of algorithms is represented by the quantum search algorithm, first introduced by Lov K. Grover in 1996 with the publication of "A fast quantum mechanical algorithm for database search" article [3], which later became known as Grover's algorithm.

Grover's algorithm offers a solution to the unstructured search problem, establishing a quadratic speed-up over all classical algorithms. The algorithm finds the value x that satisfies a black-box function such that $f(x) = 1$, requiring just $O(\sqrt{N})$ evaluations of the function, where N is the size of the function domain (i.e., the size of the search space). On a classical computer, the same problem would require $O(N)$ evaluations, which, on average, takes $N/2$ steps. The quadratic speed-up is not impressive as the exponential speed-up offered by Shor's algorithm, but Grover's algorithm is still of great interest for a wide range of problems.

Also Grover's algorithm exploits superposition and interference. The algorithm prepares a quantum state in an equal superposition of all N possibilities using a set of Hadamard gates, so all potential solutions possess equal probability amplitudes. A quantum oracle identifies the target state by inverting the phase of the corresponding quantum state, marking the solution with

a phase of -1. The core efficiency gain is achieved through amplitude amplification, a technique that uses controlled interference to selectively increase the probability amplitude of the marked state while simultaneously suppressing the amplitudes of all non-target states. Repeating this procedure ensures that the correct answer's probability amplitude is maximized, making it the result obtained upon measurement with high certainty.

From a cryptographic perspective, Grover's algorithm effectively **halves** the complexity of exhaustively searching the solution space for an n-bit symmetric key or hash preimage, reducing the time complexity from $O(2^n)$ to $O(2^{n/2})$. Because this represents only a quadratic advantage, the threat to symmetric cryptography is generally considered manageable. The solution is to double the key or hash size. For example, to maintain 128-bit security, symmetric keys must be increased to 256 bits. This adjustment ensures that the Grover search complexity remains high, specifically $O(2^{256/2}) = O(2^{128})$ operations, thereby restoring the desired level of computational security.

2.11.7 Current State and Challenges in Building Quantum Computers

The current stage of quantum computing development is classified as the **Noisy Intermediate-Scale Quantum (NISQ)** era. Introduced by John Preskill in 2018 [37], this term denotes a class of quantum processors with system sizes ranging from approximately 50 to a few hundred physical qubits. Such processors are subject to significant limitations arising from environmental noise and imperfect control. The intermediate-scale term reflects the fact that these systems are theoretically large enough to exceed capabilities of brute-force classical simulation. However, achieving the **quantum advantage** in the near term remains extremely challenging due to **quantum decoherence**, which is the physical process responsible for quantum information loss.

Formally, decoherence describes the degradation of a quantum system when it interacts with its surrounding environment, such as in the presence of thermal fluctuations or electromagnetic radiation. Unlike a closed system that evolves coherently and predictably, an open system loses its purity over time. These interactions cause the fragile quantum interference effects essential for superposition and entanglement to decay rapidly. As this process unfolds, the system effectively transitions from a quantum state capable of parallel computation into a classical mixed state described by classical probabilities, leading to computational errors. Consequently, the feasibility of large-scale quantum computing relies on **Quantum Error Correction (QEC)**, a technique that redundantly encodes logical information across many physical qubits to detect and correct errors faster than they occur.

As of early 2025, major hardware developers like IBM and Google have shifted their focus from merely doubling qubit counts annually toward prioritizing enhanced error resistance. For instance, IBM's recent roadmaps emphasize the transition from their *"Heron"* class processors toward the *"Starling"* system, which aims to demonstrate logical qubits capable of suppressing error rates below the physical threshold within the next decade [38]. Similarly, Google's *"Willow"* chip has recently demonstrated a reduction in logical error rates with increasing numbers of physical qubits, a necessary condition for scalable fault-tolerance [39].

Despite these milestones, the transition to fully **Fault-Tolerant Quantum Computers (FTQC)** requires overcoming substantial engineering challenges, primarily the ability to maintain quantum states against environmental noise. As a consequence, until these physical error rates are suppressed via quantum error correction codes, the threat to classical cryptography remains theoretical rather than immediate.

2.12 Post-Quantum Cryptography

The preceding sections have provided a more rigorous analysis on why the classical assumptions under which modern cryptographic schemes are considered computationally "unbreakable" are no longer sufficient once quantum computation is incorporated into the adversarial model. In light of this, the central question is no longer whether existing cryptographic schemes are vulnerable, but *how cryptographic design must evolve* to remain secure in a post-quantum setting.

Post-Quantum Cryptography (PQC) emerges as a direct response to this paradigm shift. PQC refers to public-key cryptographic algorithms designed to remain secure against adversaries equipped with both classical and quantum computational capabilities, while operating entirely within the classical computational model. As such, PQC is fundamentally distinct from *quantum cryptography*, which derives security from the quantum phenomena and requires specialized quantum communication channels and a completely different hardware to operate.

To contextualize the current state of the art, this section examines the diverse range of post-quantum cryptographic algorithms designed to withstand quantum cryptanalysis. Although these algorithms are not essential for understanding the contributions of this work, they provide a high-level overview of the mathematical foundations underlying the relatively young PQC landscape. The principal families of PQC algorithms include lattice-based cryptography, code-based cryptography, hash-based cryptography, multivariate polynomial cryptography, and isogeny-based cryptography.

Lattice-based cryptography

Lattice-based cryptographic constructions represent perhaps the most versatile and robust candidate for the post-quantum transition, whose security relies on the computational intractability of geometric problems defined over mathematical lattices. A lattice can be described as a regular, repeating arrangement of points in a multi-dimensional vector space, generated by linear combinations of a set of basis vectors with integer coefficients. Their relevance in the field stems from the difficulty of solving certain lattices problems, such as the **Shortest Vector Problem (SVP)** and the **Closest Vector Problem (CVP)**, which involve finding, respectively, the shortest non-zero vector in a lattice or the lattice vector closest to a given point. These problems are widely believed to be intractable for both classical and quantum adversaries [1, p. 147].

A major breakthrough in this area was notably achieved with Ajtai's worst-case to average-case reductions, which establish that breaking typical instances of certain lattice-based schemes is as hard as solving the most difficult instances of the underlying lattice problems [40]. While early constructions were largely impractical [41], subsequent work led to more efficient schemes such as NTRU [42] and later designs based on the **Learning With Errors (LWE)** problem [43]. Introduced by Regev, LWE and its variants, including Ring-LWE, Module-LWE, and Learning With Rounding (LWR), form the basis of many modern lattice-based encryption and key agreement schemes, offering improved efficiency through added algebraic structure. Unlike classical public-key cryptography, lattice-based schemes do not rely on integer factorization or discrete logarithms and are therefore resistant to quantum algorithms such as Shor's.

Given their flexibility, mathematical robustness, and support for advanced functionalities like fully homomorphic encryption it's not surprising that three lattice-based candidates have been selected for standardization in NIST PQC standardization project [4]. In particular, NIST selected ML-KEM (FIPS 203 [5]), derived from CRYSTALS-Kyber, as the primary standard for key encapsulation, alongside ML-DSA (FIPS 204 [6]) for digital signatures, based on CRYSTALS-Dilithium, and FN-DSA, based on FALCON, another lattice-based digital signature scheme which is currently being finalized as FIPS 206.

Code-based cryptography

Code-based cryptography represents one of the most mature approaches within post-quantum cryptography. Its security is based on the **General Decoding Problem**, that is the computational intractability of decoding a general linear error-correcting code without the knowledge of the underlying structure. This problem is recognized as NP-hard, and is robust also against quantum adversaries. The most significant example roots back to 1978 with the **McEliece cryptosystem** [44], which utilizes binary Goppa codes, and has withstood nearly fifty years of cryptanalysis. Unlike RSA and ECC, the McEliece system remains fundamentally unbroken by quantum algorithms, as it does not rely on the number-theoretic symmetries targeted by Shor's algorithm.

Despite its robustness, the main disadvantage of traditional code-based schemes is the large size of their public keys, which often reach several hundred kilobytes. While code-based cryptography

has proven highly successful for encryption and key agreement, the development of efficient code-based digital signature schemes has been more challenging. Traditional constructions tend to suffer from large signatures or high computational costs, limiting their practical deployment. Among the code-based candidates evaluated in the NIST PQC standardization project, Classic McEliece, BIKE, and HQC represent the primary submissions, with HQC currently selected for standardization as the main code-based KEM [45].

Hash-based cryptography

Hash-based cryptography constitutes a highly conservative and mathematically robust branch of post-quantum cryptography. Hash-based schemes derive their security entirely from the collision resistance and one-way properties of cryptographic hash functions, and are considered inherently resistant to quantum cryptanalysis, suffering only a quadratic reduction in security by Grover's algorithm.

The primary cryptographic application of hash-based schemes in the post-quantum setting is the construction of digital signatures. These schemes trace their origins to early constructions such as Lamport's One-Time Signature schemes (OTS), and the later work of Ralph Merkle, who in 1979 introduced the **Merkle Tree** [46]. A Merkle tree is essentially a cryptographic commitment scheme in which a large set of one-time signature public keys are aggregated into a single root hash. In this hierarchical construction, the leaves of the binary tree represent the hashes of individual OTS public keys. Internal nodes are generated by the recursive concatenation and hashing of child nodes until the root is reached, and the resulting root hash serves as the general public key for the entire scheme. The main advantage lies in the efficient verification of authenticity by providing an authentication path (or Merkle proof) from a specific leaf to the root.

Modern hash-based signature schemes are divided into "stateful" and "stateless" schemes. Stateful schemes, such as the **Leighton-Micali Signature (LMS)** [47] and the **eXtended Merkle Signature Scheme (XMSS)** [48], require the signer to maintain a persistent record of used OTS keys. Because a one-time key is cryptographically compromised if reused, the signer must increment an internal index to ensure each leaf in the Merkle tree is used only once. This architecture is computationally efficient and allows for relatively small signature sizes, but any failure in the state management can reset the index and lead to security vulnerabilities [49].

To mitigate these risks, stateless schemes such as **SLH-DSA** (formerly SPHINCS+) were developed and finalized by NIST in 2024. These schemes eliminate the need for index tracking by utilizing a multi-level "hyper-tree" structure containing a large number of few-time signature (FTS) keys. By picking keys pseudorandomly from an enormous pool, the probability of an accidental collision becomes negligible, effectively removing the danger of state synchronization errors. On the other hand, the primary trade-off is a substantial increase in signature size and signing latency compared to its stateful counterparts [7].

Multivariate cryptography

Multivariate cryptography refers to a class of asymmetric cryptographic primitives based on systems of multivariate polynomials over finite fields. The security of these schemes is derived from the difficulty of solving systems of non-linear equations, a problem known to be NP-complete (or NP-hard in the case of quadratic equations).

The most prominent example of a multivariate construction is the **Unbalanced Oil and Vinegar (UOV**) digital signature scheme [50], which remains one of the most secure and stable designs to date. In contrast, several other multivariate submissions to the NIST PQC standardization project, including Rainbow, LUOV, GeMSS, and MQDSS, were either significantly weakened or entirely broken by cryptanalysis and subsequently rejected [51].

Generally, multivariate solutions like UOV offer exceptionally small digital signatures and high-speed verification. However, they require very large public keys, often exceeding several hundred kilobytes. Consequently, multivariate cryptography is not considered a general-purpose

replacement for current standards. Instead, it is viewed as a specialized solution for specific use cases, such as securing communication channels where signature bandwidth is highly constrained, but sufficient memory is available to accommodate large public keys.

Isogeny-based cryptography

Isogeny-based cryptography, which emerged from research in late 1990s, is one of the most recent cryptographic families to gain particular attention in the context of post-quantum standardization process. Isogeny-based schemes are built on the presumed computational hardness of finding isogenies, structure-preserving maps, between elliptic curves. Notable examples are the Supersingular Isogeny Key Encapsulation (SIKE), which reached the fourth round of the NIST PQC standardization project, and the Commutative Supersingular Isogeny Diffie–Hellman (CSIDH) construction, a relevant alternative for non-interactive key exchange.

Initially, SIKE gained particular interest due to its compact public keys, often under 600 bytes even at high security levels, offering significant bandwidth advantage over lattice-based alternatives. However, the field faced a critical inflection point in July 2022 when a classical cryptanalytic attack successfully broke SIKE [52]. By exploiting auxiliary points within the protocol, researchers recovered secret keys using conventional hardware in less than an hour.

Beyond this vulnerability, isogeny-based schemes generally suffer from high computational costs compared to lattice-based approaches, despite their competitive smaller key sizes. Furthermore, because these schemes have been studied for a relatively short period, they carry less resonance than other well-known established methods, as brief history of these primitives suggests that further attacks may yet be discovered.

2.12.1 NIST's Security Levels

NIST established five security strength categories during its PQC standardization process to facilitate systematic comparison of post-quantum algorithms against well-understood classical cryptographic primitives. These categories, referred to as **security levels**, are defined in terms of the computational resources required to break them, measured against reference problems from symmetric cryptography that are expected to remain secure even against quantum attacks. The five NIST security levels are presented in Table 2.3, listed in order of increasing strength. Each level specifies that any attack breaking the relevant security definition must require computational resources comparable to or greater than a designated reference problem.

Level	Equivalent Security	Reference Problem
I	AES-128	Exhaustive key search (128-bit)
II	SHA-256	Collision search (256-bit hash)
III	AES-192	Exhaustive key search (192-bit)
IV	SHA-384	Collision search (384-bit hash)
V	AES-256	Exhaustive key search (256-bit)

Table 2.3. NIST PQC Security Levels

2.12.2 From Harvest Now, Decrypt Later to Trust Now, Forge Later

The urgency of transitioning to post-quantum cryptography is often formalized using **Mosca's inequality**, illustrated in Figure 2.6. In his work [53], M. Mosca characterizes the quantum threat as a race against time, defined by three critical timeframes: the duration $\mathbf{x}$ for which current communications must remain secure *(shelf-life)*, the time $\mathbf{y}$ required to transition systems to a quantum-safe state *(migration time)*, and the time $\mathbf{z}$ until a cryptographically relevant quantum computer is realized *(collapse time)*. Mosca's observation is expressed by the inequality $\mathbf{x} + \mathbf{y} > \mathbf{z}$: if the requirements for data shelf-life and infrastructure migration exceed the time remaining until

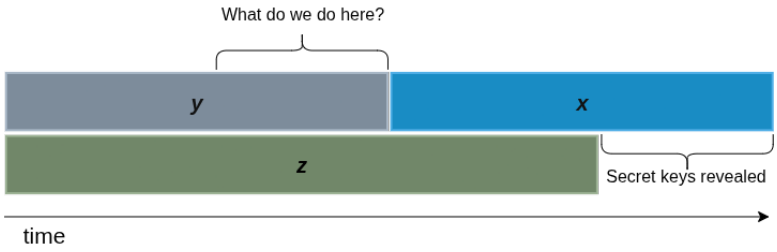

Figure 2.6. Mosca's inequality

the quantum "breaking point" (also known as the *quantum day*), the security of the system is already compromised. Unfortunately, it is currently uncertain whether the migration time can be completed before the collapse time arrives.

Mosca's inequality is commonly invoked to motivate the urgency for the post-quantum transition, especially when the timeline is framed in terms of threat models such as Harvest Now, Decrypt Later (HNDL). However, although the loss of long-term confidentiality is a major concern, it is not the only security property threatened by the advent of cryptographically relevant quantum computers (CRQCs). A more comprehensive risk assessment would easily reveal a secondary, perhaps more destructive, threat that has recently gained attention within the community, referred to as **Trust Now, Forge Later (TNFL)**.

The TNFL threat model targets authenticity and integrity which are foundational to establish digital trust in modern information systems. Widely adopted, classical, digital signature schemes such as RSA and ECDSA, are mathematically vulnerable to future CRQCs executing Shor's algorithm. Therefore, an adversary will eventually be able to derive private signing keys from publicly available verification keys. The risk posed by TNFL stems from the fact that trust decisions made today are cryptographically anchored to public keys that may become forgeable in the future. Certificates, signed software updates, identity assertions, audit logs, legal documents, and blockchain transactions all rely on the assumption that a digital signature uniquely and permanently binds an action or statement to a specific entity. If the underlying signature scheme is compromised, an adversary could generate valid signatures indistinguishable from authentic ones, effectively rewriting history or impersonating trusted actors. In this sense, TNFL represents a temporal integrity failure where trust is established now under assumptions that will not hold indefinitely.

While HNDL targets long-term value of encrypted data to compromise confidentiality, TNFL exploits the long-term validity of public keys and digital signatures to compromise non-repudiation. TNFL is not limited to future communications. For instance, previously issued signatures, such as those in long-lived certificates or signed artifacts, lose their cryptographic assurance once the signing key can be retrospectively forged. The exposure to this risk threatens entire trust infrastructures, such as Public Key Infrastructures (PKIs), secure boot chains, and distributed ledgers.

Moreover, TNFL exacerbates the urgency expressed by Mosca's inequality. The migration time y for signature systems is often significantly longer than for encryption alone, as it involves not only cryptographic primitives but also certificate lifecycles, root trust anchors, long-lived credentials, and legal or regulatory dependencies. Moreover, many signed objects are designed to remain valid for decades, far exceeding typical data confidentiality requirements. In such cases, the effective shelf-life x of authenticity may be substantially longer than that of secrecy, increasing the likelihood that $x + y > z$ will be satisfied even if data is not explicitly harvested for decryption. Once a signature scheme is broken, previously established trust cannot be reliably recovered, even if systems are later upgraded. Consequently, post-quantum migration strategies must treat signature algorithms as a primary concern rather than a secondary objective to key establishment and encryption.

2.12.3 Post-Quantum Migration Planning of Web PKI

Following NIST's standardization of quantum-resistant algorithms in FIPS 203, 204, and 205, the industry has entered a critical implementation phase characterized by considerable technical complexity and differing strategic views regarding optimal migration paths in Web Public Key Infrastructure (PKI). The **PKI consortium**[2], which has emerged as a central coordinating body for industry-wide PQC adoption efforts, emphasizes that the transition is not merely about cryptographic algorithms replacement but rather a comprehensive infrastructure transformation requiring careful orchestration across multiple stakeholders categories.

The industry's approach to PQC migration has converged on several strategies, with hybrid cryptographic schemes outstanding as the predominant transitional mechanism. Hybrid approaches combine traditional cryptographic algorithms with post-quantum ones, providing a security safety fallback that maintains protection against both classical and quantum adversaries during the migration period. This strategy reflects the industry's recognition that immediate wholesale replacement of classical algorithms poses unacceptable operational risks, particularly given the extended deployment timeline characteristics of PKI ecosystems where certificate lifetimes, validation infrastructure dependencies, and backward compatibility requirements create substantial inertia. For example, the **European Telecommunications Standards Institute (ETSI)** has provided technical guidance on migration strategies through its **TR 103 619** specification [54], which delineates multiple migration paths including pure PQC deployment, hybrid certificate chains, and composite cryptographic constructions. ETSI's framework acknowledges that different organizational contexts will necessitate different migration strategies based on factors such as existing infrastructure capabilities, risk tolerance, and operational constraints.

Significant technical debates have arisen regarding concrete implementation mechanisms for the hybrid approaches, particularly the distinction between composite and hybrid certificate structures. Composite certificates, as defined in the IETF Internet-Draft for composite ML-DSA signatures [55], bind both classical and post-quantum algorithms together within a single X.509 structure, where the composite keys and composite signatures are treated as a unified cryptographic algorithm from the perspective of the application layer, effectively creating a new algorithm identifier that abstracts the dual-algorithm nature from relying parties. This approach offers the advantage of simplified application logic, as software is not required to perform multiple independent signature verifications. However, composite approaches introduce new standardization challenges and create dependencies on broad ecosystem support before deployment can begin. In contrast, hybrid approaches based on dual-signature mechanisms or certificate chain hybridization distribute classical and post-quantum elements across multiple certificates or signature layers, potentially offering greater flexibility and incremental deployment capabilities at the cost of increased structural complexity. The latest PKI Consortium's 2025 PQC Conference in Kuala Lumpur revealed substantial industry uncertainty regarding which approach will ultimately achieve dominance, with participants expressing concerns about premature standardization potentially fragmenting the ecosystem [56].

At the same time, **cryptographic agility** has been identified as a fundamental prerequisite for successful PQC migration and future cryptographic transitions. In this context, crypto-agility extends beyond the ability to easily replace algorithms to encompass also organizational capabilities including algorithm discovery mechanisms, rapid key rotation procedures, multiple algorithm support across all PKI components, and systematic processes to maintain accurate inventories of all cryptographic assets. Achieving this level of agility requires substantial architectural investments to redesign legacy systems that were built assuming cryptographic permanence. Nevertheless, the PKI Consortium emphasizes that these investments yield lasting benefits that extend beyond the immediate quantum threat [56].

[2] https://www.pkic.org/

Industry Initiatives

The transition to post-quantum cryptography in Web PKI presents unique governance challenges, as the ecosystem operates through industry self-regulation coordinated by the **Certificate Authority/Browser Forum (CA/B Forum)** rather than through direct governmental mandate. The CA/B Forum, a voluntary consortium of Certification Authorities, browser vendors, and operating system providers, establishes the baseline requirements that govern all publicly trusted digital certificates. While governments have issued directives for their own internal PKI infrastructure, Web PKI standardization progresses through multi-parties consensus within the CA/B Forum, with browser vendors holding significant influence through their root store policies.

The CA/B Forum's approach to PQC integration has proceeded cautiously through targeted experimental frameworks. In August 2025, the S/MIME Working Group passed Ballot SMC013[3], which introduced specifications for post-quantum algorithm use in S/MIME certificates, marking the first formal acceptance of PQC algorithms within CA/B Forum baseline requirements. This ballot enables Certificate Issuers to experiment with pure PQC certificates (non-hybrid implementations) utilizing ML-DSA signature algorithm, though it explicitly notes that additional requirements may be imposed by individual browser root programs. Significantly, the ballot does not yet extend to TLS certificates, reflecting the more complex technical and operational challenges associated with Web PKI server authentication.

The Chromium project has articulated a strategic framework for post-quantum Web PKI [57], establishing a rigorous performance requirement that certificate chains should not exceed approximately 7 kilobytes in total handshake overhead unless a cryptographically relevant quantum computer becomes tangibly imminent. To reconcile the significant size of standardized post-quantum signatures (such as ML-DSA) with these strict boundaries, Chromium proposes more dynamic alternatives to the traditional "drop-in" replacement of algorithms. Foremost among these is the exploration of **Merkle Tree Certificates (MTC)** [58], a mechanism that leverages existing Certificate Transparency (CT) infrastructure. By replacing large, distinct signature chains with compact proofs of inclusion within a verifiable log, MTC aims to suppress the on-the-wire size of authentication credentials, ensuring that the migration to quantum resistance does not come at the cost of the speed and stability of the TLS handshake.

Furthermore, the Chromium design principles emphasize the necessity of **negotiated agility**. Recognizing that no single post-quantum algorithm serves all use cases efficiently, the framework advocates for a flexible deployment model where servers can selectively present different certificate chains based on the client's advertised capabilities. This allows for a heterogeneous environment where legacy clients receive classical RSA or ECC chains, while modern clients negotiate quantum-resistant MTCs. This architectural agility is intended to prevent "algorithm lock-in", ensuring that the Web PKI can rapidly pivot to alternative primitives, such as multivariate schemes, should the standardized lattice-based algorithms face unexpected cryptanalytic failures in the future.

The CA/B Forum's recent approval of Ballot SC-081v3[4] in April 2025, which reduces maximum TLS certificate validity from 398 days to **47 days** by March 2029, indirectly supports PQC preparedness by forcing organizations to implement certificate lifecycle automation. Shorter certificate lifetimes require widespread automation of certificate issuance and renewal processes, capabilities that are essential for managing future algorithm transitions. This represents an indirect regulatory mechanism: by mandating operational practices that necessitate automation and cryptographic agility, the CA/B Forum creates conditions favourable to future PQC migration without explicitly requiring quantum-resistant algorithms.

The lack of governmental directives specific to public Web PKI reflects the ecosystem's reliance on market-driven coordination, though regulatory pressure exists indirectly through government requirements for their own systems. A principal area of strategic divergence lies in the choice

[3] https://cabforum.org/2025/07/02/ballot-smc-013/#results-of-ballot-smc013-enable-pqc-algorithms-for-smime

[4] https://cabforum.org/2025/04/11/ballot-sc081v3-introduce-schedule-of-reducing-validity-and-data-reuse-periods/

between hybrid versus pure cryptographic schemes. The U.S. National Security Agency (NSA), in its Commercial National Security Algorithm Suite 2.0 (CNSA 2.0), has advised against the use of hybrid schemes for National Security Systems [8]. Conversely, in Europe, France's Agence nationale de la sécurité des systèmes d'information (ANSSI), or German Federal Office for Information Security (BSI), advocate for a precautionary approach based on hybrid implementations during the transitional phase to ensure that data confidentiality and integrity remains anchored by mature classical assumptions should the newer post-quantum primitives succumb to unexpected cryptanalytic attacks [59] [60]. This creates a complex regulatory landscape where Web PKI participants must navigate industry self-regulation while remaining aware of governmental timelines that indirectly constrain their operational freedom.

Technical Challenges in PKI Components Migration

The practical implementation of PQC within existing Web PKI infrastructure faces numerous technical challenges. This section explores the most relevant complexities and barriers identified in IETF Internet-Draft *Guidance for Migration to Composite, Dual, or PQC Authentication* [61].

As briefly anticipated in the previous sections, the IETF has defined three transition models for use in TLS, IKEv2/IPsec, and PKIX:

- **Composite certificates** facilitate a single-path validation process: because both post-quantum and traditional algorithms are embedded in one certificate chain, the relying party needs only to validate one path. They provide defense-in-depth as traditional algorithms preserve security if a flaw is found in PQC algorithms before a CRQC exists while PQC preserve security once CRQCs can break traditional algorithms. However, this model faces ecosystem readiness issues. Clients, servers, and CAs must all support composite keys and signatures, which is not yet widely common. There is also a "algorithm-set coordination" challenge: all parties must agree on the same combinations of PQC and traditional algorithms, or validation and interoperability failures can occur. These are mainly early-deployment policy and interoperability problems. On top, composite certificates are only a transitional solution. Once traditional algorithms are fully deprecated, organizations must still move to pure PQC certificates, which involves deploying new PQC trust anchors and revoking composite certificates. Because trust anchor updates often rely on OS or platform update mechanisms, this two-step migration can increase overall transition time for some organizations.

- **Dual-certificates** use separate traditional and post-quantum certificates, prioritizing negotiation flexibility because endpoints do not need to pre-agree on specific algorithm combinations. They rely on standard X.509 structures and single-algorithm chains, which maximizes compatibility with existing PKI systems. However, the trade-offs are higher overhead and complexity: both certificate chains and signatures must be transmitted, validated, and managed, increasing handshake latency (e.g., in TLS) and operational costs. Debugging and operations are more difficult due to multiple chains and potential binding issues. Dual certificates also require explicit mechanisms to bind the traditional and PQC certificates to the same identity. Inconsistent binding policies can cause interoperability problems and potential downgrade risks.

- **Pure PQC certificates** avoid future deprecation or phased-out support for RSA and ECC, but the substantially larger key and signature sizes associated with algorithms such as ML-DSA and SLH-DSA create immediate compatibility issues with protocols and data structures designed under assumptions of compact classical cryptographic objects. Certificate chain validation, which in classical PKI typically involves transferring and processing several kilobytes of data, may require handling hundreds of kilobytes when quantum-resistant algorithms are employed, potentially exceeding protocol message size limits and creating performance degradation in bandwidth-constrained environments. Examples include embedded systems with limited memory, IoT devices with constrained network connectivity, and legacy protocols with hard-coded size limitations that cannot accommodate PQC certificates without proper protocol revision.

Hardware Security Modules (HSMs), which provide cryptographic key protection and signature generation capabilities for Certification Authorities and other critical PKI components, present particularly complex challenges. Many deployed HSM devices lack sufficient memory and the computational capabilities necessary to perform post-quantum cryptographic operations at operationally required throughputs. Modern HSM hardware may require firmware updates or complete replacement to support PQC algorithms, creating significant costs and operational disruption.

Certificate Transparency (CT) systems, which have become integral to Web PKI security by providing public logs of certificate issuance, also face scalability concerns. CT logs must store and serve complete certificates to monitors and auditors, with the expanded size of PQC certificates directly impacting storage requirements and network transfer costs. The multiplicative effect across the billions of certificates issued annually creates non-trivial infrastructure scaling requirements. Moreover, CT systems must support both classical and post-quantum certificates during transition periods, potentially requiring parallel logging infrastructure or enhanced log formats capable of efficiently handling mixed certificate types.

Online Certificate Status Protocol (OCSP) responders and **Certificate Revocation List (CRL)** distribution mechanisms face substantial challenges during the post-quantum transition, mainly related to increased certificate and signature sizes. CRLs must be updated at each transition stage to include post-quantum (pure, hybrid, or composite) certificates, otherwise, attackers could exploit inconsistencies to forge revocation proofs or suppress legitimate revocations. Although PQ-based CRLs are significantly larger than classical RSA/ECC-signed CRLs, this increase is unlikely to be a dominant issue, as CRLs are already large in high-revocation environments. Instead, the main challenge lies in adapting CRL formats and extensions. Similarly, hybrid/composite certificates in OCSP are possible even with very large certificates, though messages become substantially larger due to PQ extensions. On the other hand, OCSP responses must be generated and validated with minimal delay during TLS handshakes, and the impact of post-quantum certificate sizes and signatures on response time remains a key performance concern.

Root Certification Authorities face the unique challenge of extremely long-lived root certificates, some with validity periods exceeding twenty years, which creates complex migration dynamics. This migration must be phased, using approaches such as parallel hierarchies, composite roots, PQC intermediates cross-signed by classical roots, and pure PQC roots. During the transition, deployments may operate multiple trust models at once. As a result, clients fall into several categories, ranging from legacy-only to PQC-only support. The main issue is that servers cannot easily distinguish clients that merely support PQC algorithms from those that actually trust PQC roots. To maintain compatibility, servers may send longer, cross-signed chains, which can cause inefficiency or validation failures for some clients. To overcome this, emerging extensions like **Trust Anchor Identifiers (TAI)** in TLS allow clients to specify their recognized trust anchors on a per-connection basis, providing also essential telemetry on the ecosystem's PQC readiness [62].

Chapter 3

Related Work

This chapter reviews prior work and related efforts that form the basis of this thesis, with particular emphasis on initiatives addressing the transition to post-quantum cryptography. It surveys contributions from the QUBIP consortium, focusing on operating system-level integrations, and crypto-agile frameworks, as well as earlier work on integrating post-quantum mechanisms into Mozilla Firefox. Together, these efforts provide the technical and conceptual context in which the contributions of this thesis are positioned. To conclude, a brief analysis of the core building blocks is presented, offering the necessary knowledge about the architectural components and assumptions discussed in the subsequent chapters.

3.1 QUBIP Consortium Efforts

The QUBIP project brings together a consortium of European research institutions and industry partners, each addressing distinct aspects of the post-quantum transition across multiple technological domains, from low-level hardware implementations to high-level application protocols. This section summarizes only the main contributions of the partners whose work is complementary to, but distinct from, the Firefox/NSS efforts discussed separately.

3.1.1 Public Key Infrastructure

Politecnico di Torino (POLITO) led the development of QUBIP's Public Key Infrastructure, which provides authentication and trust establishment across different project components. It represents the foundation of the security of the main protocols targeted by the QUBIP project. Recognizing that different applications have varying security requirements and resource constraints, POLITO implemented a hierarchical PKI consisting of a Root Certification Authority (CA) and an Intermediate CA responsible for issuing end-entity certificates and signing Certificate Revocation Lists (CRLs). To address diverse deployment scenarios, three distinct certificate chains were established:

1. **Pure PQ Chain**: Utilizing SLH-DSA-SHAKE-256s for the root CA and ML-DSA-65 for the intermediate CA, this chain targets TLS endpoints, providing NIST security levels 5 and 3 respectively.

2. **MLDSA65-Ed25519 Composite Chain**: Utilizing MLDSA65-Ed25519 for both root and intermediate CAs, this chain targets MPU-based devices, providing NIST security level 3.

3. **MLDSA44-Ed25519 Composite Chain**: Utilizing MLDSA44-Ed25519 for both root and intermediate CAs, this chain targets resource-constrained environments, particularly MCU-based devices, providing NIST security level 2.

POLITO developed a web application[1] that manages certificate distribution, allowing clients to request X.509v3 certificates and select the appropriate issuing chain based on their requirements.

3.1.2 Operating Systems Integration

Red Hat selected **Fedora Linux** [63] as the ideal platform for providing initial PQC support, leveraging the Fedora Project's role as a community-driven testing ground for features intended for Red Hat Enterprise Linux (RHEL). This strategic choice allows experimental PQ integration to mature in a production-like environment before enterprise deployment.

The Fedora OS implementation, extends widely-known cryptographic libraries (OpenSSL, NSS, GnuTLS) to support PQ algorithms, automatically extending these capabilities to applications depending on those libraries. A system-wide mechanism called "**crypto-policies**" ensures consistent usage policies of certain classes for algorithms across the operating system. Red Hat's integration strategy prioritized open-source components such as:

- **liboqs** v0.12.0: Written in C, liboqs is an open-source library that provides comprehensive collection of NIST-standardized algorithms, maintains a suitable license, has responsive upstream maintenance, and crucially, uses **PQClean** as its low-level implementation source, which is the same implementation chosen for NSS integration, improving inter-library compatibility.
- **oqsprovider** v0.8.0: An OpenSSL `Provider` based on liboqs, developed by the same team. Its tight integration with OpenSSL (tested as part of OpenSSL's own integration tests) ensures compatibility between the provider and OpenSSL core functionality.

As of Fedora 42, only NIST-standardized post-quantum algorithms are compiled in. Experimental or pre-standardization algorithms, such as hybrid/composite variants, are excluded to maintain stability and security guarantees. Nevertheless, all the post-quantum algorithms are disabled by default and require activation of the TEST-PQ subpolicy through the crypto policies mechanism.

pq-container

Red Hat developed a dedicated container image, called **pq-container** [64], that demonstrates Fedora 42's post-quantum capabilities. The container includes all necessary components with activated TEST-PQ subpolicy and a configured NGINX TLS web server demonstrating PQ algorithm usage in practical scenarios. This containerized approach was also adopted throughout the work presented in this thesis, proving particularly valuable during the integration testing phase of the **QUBIP Internet Browsing Pilot**, one of the deliverables led by Tampere University and discussed later in this document. As will be shown in the following chapter, Fedora 42 was selected as the operating system for both client and server instances used to deploy the Internet Browsing Pilot.

Red Hat's contribution demonstrates that OS-level PQ integration can be achieved without disrupting existing ecosystems, providing a migration path that respects backward compatibility while enabling forward progress toward quantum resistance.

[1] https://ca.all.qubip.eu

3.1.3 Aurora: A Shallow Provider for OpenSSL

This section poses particular attention to a closely related work conducted by the NISEC research group at Tampere University: the `aurora Provider` [65]. Notably, the framework proposed in this thesis is methodologically aligned with `aurora` in its approach to cryptographic agility. As both the `aurora Provider` and the work carried out in this thesis originated from the same research environment, these two contributions, while targeting different software domains, were closely coordinated to ensure alignment in technical progression and strategic project objectives.

Formalized by Mehmood and Tuveri in "*Integrating PQC in OpenSSL via Shallow Providers for Cryptographic Agility*" [66], `aurora` establishes a central building block for the QUBIP project within the widely used OpenSSL ecosystem. It introduces the *shallow* `Provider` concept, which facilitates the modular integration of external cryptographic implementations into the OpenSSL library, thereby enabling a higher degree of cryptographic agility.

OpenSSL Providers and Shallow Provider Methodology

OpenSSL version 3.x introduced a flexible plugin architecture called `Providers`, which decouples cryptographic primitive implementations from the core library. `Providers` register support for key encapsulation mechanisms (KEMs), digital signature schemes, and other primitives, making them available through OpenSSL's high-level APIs, particularly the EVP API. This architecture naturally enables post-quantum migration by allowing novel algorithms to be introduced without modifying library internals or requiring changes to application code. However, the conventional `Provider` model assumes that cryptographic implementations reside within the `Provider` itself. Embedding implementations tightly within a `Provider` creates problematic coupling between the `Provider` lifecycle and the implementation evolution, necessitating frequent updates, and integration complexity. `aurora` addresses these limitations through the shallow `Provider` methodology.

The shallow `Provider` concept deliberately avoids embedding cryptographic implementations within the `Provider` module. Instead, the `Provider` functions as a lightweight delegation layer that forwards algorithm execution to external implementations through structured adapters. This design achieves clean decoupling from the implementation code of certain algorithms, allowing the `Provider` to maintain stability while algorithms are linked at runtime. Implementations can be rapidly replaced as standards evolve or vulnerabilities emerge without requiring substantial `Provider` modifications.

Architecture and Implementation

`aurora` is implemented in Rust, exploiting its memory safety guarantees and foreign function interface (FFI) capabilities to interoperate with C-based OpenSSL and external PQC libraries. The architecture comprises a `Provider` core that implements the OpenSSL interface and manages configuration, alongside an *adapter layer* that bridges specific external implementations with OpenSSL's `Provider` API expectations. The `Provider` core is built upon the `openssl-Provider-forge` Rust crate, developed by the same team at Tampere University, specifically to provide FFI bindings for OpenSSL 3.2+ and abstract common `Provider` implementation tasks. This toolkit lowers barriers for researchers experimenting with `Provider`-based extensions while promoting reuse of well-tested patterns, enabling `aurora`'s development without requiring direct engagement with low-level OpenSSL C APIs.

At the time of writing, `aurora` version 0.10.0 supports newly standardized post-quantum algorithms and their hybrid variants. For key encapsulation, it implements `X25519MLKEM768` and `Secp256r1MLKEM768`. For authentication, it provides `ML-DSA` and `SLH-DSA` signature schemes, as well as composite signature schemes such as `MLDSA44/65-Ed25519`. Applications using OpenSSL's EVP API automatically access `aurora`'s capabilities when the `Provider` is enabled, requiring no application modifications. Its modular structure facilitates the support of new schemes that can be accommodated by developing adapters without changing `aurora`'s `Provider` core.

Aurora Integration

`aurora` supports comprehensive post-quantum integration within TLS 1.3 for both key exchange and authentication. OpenSSL's Provider API includes explicit capabilities for registering KEMs and signature algorithms in a manner that **libssl**, OpenSSL's TLS protocol implementation, can directly consume during TLS handshake operations. `aurora` leverages this architectural capability to present its supported post-quantum and hybrid schemes to OpenSSL, thereby enabling PQ/T hybrid TLS 1.3 negotiation without requiring modifications to libssl internals or the TLS state machine.

`aurora` serves as a crypto-agile substrate for OpenSSL, reusable across applications that range from IoT devices to full-scale TLS server deployments. Beyond its role as a shallow provider, `aurora` serves as the underlying cryptographic "engine" for the QUBIP Certification Authorities managed by Politecnico di Torino. The `Provider` is instrumental in generating the post-quantum/hybrid PKI hierarchies required by project partners, such as the MPU- and MCU-based systems developed by POLITO, TELSY, and CSIC, and the TLS certificates required for both server and client authentication within the broader **QUBIP Internet Browsing Pilot**, delivered by Tampere University. This pilot is of particular significance to the current research. The Firefox/NSS post-quantum integration later presented in this thesis constitutes the client-side component of the aforementioned pilot. The architectural synergy between `aurora` on the server-side and the framework developed in this work ensures the end-to-end cryptographic consistency discussed in Chapter 4.

3.1.4 Prior Work and Contributions to Firefox's Crypto-Agility

The initial strategy for integrating post-quantum cryptography into Mozilla Firefox through a modular, crypto-agile approach was established by D.Luoma's research [12] during the early phase of the QUBIP project. The work presented in this thesis builds directly upon that prior contribution, addressing the gaps left by the previous author and extending its scope. The main concepts and rationale behind them are further revised and expanded upon in Chapter 4. This section highlights the key results achieved by Luoma in order to provide the necessary context for understanding the original contributions of this thesis and clearly delineate the new achievements relative to the initial goals set out for this work.

Luoma's thesis defines the **Kryoptic** project [67] as the chosen building block for implementing a crypto-agile `PKCS#11` loadable module to be integrated with Mozilla Firefox. Kryoptic, a Rust-based `PKCS#11` software token, was selected after an initial attempt to develop a loadable module from scratch in C proved impractical within the project's timeframe. The selection of Kryoptic was driven by several factors. First, the project provides a complete and well-maintained implementation of the Cryptoki API compliant with `PKCS#11` version 3.1, including essential cryptographic primitives, object management, and storage infrastructure. Second, Kryoptic offers a mature codebase with comprehensive unit testing, reducing implementation risks and development efforts. Third, the association of Kryoptic's developers with Red Hat, a direct partner in the QUBIP project, enabled direct collaboration with the original authors, further strengthening confidence in the framework's suitability.

Despite its robust framework, Kryoptic exhibited a heavy dependency on OpenSSL for its core cryptographic services. These services are critical not only for session-based operations requested by relying applications but also for internal management tasks, ranging from token initialization to the protection of user-associated cryptographic objects. Since the proposed module is intended to function as a loadable component to enhance Network Security Services (NSS) library (Firefox's cryptographic core), a hard dependency on a single, external library like OpenSSL is considered poor practice. Such a design exposes to conflicts and overlapping implementations, while also inheriting OpenSSL 3.2's specific limitations regarding post-quantum and hybrid algorithms support. This dependency contradicts the primary objective of providing a flexible, modular architecture capable of seamlessly swapping external cryptographic backends, a methodological approach analogous to the `aurora Provider` discussed previously and further expanded upon in Chapter 4.

The critical OpenSSL-dependent components identified were the Random Number Generation (RNG) mechanism, the PBKDF2 key derivation function, and several AES modes (AES-GCM, AES-KW, and AES-KWP) used for encryption and key wrapping. To address these dependencies, Luoma extended Kryoptic's build system using Cargo's conditional compilation features, introducing a "pure-rust" build target. This approach preserved the original Kryoptic codebase while providing an alternative compilation path that omitted OpenSSL-dependent features. Following these modifications, the adapted software token was successfully rebuilt and validated using standard `PKCS#11` tooling, demonstrating that the framework remained operational after the removal of OpenSSL.

The primary technical contribution of Luoma's work was the addition of support for **ML-KEM-768**. Following the release of Firefox 132[2], which introduced native support for post-quantum key exchange via the `mlkem768x25519` hybrid scheme, the author conducted a careful analysis of the internal architecture of NSS. The goal of this analysis was to determine how an external `PKCS#11` module could be invoked to delegate the post-quantum component of the hybrid key exchange to Kryoptic and, subsequently, how the underlying cryptographic operations could be offloaded to an external implementation. This was achieved by leveraging NSS's vendor-defined `PKCS#11` mechanism for `ML-KEM-768` and by implementing, within Kryoptic, a preliminary interface that bridges the module to **libcrux** [68], a formally verified cryptographic library written in Rust. By offloading the post-quantum computation from the core browser library to a modular provider, this implementation served as an initial proof of concept for cryptographic agility. This early experiment demonstrated the feasibility of a crypto-agile architecture, which is further expanded and re-structured in this thesis to support a wider range of emerging post-quantum algorithms.

3.2 Building Blocks: Preliminaries

This section introduces the building blocks that lay the background for the methodology adopted in this thesis. Before delving into the overall architecture and the workflow that enables post-quantum cryptography in Firefox, it's essential to dwell on the roles of Firefox and NSS, as well as the capabilities of Kryoptic, to fully understand how the final objective is achieved.

3.2.1 Kryoptic: RedHat's PKCS#11 Software Token

As introduced in the previous section, Kryoptic is an open-source, software-based `PKCS#11` token implemented in Rust, developed and maintained by Red Hat engineers Simo Sorce, Jakub Jelen, and Sergio Arroutbi. Kryoptic is the module selected to advance support for post-quantum and hybrid algorithms in a crypto-agile manner. In the Methodology chapter (Section 4.1), Kryoptic is referred to as the **QUBIP module** and underwent substantial modifications during the development of this thesis, after which it was renamed **qryptotoken**.

Without anticipating the results achieved, this section shifts focus to a general overview of Kryoptic's initialization workflow, internal database management, and encryption architecture, with particular attention to how cryptographic objects are securely stored and protected through PIN-derived keys and layered encryption mechanisms. This background is necessary to understand how cryptographic material migration can be achieved, particularly in light of certain limitations imposed by Firefox. As discussed later (Section 5.4), these limitations prevent cryptographic material from being exported or imported using the `PKCS#12` format (directly from Firefox GUI) when external modules are used instead of Firefox's built-in cryptographic module, as is the case in this thesis.

[2] https://www.firefox.com/en-US/firefox/132.0/releasenotes/

Storage Architecture Overview

Kryoptic employs a modular storage architecture that abstracts persistent data management behind a unified `Storage` trait interface. This design supports multiple backend implementations, including SQLite, JSON file-based, and in-memory stores, selected dynamically based on the token database file extension. The token runtime maintains a split object model: **persistent objects** (marked with `CKA_TOKEN` attribute) reside within the chosen persistent storage backend, while **session objects** exist transiently in a volatile hash map, reflecting their lifespan only within an active `PKCS#11` session. A handle-mapping subsystem mediates between external `PKCS#11` object handles and internal unique identifiers (UIDs), ensuring proper object lifecycle management across these two domains.

Token Initialization

When token initialization is invoked, the token first attempts to open an existing database file. If the file does not exist, or if explicit re-initialization is requested, the storage backend's irreversibly overrides any existing data structures and creates a new, empty database schema. Following storage preparation, the token's internal state, including session caches, login status, and the **Key Encryption Key (KEK)**, is reset to an empty condition.

The initialization sequence then establishes the Security Officer (SO) credential by processing the supplied SO PIN through Password-Based Key Derivation Function 2 (PBKDF2) algorithm, storing the derived key material in a reserved internal object. Subsequently, a cryptographically random AES-256 key is generated and serves as the master Key Encryption Key (KEK). This KEK is immediately encrypted (or "wrapped") using a key derived from a default user PIN, if not specified by the user. Finally, token metadata such as its label and serial number are serialized and stored, completing the initialization phase and marking the token as ready for use.

PIN Management and Key Encryption Key (KEK) Hierarchy

Kryoptic implements a hierarchical credential and key management system centred on the PINs defined by the `PKCS#11` standard and the KEK. The SO PIN value is never recorded. Instead, the PIN is processed along with a random salt through PBKDF2 with a significant iteration count, in the range of thousands. The resulting derived key is stored in a reserved secret key object (UID "0") alongside the salt and iteration parameters. Authentication involves re-deriving the key from the presented PIN and comparing it to the stored value.

The User PIN protects access to the token's encrypted contents via the KEK. Unlike the SO PIN, the User PIN object (UID "1") does not store a derived key directly. It stores the wrapped KEK that is the master AES-256 KEK encrypted using a key derived from the user PIN via PBKDF2. The wrapping operation is performed with AES in GCM mode with a fixed initialization vector and additional authenticated data, ensuring both confidentiality and integrity of the wrapped KEK. Successful user login requires deriving the key from the presented PIN, unwrapping (decrypting) the KEK, and caching it in memory for subsequent cryptographic operations. This design ensures that the KEK is never stored in plaintext and is inaccessible without the correct user PIN.

Both PIN mechanisms incorporate stateful attempt counters. After a configurable number of failed authentication attempts, the respective PIN is locked, a state reflected in the token's flag attributes. This defends against brute-force attacks.

Object Encryption and Persistent Storage

The most important security feature is the on-disk encryption of sensitive objects. When the `CKF_RESTORE_KEY_NOT_NEEDED` flag is set, indicating encryption is enabled, sensitive attributes of private objects are encrypted before being committed to persistent storage.

The encryption process is attribute-granular and context-aware. Object factories determine which attributes of a given key class are deemed sensitive. For storage, each sensitive attribute's plaintext value is encrypted independently with AES-GCM using the in-memory KEK and a randomly-generated IV. Crucially, the object's UID is included as Additional Authenticated Data (AAD) in the GCM construction, cryptographically binding the ciphertext to its specific container object and preventing substitution or relocation attacks.

Non-sensitive attributes (e.g., public keys, object labels, certificate data) and all object metadata are stored in plaintext, preserving the ability to search and enumerate objects without requiring decryption.

Database Portability and Security Guarantees

The Kryoptic design facilitates token portability through a single-file database. Transferring this file to a new host machine migrates all token contents, including encrypted private keys, certificates, and public objects. However, access to encrypted material remains confined to authentication. The security architecture adheres to the defense-in-depth principle: the layered encryption (PIN-derived keys wrap the KEK, which in turn encrypt object attributes), combined with attempt limiting and algorithmically sound constructions (PBKDF2, AES-GCM) it provides a robust defense against offline attacks on the stolen database.

In summary, Kryoptic implements a coherent and secure software token model by leveraging a pluggable storage database, a hierarchical key management system anchored by a PIN-protected KEK, and granular, attribute-level encryption for sensitive data. Its design ensures that the persistent database file is portable while maintaining strong security properties that protect key material even if the storage medium is compromised.

3.2.2 Mozilla Firefox

Mozilla Firefox is an open source cross-platform web browser developed by the Mozilla Foundation and actively maintained by its subsidiary, Mozilla Corporation. Historically, Firefox surpassed the well-known Microsoft's Internet Explorer to account for roughly 32% of all desktop browser usage as its peak in the late 2000s. Nowadays, Firefox's market share has declined due to the success of other vendors and the rise of Google Chrome. It represents approximately 5% of the global desktop market as of August 2025 [69]. Despite this, Firefox remains an important actor in the web ecosystem because of its excellent performance, interoperability, open standards, and strong commitment to privacy. Firefox is well-suited for experimental research and real-world application of the post-quantum cryptographic algorithms, thanks to its underlying open, transparent and accessible nature.

Although Firefox is an open-source project with large codebase, its internal architecture is not comprehensively documented in any recent official references. Much of the available material is either outdated or fragmented across developer resources, bug trackers, and source code comments. To provide a structured view, we refer to an architecture analysis study from 2007 by Alan Grosskurth and Michael W. Godfrey [70], which, while describing an earlier version of Firefox, still offers a valuable high-level overview of the browser's architecture.

Firefox has a complex, layered, and modular architecture in which different subsystems are responsible for distinct browser's functionalities. As shown in Figure 3.1, this architecture distinguishes the upper-layer components, responsible for user interface and interaction, from the lower-layer components, responsible for rendering, networking and security services. At the top, the **Cross-Platform Front End (XPFE)** provides a framework for implementing the **User Interface** independently of the underlying operating system, allowing Firefox to offer a consistent user experience across all supported platforms. Beneath it, the **Gecko engine** integrates both the browser engine and the rendering engine to process and display web content. Around Gecko, subsystems like **SpiderMonkey** provides the JavaScript runtime, **Expat** manages XML parsing, and the **GTK+ adapter** ensures integration with the underlying graphical libraries of the host operating system. All data persistence is provided by Mozilla's **profile mechanism**, which

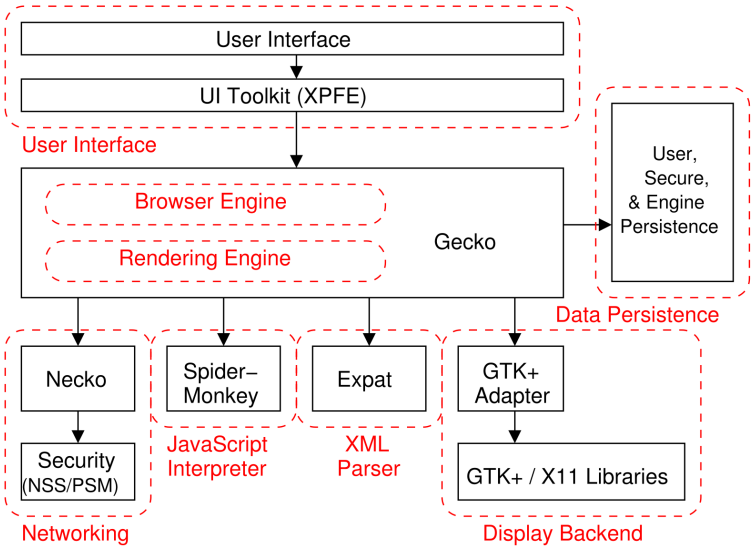

Figure 3.1. Mozilla Firefox Architecture (reproduced from [71, p. 10])

stores both high-level data such as bookmarks and browsing history, and low-level data such as `PKCS#11` tokens database, trusted certificates stores, page caches etc. Networking is delegated to the **Necko** subsystem, which implements internet protocols for client-server communication such as HTTP(S), FTP and WebSockets. Necko does not implement cryptography itself, but relies on the **Network Security Services (NSS)** library to establish secure communication channels. Communication between Necko and NSS is primarily mediated by the **Personal Security Manager (PSM)**, which is responsible for configuring TLS sockets, client certificate selection and server certificate verification.

Understanding this architectural separation is fundamental to clearly identify which are the components directly, and indirectly, involved in enabling PQC in Mozilla Firefox. The main focus is on the TLS stack, with NSS as the primary target since it serves as the cryptographic backbone of the browser. Although several components of Necko necessarily interact with or depend on NSS, and thus also require adaptation to ensure internal interoperability within Firefox, the bulk of the cryptographic transition remains concentrated at the NSS layer.

3.2.3 Network Security Services (NSS)

Network Security Services (NSS), originally known as Netscape Security Services, is a set of libraries designed to support cross-platform development of security-enabled applications through the **Netscape Portable Runtime (NSPR)** [72]. NSS provides implementation for a wide range of security standards, including `SSL/TLS`, `PKCS#5`, `PKCS#7`, `PKCS#11`, `PKCS#12`, `S/MIME`, and `X.509v3` certificates. NSS is an integral part of the Mozilla Firefox codebase, delivering the primitives and higher-level services to establish secure connections and validate trust on the web.

According to the official NSS documentation [73], one of the library's key design principles is its strict adherence to the `PKCS#11` standard, which allows NSS to work with many hardware or software modules that expose cryptographic services. Multiple elements of NSS's own modules have been implemented to work with the `PKCS#11` interface in mind, and NSS makes use of this interface when interacting with those modules.

The internal architecture of NSS is organized into several key components [74]:

- **FreeBL**: A base library that implements fundamental cryptographic primitives. Some operations are implemented internally, while others may delegate to external libraries.

- **softoken**: Wraps FreeBL and acts as the internal `PKCS#11` software module of NSS, allowing applications to access cryptographic services via the standard `PKCS#11` interface. The most direct example is Firefox, which uses softoken as its predefined internal `PKCS#11` module. By default softoken exposes two slots:
  - The **Generic Crypto Services** token, which performs all cryptographic operations.
  - The **NSS Certificate DB** token (where on Firefox is displayed as **Software Security Device**), which stores certificates and private keys in the user-provided directory, specifically in the files `cert9.db` and `key4.db`. The key database file contains private keys, instead the certificate database file contains the public portion of user's certificates, the certificates of peers or CAs, and a list of trust decisions (such as to not trust a built-in CA, or to explicitly trust other CAs).
- **CKBI (Built-in Root Module)**: Another `PKCS#11` module that exposes one slot containing the **Builtin Object Token**, which provides a predefined set of trusted root certificates following Mozilla's Root CA Policy. By default, Firefox uses the CKBI module allowing root trust anchors to be accessed in the same uniform way using the `PKCS#11` interface.
- **pk11wrap**: The crucial abstraction layer, namely the `PKCS#11` wrapper, that provides a unified interface to all `PKCS#11` modules, such as softoken and CKBI. It allows higher-level code to find slots, tokens, and perform operations without directly managing low-level `PKCS#11` details.
- **cryptohi**: The high-level cryptographic API for signing, verifying signatures, key generation and manipulation, and hashing. It builds upon "`pk11wrap`", providing simpler APIs for common operations related to Sign/Verify contexts.
- **certdb**: The core certificate and CRL database management library. It provides APIs to store, retrieve, and manage certificates and their trust attributes within NSS databases (i.e., "`cert9.db`"). It is the primary module for certificate object handling.
- **certhi**: The high-level certificate utilities library. It provides high-level certificate-related functions that do not access the certificate database nor individual certificate data directly.
- **mozpkix (libpkix)**: The modern PKIX certificate validation engine. It performs sophisticated certificate path building and policy evaluation, handles cross-signing and alternative trust paths, enforces policy constraints, evaluates extended key usages and algorithm policies, and performs revocation checks (OCSP/CRL). Mozpkix replaces the older, simpler verification engine and is responsible for robust certificate chain validation in modern and complex PKI scenarios.
- **pkcs12**: The library for encoding and decoding `PKCS#12` files, which bundle certificates and private keys for import/export. It relies on "`certdb`" to store imported certificates and on "`pk11wrap`" to manage keys.
- **secutil**: The general-purpose utility library providing utility functions and data types used by other libraries. The library supports base-64 encoding/decoding, reader-writer locks, the SECItem data type[3], DER encoding/decoding, error types and numbers, OID handling, and secure random number generation.

Building on these components, the **SSL/TLS** implementation in NSS (also known as `libssl`) manages secure communication. During a TLS handshake, cryptographic operations such as key exchange, signature verification, and key derivation are executed through the `PKCS#11` interface. Certificate chains are validated by `libpkix`, which consults CKBI and the NSS databases for trusted anchors and cached certificates. The result is a modular yet tightly integrated stack, where TLS depends on the uniformity of the `PKCS#11` abstraction to orchestrate complex tasks inherent to the protocol itself.

[3]SECItem is a C structure used to handle blobs of arbitrary data.

Chapter 4

Methodology

The previous chapters provided a conceptual foundation on cryptography and quantum computing, highlighting the real threat posed by future cryptographically relevant quantum computers (CRQCs) to current public-key cryptography. The rapid advancements in quantum computing demand an urgent transition to post-quantum cryptography, as they have the potential to severely weaken or even outright break modern cryptographic algorithms.

As discussed in Chapter 1, the work conducted by this thesis aligns and contributes to two of the strategic objectives defined by the QUBIP project. This chapter explores the methodology adopted for transitioning **Mozilla Firefox**, and consequently its **Network Security Services (NSS)** cryptographic library, to support a quantum-secure Internet browsing. The common denominator in the PQC transition, especially within the scope of this work, is cryptographic agility. The generalization of this concept to PQC transition has already been examined throughout this thesis and, within Mozilla Firefox, this property is applied carefully as it serves as the real enabler to smooth the entire process.

To enable post-quantum cryptography in Mozilla Firefox, a **loadable shallow module** approach is followed. The term "shallow" refers to the fact that the cryptographic implementation is fully decoupled from the module itself with the purpose of serving as an **intermediate layer** that enables NSS to interact with **third-party** implementations. This design choice significantly simplifies the replacement of cryptographic algorithms as the PQC landscape evolves, allowing the application to have a flexible plug-and-play functionality for cryptographic components, and to reduce the overhead required to apply such changes. This shallow module, referred to as the **QUBIP module**, sits between NSS and third-party crypto implementations. The QUBIP module, hereafter referred to as **qryptotoken** following significant modifications to the Kryoptic fork, is implemented as a standard `PKCS#11` software token and can be loaded into any application that generally supports the `PKCS#11` standard.

By default, Firefox uses `softoken` as its internal `PKCS#11` module. However, by design, NSS can load alternative modules. By introducing a loadable `PKCS#11` module dedicated to PQC algorithms, and carefully refactoring the affected components to recognize newly post-quantum primitives, NSS can incorporate post-quantum cryptography without altering its core. In this way, TLS in Firefox can seamlessly negotiate and establish quantum-safe sessions while preserving backward compatibility with classical cryptography and adhering to NSS's design principles of modularity and interoperability.

4.1 QUBIP Module Overview

This section brings together the building blocks introduced earlier and positions them within the architecture of the QUBIP module, whose role, as anticipated, is to enable the underlying support for post-quantum TLS 1.3 in Mozilla Firefox. In the remainder of this section, a high-level overview of the workflow is presented through a client-side TLS 1.3 integration example, in which the various concepts and components introduced so far converge into a unified, practical

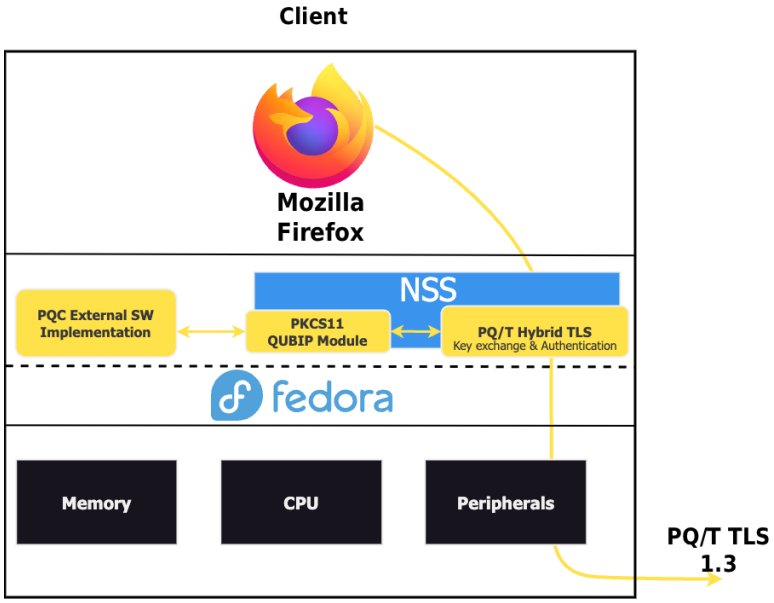

Figure 4.1. Client-side TLS 1.3 Integration Workflow

design. As outlined in the previous chapter (Section 3.1.4), the module's architectural design presented here, builds directly on the prior work of D. Luoma [12]. The initial implementation of the QUBIP module established a three-layer architecture:

- **Cryptoki API Layer**: A complete implementation of `PKCS#11` functions for module initialization, session management, key object management, and mechanism-specific cryptographic operations. The module presents itself to NSS as a standard `PKCS#11` token with defined slots and advertises its supported mechanisms through the standard Cryptoki interface.
- **Software Interoperability Layer**: Serves as the "glue" between the module's Cryptoki API implementation and the external PQC software implementation. It translates `PKCS#11` calls into the appropriate function calls for the external cryptographic libraries.
- **External Cryptographic Libraries**: Contain the actual post-quantum algorithms implementation. All cryptographic operations are delegated here, which keeps the QUBIP module lightweight and easy to extend as new standards emerge.

This separation of concerns makes the module **shallow**: it does not implement cryptography itself, but delegates to trusted third-party libraries. As a result, the module becomes a flexible and pluggable framework for integrating PQC into NSS and, as a direct consequence, into Firefox and any other application built on NSS.

The best way to understand the architecture in practice is to follow its use during a client-server TLS 1.3 handshake, as shown in Figure 4.1. When Firefox initiates a TLS connection, the Necko subsystem delegates secure channel setup to NSS via the PSM interface, which is in charge of configuring TLS sockets. NSS maintains a list of available slots for each supported cryptographic algorithm. When a module is loaded, NSS discovers the mechanisms (i.e., cryptographic algorithms) supported by the corresponding token and maintains a reference to the slot associated with each mechanism. When the user loads the QUBIP module through Firefox's *Security Devices* settings, NSS dynamically links the module, which is implemented as a shared library, and becomes aware of its cryptographic capabilities.

If a PQ/T hybrid TLS 1.3 handshake is requested, NSS can select the QUBIP module as the token that supports the required post-quantum or hybrid algorithm and establish a session with it via the Cryptoki API. The module forwards the requested key exchange operations to the external PQC libraries, receives the results, and returns them to NSS through the same `PKCS#11`

session. NSS integrates these results into its internal TLS handshake routines, ensuring that the negotiated session is both quantum-safe and fully compliant with the TLS 1.3 specification. In this way, the traditional NSS workflow remains intact, with the QUBIP module acting as a plug-in that introduces PQC capabilities. This design preserves backward compatibility with classical cryptography, while enabling a smooth and agile transition to a quantum-safe Internet browsing.

The term QUBIP module was introduced to generally denote the building block responsible for enabling cryptographic agility in Firefox. As discussed in Section 3.2.1, this module has been finalized under the name `qryptotoken`. For the sake of consistency, the module will hereafter be referred to as `qryptotoken` throughout the remainder of this document.

4.2 Next Steps

Luoma's research [12] provided an initial proof of concept demonstrating that cryptographic agility can be achieved via such a module. Luoma's contributions prioritized post-quantum key exchange, specifically enabling the post-quantum component of PQ/T hybrid TLS 1.3 key exchange, a capability already supported internally within vanilla Firefox/NSS. This prioritization also aligned with the broader industry strategy for PQC transition. Key exchange algorithms have been identified as the primary target for early post-quantum adoption for several reasons.

First, key exchange represents the most critical vulnerability in the TLS handshake making it a prime target for the so-called "harvest-now, decrypt-later" attacks, already explored in Section 2.12.2.

Second, the technical properties of post-quantum key encapsulation mechanisms (KEMs) make them comparatively easier to integrate into TLS than other post-quantum primitives. In a TLS connection, the key exchange occurs only once during the handshake, and TLS 1.3 already defines explicit negotiation mechanisms for advertising and selecting supported key exchange algorithms through the `supported_groups` extension (RFC 8446, section 4.2.7 [34]). The singular nature of this operation reduces implementation and testing complexity.

Third, the performance characteristics of the newly standardized post-quantum KEMs are favourable. While post-quantum KEMs involve larger key sizes than classical elliptic curve cryptography, the computational overhead is manageable, and occurs only once per connection. The impact on handshake latency, while measurable, does not drastically alter the user experience.

While the prior work successfully enabled PQ/T hybrid key exchange, it did not address **post-quantum authentication**. The initial implementation relied entirely on traditional signature algorithms for authentication. Migrating to post-quantum digital signatures in TLS is undoubtedly more complex compared to key exchange migration. A typical TLS connection can involve several distinct signatures:

- A signature computed over the entire TLS handshake in the **CertificateVerify** message to provide explicit proof that an endpoint possesses the private key corresponding to its certificate (RFC 8446, section 4.4.3 [34]). The CertificateVerify message also provides integrity for the handshake up to this point. This message is sent by the server after sending its certificate, or by the client in response to an explicit request for authentication from the server.
- Each certificate in a trust chain contains a signature from its issuing Certification Authority. Common three-level chains include the end-entity certificate signed by an intermediate CA, and the intermediate certificate signed by a root CA. Each signature in the chain must be verified up to a trusted root CA.
- TLS certificates may also embed **signed certificate timestamps (SCTs)** as part of Certificate Transparency (RFC 9162 [27]) to avoid mis-issuance of certificates. Usually certificates carry SCTs from at least **two** independent logs.
- If the issuing Certification Authority supports the **Online Certificate Status Protocol (OCSP)** (RFC 6960 [75]), an OCSP response, either obtained directly by querying the

OSCP responder or stapled into the handshake via the **status_request** extension inside the `Certificate` message's `CertificateEntry` structure (RFC 8446, section 4.4.2.1 [34]), adds yet another signature.

4.2.1 Support of PQ/T Hybrid Digital Signatures in TLS 1.3

The present work extended `qryptotoken` and Mozilla Firefox to support post-quantum digital signatures, thereby moving beyond key exchange into the authentication layer and enabling fully quantum-resistant TLS 1.3 client-server connections. This objective was accomplished throughout several steps:

1. Support for NIST post-quantum signature schemes was implemented within `qryptotoken`. Their support required proper integration at the Cryptoki API Layer and the Software Interoperability Layer, covering both signature generation and verification, and linking to external PQC libraries that provide the necessary cryptographic primitives.

2. NSS's `libssl` was adapted to correctly negotiate and utilize post-quantum signature schemes in TLS 1.3. This involved adapting several internal components of NSS, ranging from registering new algorithm OIDs to defining new `PKCS#11` v3.2 PQ-related functions, mechanisms and attribute templates.

3. NSS's `mozpkix` library was extended to support post-quantum signatures during certificate chain validation, thereby anchoring digital trust in post-quantum Public Key Infrastructures. This involved parsing and recognizing post-quantum algorithm OIDs in DER-encoded certificates, and carefully routing signature verification operations through the `PKCS#11` interface so that they were handled by `qryptotoken`.

4. Mozilla Firefox internals were adapted to interact seamlessly with the updated NSS that now supports post-quantum signature schemes.

Together, these steps not only closed a gap left by prior research but also established a practical framework for evaluating the readiness of post-quantum signature schemes in real-world deployments such as Firefox/NSS. The methodology presented thus far brought the discussion full circle to exploring approaches for achieving a smooth cryptographic transition guided by the principle of cryptographic agility. At the same time, particular attention was given to hybridization techniques, specifically composite signature schemes, which were under active discussion in IETF working groups and close to standardization [55]. Ultimately, server-side authentication was not the sole objective; the architectural framework and implementation strategies established here laid the necessary groundwork for supporting post-quantum client authentication in mutual TLS 1.3 scenarios.

4.2.2 Mutual PQ/T Hybrid Authentication in TLS 1.3

The methodological approach described above equally applies to PQ/T hybrid client authentication in the TLS 1.3 handshake. Most changes introduced within NSS for server-side authentication extend naturally to client-side authentication. This is because `libssl`'s TLS machine is largely symmetrical between server and client authentication. Whether generating a signature to be included in the `CertificateVerify` message sent in the second client flight, or verifying a signature received in the same message during the first server flight, the underlying high-level signing and verification API (`cryptohi`) benefits from the same enhanced capability to delegate post-quantum cryptographic operations to an external `PKCS#11` token.

On top, `qryptotoken` presents itself to Firefox as a generic, standards-compliant `PKCS#11` token, exposing post-quantum and hybrid cryptographic capabilities through specific registered `PKCS#11` mechanisms. The token provides purely cryptographic services and is not aware of the TLS protocol, which eliminates the need for substantial modifications to the token. Consequently,

extending support for post-quantum signature algorithms to client-side authentication within the TLS handshake becomes a straightforward task.

However, the process required to configure `qryptotoken` for client authentication is far less trivial. In the context of TLS client authentication, the client is required to possess a private key for signing the handshake transcript, together with a corresponding public key certificate that is transmitted upon explicit request from the server.

First, Firefox does not offer any built-in mechanism for generating cryptographic key pairs through its user interface. Moreover the HTMLKeygenElement DOM interface has been deprecated since Firefox version 69 in 2019 [76, 77]. Second, even assuming that a key pair is already available within `qryptotoken`, the workflow for generating a Certificate Signing Request (CSR), submitting it to a Certification Authority, and importing the issued certificate back into the cryptographic token is a feature not available in Firefox.

Consequently, the focus shifted to extending `qryptotoken` to support these types of operations. In general, management tasks associated with `PKCS#11` tokens can be performed using external open-source tooling. In other scenarios, particularly in the case of secure hardware tokens, proprietary software is provided to perform such tasks. Although open-source tools can be employed with `qryptotoken`, they require adaptations analogous to those necessary in NSS in order to accommodate certain post-quantum cryptographic algorithms.

To enable these essential token management operations, the GnuTLS library's utilities, specifically `p11tool` and `certtool`, were carefully extended. This modification allowed the generation of a key pair within the token, the creation of a CSR to be submitted to a CA, and the subsequent import of a CA-issued certificate, all performed externally to Firefox. For the purpose of TLS 1.3 client authentication, once the `qryptotoken` module is loaded into Firefox, the imported certificate, associated by the ID of its corresponding private key in the token, becomes available for selection during the TLS handshake. These features were essential for enabling the mutual TLS use case of the Internet Browsing Pilot presented in Section 4.2.4.

4.2.3 Enhancements to `qryptotoken`

The first enhancement to `qryptotoken` involved implementing post-quantum and composite signature algorithms while adhering to a *shallow* architectural design. To achieve this, the foundational architectural change was the introduction of a dedicated interface to delegate cryptographic operations to external libraries implementation, ensuring consistent and reproducible behaviour. This established a robust and effective *Software Interoperability Layer*, which had not been fully addressed by previous project contributors.

This interface is realized through "**Adapters**", which are modular code components that act as communication channels between the `PKCS#11` environment and external cryptographic libraries. This design decouples `qryptotoken` from specific implementations, ensuring that neither component needs to know the internal details of the other. This separation also maximizes the reusability of the scaffolding code around the Adapters, which is necessary for interfacing with token's `PKCS#11`-aware context. By design, this scaffolding code is largely similar across different implementations and significantly overlaps between cryptographic operations. This design facilitates easier maintenance and enhances cryptographic agility, particularly when replacing or adding new implementations.

Using this Adapter-based architecture, the implementation targeted the NIST-standardized `ML-DSA` and `SLH-DSA` (across all parameter sets), and the composite algorithm `MLDSA65-Ed25519`. For the composite algorithm, development tracked the evolving Internet Draft standard, with an initial implementation based on draft version 7 later updated to align with the near-final specifications in draft version 13.

In practice, each external cryptographic library requires a dedicated Adapter. Selection among these Adapters is managed through Cargo feature flags, giving users direct control over their dependency graph. This allows users to select implementations based on specific criteria such as performance, trust in the underlying library, or compliance with security standards.

Testing Framework for Cryptographic Implementations

A second major enhancement was the development of a comprehensive unit-testing suite. Rather than focusing solely on the `PKCS#11` interface, including the creation of key pairs or signature operations of a specific algorithm, the test suite was extended to directly verify the correctness of external cryptographic implementations. For this purpose, test vectors were sourced from the **Wycheproof** project [78], a community managed repository providing systematic tests against known attacks, specification inconsistencies, and common implementation flaws.

At the time of this work, Wycheproof vectors were available only for `ML-DSA` and were therefore applied exclusively to that algorithm's adapters. The testing framework, however, has been designed to be easily extended to any other algorithm family once suitable test vectors become available. These type of tests hold particular significance because they enable evaluation of external libraries against a well-established set of adversarial test cases. From a broader integration perspective, this testing framework serves as a practical instrument for evaluating the quality, reliability, and readiness of post-quantum algorithms, implemented in publicly available cryptographic libraries, within the scope of a realistic deployment environment.

4.2.4 Integration and Interoperability Testing

In accordance with the deliverables of the QUBIP project, integration and interoperability tests were conducted to empirically evaluate and validate the functional correctness of `qryptotoken` and the QUBIP version of Mozilla Firefox/NSS enhanced with PQC support.

Integration Tests

The **Quantum-Secure Internet Browsing (IB) Pilot**[1] served as the primary integration test environment for validating the complete end-to-end functionality of the developed PQC-enabled components. This pilot was designed to achieve Technology Readiness Level (TRL) 6, demonstrating the system's capability to operate in a relevant environment that simulates real-world quantum-secure web browsing scenarios.

The pilot implementation, illustrated in Figure 4.2, established a complete TLS 1.3 PQ/T hybrid connection infrastructure between a modified quantum-secure client and server environment. On the client side, the implementation utilized the QUBIP version of Firefox with `qryptotoken` loaded as external `PKCS#11` module. On the server side, an NGINX web server was deployed with the `aurora Provider` (Section 3.1.3) integrated in OpenSSL, providing the corresponding quantum-resistant cryptographic capabilities.

The integration testing framework encompassed multiple authentication scenarios to comprehensively evaluate the system's operational capabilities. The first use case (UC01) examined server-only authentication, representing the standard HTTPS browsing scenario where the server authenticates itself to the client using composite certificates and the key exchange mechanism employed is a PQ/T hybrid one. Instead, the second use case (UC02) addressed also optional TLS 1.3 client authentication. In this scenario, `qryptotoken` provided client-side authentication credentials, enabling Firefox to present quantum-resistant certificates to the NGINX server during the handshake.

Interoperability Tests

Beyond the internal integration testing conducted within the QUBIP infrastructure, additional interoperability tests were performed to validate the system's capability to establish quantum-secure connections with external, publicly available PQC-enabled servers. Interoperability validation was conducted against two independent external platforms.

[1] https://fi.ib.qubip.eu

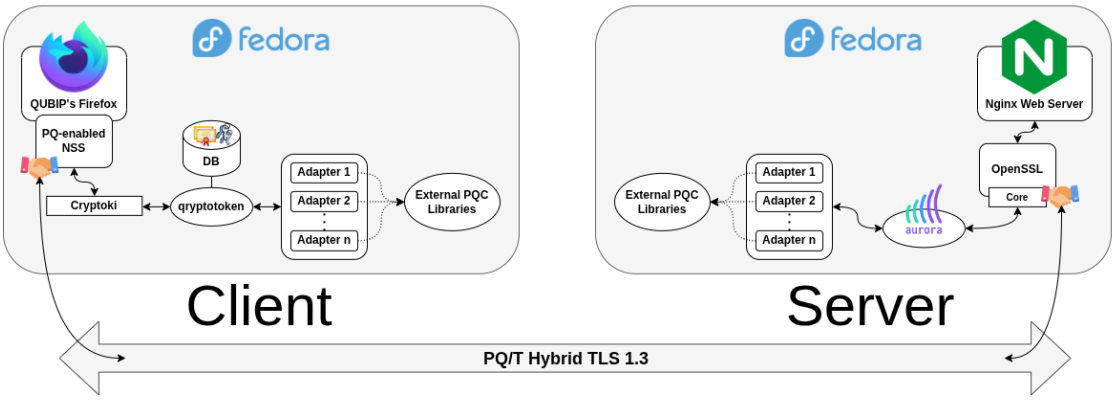

Figure 4.2. Architecture of the Quantum-Secure Internet Browsing Pilot, showing the client environment (QUBIP's Firefox with PQC-enabled NSS using `qryptotoken` via CryptoKi) and the server environment (NGINX with OpenSSL configured to use the `aurora Provider`), enabling a full PQ/T Hybrid TLS 1.3 connection between both endpoints.

The first test environment utilized the **Open Quantum Safe (OQS)** project's public test servers[2]. Their public test servers offer multiple PQC algorithm configurations for TLS 1.3 connections, encompassing both PQ/T hybrid key exchange and pure post-quantum authentication. However, the main limitation encountered was the unavailability of composite signature algorithms matching the draft specification supported by QUBIP's Firefox at the time of testing. Consequently, the OQS-based validation centred only on pure post-quantum cryptographic implementations.

The second external test platform employed was **Cloudflare Research's Post-Quantum Key Agreement** test page[3]. As the name suggests, Cloudflare's test server exclusively supported PQ/T hybrid key exchange algorithms, while authentication mechanisms continued to rely on traditional cryptographic primitives rather than post-quantum signature schemes.

[2] https://test.openquantumsafe.org/

[3] https://pq.cloudflareresearch.com/

Chapter 5

Results

This chapter presents the principal results obtained from the work conducted in this thesis, following the methodological approach described in the previous chapter. The discussion is organized into four main sections: (i) the major modifications introduced to Firefox and NSS to enable support for post-quantum algorithms; (ii) the integration of an Adapter-based architecture to provide agile PQC capabilities within `qryptotoken`; (iii) the integration and interoperability tests conducted to validate the correct operation of these components in realistic deployment scenarios; and (iv) a critical analysis of the challenges and limitations encountered during the implementation process.

5.1 Results of Integrating PQC Support in Firefox and NSS

Before presenting the results obtained and the technical details concerning the integration of post-quantum cryptography into Mozilla Firefox, it is necessary to provide some clarifications. All the features described in this section were implemented on a fork of the upstream Mozilla Firefox repository and are published under the QUBIP GitHub[1] namespace. Henceforward, the term **QUBIP Firefox** will be used to distinguish this modified codebase from the official Mozilla Firefox distribution. QUBIP Firefox is based on Mozilla Firefox Nightly version 146.0a1.

The Network Security Services (NSS) repository targeted by this work is not the upstream NSS project. Mozilla Firefox incorporates its own fork of NSS, which periodically synchronizes with the upstream NSS repository. Consequently, QUBIP's NSS fork within QUBIP Firefox is downstream of Mozilla Firefox's NSS, which itself is downstream of the main NSS project. As a result, changes developed during this work were subject to an inherent delay with respect to the upstream NSS codebase. In particular, some of the patches regarding the support of `ML-DSA` presented in this section were submitted to NSS maintainers and subsequently integrated into later NSS releases. These releases were then incorporated into Mozilla Firefox and ultimately rebased into QUBIP Firefox. Figure 5.1 illustrates this dependency chain.

The implementation of the support for post-quantum algorithms extended across multiple system components, mainly affecting the NSS library and higher-level Firefox components, which were adapted to correctly accommodate the newly introduced algorithm declarations within Firefox's TLS processing and security policy enforcement mechanisms. It is worth noting that many of the modifications introduced for a particular algorithm were also applicable to other algorithms. Given the high level of abstraction and the modular structure of NSS, the addition of support for new parameter sets of the same algorithm or for new algorithms belonging to a different category often involved largely repetitive code, with differences mainly limited to algorithm-specific parameters.

[1] https://github.com/qubip/firefox

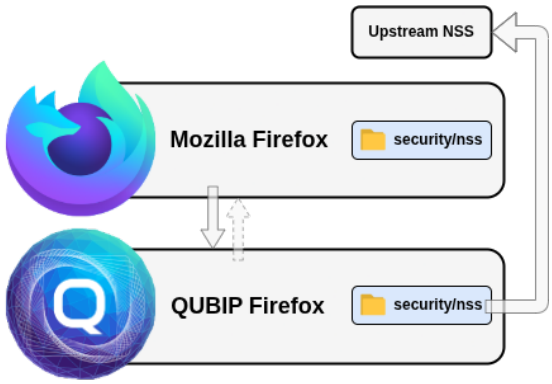

Figure 5.1. QUBIP Firefox Dependency Diagram

The changes introduced are scattered all over the codebase, involving multiple layers of NSS and Firefox internals. A comprehensive, line-by-line analysis of each modification would therefore be impractical and would unnecessarily increase the complexity of this discussion. For this reason, direct references to specific code snippets are intentionally omitted, and the presentation focuses instead on the architectural patterns, design decisions, and functional outcomes that are representative of the overall integration effort.

5.1.1 Supported PQ/Hybrid Algorithms

QUBIP Firefox introduced support for NIST-standardized pure `ML-DSA` and `SLH-DSA` schemes, as well as an experimental `composite ML-DSA` scheme, `MLDSA65-Ed25519`, included for evaluation prior to standardization.

`ML-DSA` was implemented across its three standardized parameter sets:

- `ML-DSA-44` (NIST Security Level 2)
- `ML-DSA-65` (NIST Security Level 3)
- `ML-DSA-87` (NIST Security Level 5)

`SLH-DSA` was implemented only through its six SHAKE-based parameter sets, encompassing both the relatively *small* ("s") and *fast* ("f") signature variants:

- `SLH-DSA-SHAKE-128s/f` (NIST Security Level 1)
- `SLH-DSA-SHAKE-192s/f` (NIST Security Level 3)
- `SLH-DSA-SHAKE-256s/f` (NIST Security Level 5)

An overview of all the algorithms currently supported by QUBIP Firefox and `qryptotoken` is presented later in Section 5.2.4.

5.1.2 Core Infrastructure Extensions

The OID management system was comprehensively extended to recognize post-quantum algorithm identifiers. A modular design approach was adopted, isolating the targeted OID declarations within a dedicated header file, which was subsequently incorporated into NSS's main OID registry via preprocessor inclusion. Each OID entry associated a unique numeric identifier, allocated from either the Computer Security Objects Register (CSOR) or the IANA PKIX namespace, with a human-readable descriptor, the corresponding `PKCS#11` mechanism, and a set of policy flags.

These flags define whether a certain algorithm, represented by its OID tag, is permitted across different contexts, ranging from global, low-level cryptographic operations to its use in the TLS handshake or within certificates as a valid signature scheme.

Algorithm-specific constant definitions were defined in dedicated header files that encapsulate the fixed-size parameter specifications for each security level, including private key, public key, and signature sizes. These constants were referenced throughout the codebase to allocate buffers and validate key or signature material.

The key type system within `cryptohi` was extended with new enumerations and corresponding private and public key structures to represent post-quantum keys. All structures defined the key material as a raw data blob. Instead, `ML-DSA` and `SLH-DSA` structures incorporated a parameter set field represented by an OID tag, whereas the `MLDSA65-Ed25519` structure omitted this field, due to its unambiguous algorithm identifier (i.e., no other parameter sets exist).

5.1.3 PKCS#11 Integration

A critical aspect of the implementation involved extending NSS's `PKCS#11` abstraction layer to accommodate the newly introduced post-quantum algorithms. A dedicated header file was introduced to define **vendor-specific mechanism** identifiers within the **NISEC namespace** (`0x4E534543`). This vendor-defined approach, while diverging from the mechanism identifiers specified by the `PKCS#11` standard, was adopted at the outset of this implementation work because no available `PKCS#11` version at that time provided native support for either `ML-DSA` or `SLH-DSA`.

Although version 3.2 of the `PKCS#11` standard was published during the course of this work, vendor-defined mechanisms continued to be used in order to clearly distinguish the experimental nature of the implementation from potential future upstream support. Moreover, the composite scheme `MLDSA65-Ed25519` necessarily required a vendor-defined mechanism identifier, as its standardization process was still ongoing and no `PKCS#11` version currently provided support for this scheme.

Notably, support for `ML-DSA` was later added upstream as a result of contributions made during this thesis in **Bug 1983320**[2]. Following a subsequent rebase of QUBIP Firefox, the implementation was adapted to use the standardized mechanism identifiers where available. However, the vendor-defined mechanisms were intentionally retained and duplicated alongside the upstream `ML-DSA` definitions. Conditional compilation directives (`DISABLE_QUBIP_CHANGES`) were strategically placed throughout the codebase to clearly demarcate these vendor-specific extensions, thereby facilitating maintenance and enabling a controlled migration path toward fully standardized identifiers.

The key management infrastructure in `pk11wrap` was extended to support the newly introduced `PKCS#11` mechanisms. Mechanism-specific base templates were defined in accordance with the attributes specified by the `PKCS#11` v3.2 standard. The key pair generation logic was augmented to extract the parameter set from the input parameters and to configure the corresponding `PKCS#11` templates based on the requested algorithm.

`ML-DSA` and `SLH-DSA` followed a common integration pattern, each defining an algorithm-specific parameter type as prescribed by the standard. In contrast, the composite scheme omitted this parameter configuration step due to its single-algorithm nature. Public key import functionality, required for processing remote public keys needed for signature verification, such as during the TLS handshake or certificate chain validation, was extended accordingly.

Dedicated slot lists were introduced for post-quantum algorithm support, enabling specialized getter functions to route mechanism queries to the appropriate slot list. This functionality permitted the discovery of tokens capable of performing post-quantum cryptographic operations.

[2] https://bugzilla.mozilla.org/show_bug.cgi?id=1983320

5.1.4 TLS Protocol Integration

The TLS implementation in NSS's `libssl` was modified to recognize and negotiate post-quantum signature schemes. The TLS signature scheme enumeration, commonly referred to as TLS "*sigalgs*", was extended with IANA-assigned identifiers for `ML-DSA` and `composite ML-DSA` algorithms, and the corresponding authentication type enumerations, required for client and server authentication in the handshake, were updated accordingly. Post-quantum algorithms were incorporated into the default signature scheme preference list and positioned ahead of traditional signature algorithms. However, the ordering within this list did not directly determine algorithm prioritization, as the effective selection of signature schemes was governed by Firefox's security policies during the TLS configuration phase. In QUBIP Firefox, the default list of enabled TLS signature algorithms applied to NSS was configured to always prioritize composite over pure implementations.

The signature generation and verification pipeline was extended to support post-quantum algorithms while preserving the existing TLS handshake control flow. A fundamental difference from previously supported signature algorithms is that post-quantum schemes do not follow the conventional *hash-and-sign* paradigm, as message digest computation is performed internally by the algorithm.

The existing high-level signing and verification interfaces provided by NSS's `cryptohi` component did not initially accommodate this operational model. Consequently, during the early stages of the implementation, a temporary workaround bypassed the standard interfaces and triggered signature verification without externally computing the handshake transcript digest. Through collaboration with NSS maintainers, these interfaces were subsequently extended upstream to natively support post-quantum algorithms. The updated interfaces were later integrated into QUBIP Firefox, and consequently into the NSS version included within it.

In general, the majority of the required changes were confined to lower-level components such as the `PKCS#11` abstraction layer and higher-level cryptographic interfaces, thereby minimizing modifications to the handshake logic itself. All relevant code paths responsible for mapping between object identifiers (OIDs) and TLS signature scheme identifiers were updated to recognize and process post-quantum algorithms within the TLS protocol. Finally, the TLS handshake implementation was slightly modified to correctly advertise and accept supported post-quantum signature schemes during negotiation.

Nevertheless, support for `SLH-DSA` was intentionally restricted to certificate chain validation. Its use in the TLS 1.3 handshake was deliberately avoided, as `SLH-DSA` signatures demonstrate significantly higher computational costs and substantially larger signature sizes compared to other schemes, rendering them impractical for typical TLS handshake use cases. Although the corresponding IANA TLS codepoints for `SLH-DSA` were registered in the `libssl` signature scheme enumeration, they were intentionally excluded from the default list of negotiable TLS signature algorithms. Consequently, `SLH-DSA` support was implemented primarily at the `PKCS#11` level to enable verification of certificate signatures, particularly for root Certification Authorities that may adopt `SLH-DSA` as a conservative choice for long-term cryptographic security.

5.1.5 Certificate Validation Infrastructure

The Mozilla PKIX library (`mozpkix`), responsible for X.509 certificate path validation, required only limited modifications to support post-quantum signature algorithms. These changes followed a systematic and generalizable pattern that was consistently applied across `ML-DSA`, `SLH-DSA`, and `composite ML-DSA` schemes. The certificate validation logic was extended to correctly decode DER-encoded post-quantum public keys and signature algorithm OIDs in accordance with the relevant IETF draft specifications establishing their use within the PKIX framework.

A central aspect of this integration involved extending the internal enumeration of supported public key algorithms to explicitly recognize post-quantum schemes. This extension was essential during certificate parsing and signature algorithm decoding, as it enabled the correct mapping between signature algorithm identifiers, their corresponding public key algorithms, and the associated digest algorithms. To accommodate post-quantum algorithms that perform internal hashing,

a novel "no_digest" value was introduced. By formally identifying the public key algorithm, the validation engine could route signature verification requests to algorithm-specific verification paths introduced for post-quantum support (i.e., new APIs). Given the public key algorithm identifier of the certificate issuer, the implementation could then determine the appropriate PKCS#11 mechanism based on the relevant parameter set, where applicable, and any algorithm-specific policy constraints, before invoking the generic NSS verification interface that ultimately delegates the operation to the PKCS#11 layer.

TrustDomain Interface Integration

In Firefox, the TrustDomain interface defines the contract between certificate validation policy (what must be verified) and cryptographic services (how verification is performed). Within QUBIP Firefox, this abstract interface was extended to support post-quantum signature verification, and all concrete implementations of TrustDomain were updated to conform to this extended interface.

Most notably, NSSCertDBTrustDomain, which serves as the primary trust domain for X.509 certificate path validation within Firefox, was enhanced to incorporate these new verification paths. Extending this critical component was of paramount importance, as it serves as the central component through which most TLS, S/MIME, and code-signing certificate validation is performed, also handling revocation checks (OCSP, CRLite), trust decisions, and Certificate Transparency verification.

Certificate Viewer

Although not part of the initial objectives, Firefox's certificate viewer component (i.e., certviewer) was updated to correctly display certificates employing post-quantum cryptographic algorithms. The PKI JavaScript library was extended with new classes implementing the ASN.1 parsing logic specific to each supported post-quantum algorithm. The public key class was enhanced to recognize post-quantum algorithm OIDs and to instantiate the corresponding public key representations accordingly.

The user interface was improved to present post-quantum public keys by displaying their raw key material as hex-encoded byte sequences. A further enhancement introduced an automated **security level indicator**, which maps post-quantum signature algorithm identifiers to their corresponding NIST security levels, thereby providing users with an immediate and standardized assessment of the cryptographic strength of the signature algorithm used for signing the certificate.

5.1.6 Post-Quantum Algorithm Policy Enforcement

Although pure ML-DSA signature schemes were fully implemented and supported within the NSS version shipped with QUBIP Firefox, they were intentionally disabled by default through a layered policy enforcement mechanism. This design choice reflects current post-quantum transition guidance, including European recommendations [79, p. 6], which generally favour hybrid or composite post-quantum constructions over the immediate deployment of standalone post-quantum algorithms. It also aligns with the prevailing conservative security perspective that prioritizes gradual adoption of post-quantum cryptography while maintaining compatibility with well-established classical primitives.

NSS-Level Policy At the NSS level, all ML-DSA object identifiers are marked as disabled for every possible usage context by default. This is achieved through the algorithm policy framework, which allows cryptographic primitives to be selectively enabled or restricted based on policy flags. By default, ML-DSA algorithms are excluded from signature generation and verification, certificate validation, and TLS usage unless explicitly permitted by higher-level policy decisions. This ensures a *secure-by-default* configuration in which experimental algorithms cannot be inadvertently negotiated or accepted.

Firefox-Level Policy At the Firefox integration level, this policy is further reinforced through explicit controls governing TLS and certificate validation behaviour. A dedicated configuration option (`security.tls.enable_mldsa`) allows users to explicitly enable `ML-DSA` support for experimentation purposes. This option is disabled by default and, when unset, prevents `ML-DSA` from being advertised or negotiated as a TLS signature scheme and causes certificates signed with `ML-DSA` to be rejected during validation. When enabled, the corresponding NSS policy flags are dynamically relaxed to permit `ML-DSA` usage both in TLS handshakes and in certificate signature verification.

This mechanism differentiates between pure post-quantum schemes and composite constructions. While pure `ML-DSA` remains opt-in, `MLDSA65-Ed25519` is enabled by default and preferred during TLS negotiation. Composite signatures provide stronger security assurances during the transition period, as they remain secure unless both the classical and post-quantum components are simultaneously compromised. This approach minimizes the risk to end users in the event that a post-quantum algorithm or its implementation is later found to be vulnerable.

Overall, this policy enforcement strategy ensures that QUBIP Firefox maintains forward compatibility with post-quantum cryptography while adhering to conservative deployment principles. It allows controlled experimentation with pure post-quantum algorithms while promoting safer composite constructions for default use.

5.2 Results of Integrating PQC Support in `qryptotoken`

The primary enhancement achieved within `qryptotoken` consisted in the establishment of a robust **Software Interoperability Layer** based on a modular Adapter-based architecture. The selection of supported algorithms was aligned with the set of algorithms currently supported by QUBIP Firefox to ensure compatibility across the ecosystem. The implementation followed a phased approach: first, the targeted post-quantum signature algorithms were integrated into `qryptotoken` by defining the required `PKCS#11` mechanisms and enabling cryptographic services through newly developed Adapters; subsequently, QUBIP Firefox was adapted to recognize and utilize the newly available signature schemes (i.e., mechanisms) through the Cryptoki API, thereby enabling their use in TLS 1.3 handshakes for both server-only and mutual authentication scenarios, as well as for the verification of certificates signed with these schemes.

5.2.1 Adapter Architecture

Each cryptographic algorithm family was supported through dedicated Adapters that provided an abstraction layer over the corresponding third-party library implementations. The external implementation selected for `ML-DSA`, in all its three variants, was **libcrux** [68], a formally verified cryptographic library written in Rust. Similarly, support for `SLH-DSA`, in its six SHAKE variants, was provided through the **RustCrypto** library [80], which offers pure Rust implementations of cryptographic primitives.

Support for `composite ML-DSA` was implemented in a modular manner in order to accommodate multiple variants. However, support was limited to `MLDSA65-Ed25519`, which was initially developed following draft version 7 of the specification and later updated to align with draft version 13 as the standard approached finalization and received official OID registration by IANA. The composite implementations required coordination between two distinct cryptographic primitives, `ML-DSA-65` and `Ed25519`, both provided by the **libcrux** library. Two distinct Adapters were therefore implemented to reflect these two developmental stages. Although retaining the legacy implementation may appear unnecessary from an interoperability perspective, it was preserved as an opt-in Adapter to enable interaction with older versions of QUBIP Firefox that relied on vendor-defined OIDs registered under the Entrust namespace in draft version 7. More generally, its inclusion served to demonstrate the modularity and agility afforded by this shallow architecture.

Unified Abstraction Layer

Each cryptographic algorithm family was represented through a set of unified abstractions that translate the various `PKCS#11` cryptographic services, such as key pair generation, signing, and verification, into underlying implementations integrated by different Adapters. The implementation leveraged Rust's trait system to ensure type safety when delegating cryptographic operations to external libraries. Specifically, it provided type-safe wrappers for public keys, private keys, and signatures, along with encoding and decoding routines to translate between the `PKCS#11` byte-level representations and the internal formats of the underlying libraries, across all algorithm variants and parameter sets. This design ensured a uniform abstract interface, regardless of which Adapter provided the actual cryptographic primitives.

The Adapter selection mechanism was implemented using Cargo feature flags, which enable conditional compilation of specific cryptographic backends. At compile time, users can specify which Adapter to include for a particular algorithm by enabling the corresponding feature.

A critical aspect of the Adapter architecture is the consistent handling of errors across all external libraries. Each Adapter conformed to a custom error model that mapped library-specific errors to `qryptotoken`'s internal error representation. These error types converged in the unified abstract interface, which processed them appropriately and converted them into the corresponding `PKCS#11` return values defined by the standard, such as function failures, signature or key length range violations, and invalid attribute values. This design ensured that error conditions were correctly propagated through the token interface.

Adapter Contract and Interface Requirements

To ensure interoperability across different cryptographic libraries, each Adapter satisfied a well-defined contract, specifying the types, methods, constants, and functions that must be provided for each algorithm family. This contract includes:

- **Type definitions**: Each Adapter defines representations for public keys, private keys, and signatures that are compatible with the external library's format.
- **Encoding and decoding methods**: Each Adapter provides functionality to serialize cryptographic objects to byte arrays and deserialize them from such representations, with appropriate error handling for malformed inputs.
- **Cryptographic operations**: Adapters implement key pair generation, signature generation, and signature verification, along with any algorithm-specific variants such as context-aware signing and configurable hedge, or deterministic, signing modes.
- **Size constants**: Each parameter set exposes compile-time constants for public key, private key, and signature sizes, enabling controlled buffer allocation and input validation.

The implementation of these requirements is verified through Rust's type system, ensuring that missing or incorrectly typed components result in compilation errors rather than runtime failures.

Beyond basic signature generation and verification, Adapters implementing `ML-DSA` or `SLH-DSA` supported advanced features required by the `PKCS#11` v3.2 specification. Specifically, the implementation provides context-aware signing capabilities and configurable randomization modes, allowing users to select between **hedged** (randomized) and deterministic signatures where applicable.

This Adapter-based architecture significantly enhances maintainability by centralizing the scaffolding code required for `PKCS#11` integration. The unified abstractions and their associated implementations follow a consistent pattern that can be replicated when adding support for new algorithms. This pattern-based approach reduces the complexity overhead on developers and minimizes the risk of introducing subtle bugs during extension or modification.

5.2.2 PKCS#11 Integration

Since `qryptotoken` inherits the overall architectural structure of Kryoptic, support for new algorithms had to follow the same general design principles. Before developing any Adapter-based interface to offload cryptographic operations to external implementations, it was necessary to adhere to the existing architectural model adopted for other algorithms. This was achieved by examining the implementations of previously supported algorithms and adapting their structure to accommodate post-quantum primitives.

One of the key steps in this process was the development of **object factories**, which serve as the authoritative components for defining the creation, validation, and lifecycle management of cryptographic key objects. These factories encapsulate the rules that determine which attributes are required, optional, or prohibited for each object type.

Two distinct factory pairs were implemented: one for public key objects and one for private key objects. These factories reused common functionality from base implementations that handled standard `PKCS#11` object attributes and were extended to include the algorithm-specific attributes required for post-quantum schemes. The `ML-DSA` or `SLH-DSA` public key factory, for instance, defined `CKA_PARAMETER_SET` as a required, unchangeable attribute that must be specified during object creation, along with `CKA_VALUE` containing the encoded public key material. Similarly, the private key factory mandated the same `CKA_PARAMETER_SET` attribute and required `CKA_VALUE` to be marked as sensitive and explicitly non-extractable, thereby ensuring that private key material could not be exported from the token once created. In line with `qryptotoken`'s design goals, this behaviour mirrors that of a **hardware secure token**, in which private keys never leave the token boundary.

An important aspect of the factory implementation was the support for both key import and key generation workflows. When importing existing keys, the factories validated the provided key material to ensure it matched the declared parameter set, if applicable, rejecting keys with incorrect sizes or invalid structure. For key generation, the factories coordinated with the mechanism implementation to invoke the appropriate Adapter functions, then populated the newly created key objects with the generated material and automatically set derived attributes such as `CKA_KEY_TYPE` and default operational attributes (`CKA_SIGN` for private keys, `CKA_VERIFY` for public keys).

For private keys, additional considerations applied: `ML-DSA` private keys could be stored either as the complete signing key or as a 32-byte seed from which the key could be deterministically regenerated, while `SLH-DSA` private keys always stored the full key material. The validation routines checked these constraints and rejected malformed key objects with appropriate error codes.

Mechanism Implementation

Complementing the object factories, mechanism implementations defined the cryptographic operations that could be performed using post-quantum keys. Each algorithm family required the registration of two mechanisms: a base signing/verification mechanism and a key generation mechanism. The mechanism registration process associated each mechanism identifier with a mechanism information structure that specified supported key sizes, operation flags, and parameter requirements.

When an application initiates a signing or verification operation through the Cryptoki interface, the mechanism implementation performs several critical validation steps before delegating to the cryptographic Adapters. The implementation verifies that the provided key object is of the correct type, belongs to the expected key type family, and possess the appropriate operational attribute. Only after these checks succeeds does the mechanism create an operation context that maintains state throughout the multi-phase signing or verification process.

The mechanism implementations supported advanced parameters through the `CK_MECHANISM` structure's parameter pointer. A notable enhancement involved the implementation of advanced signature parameters through the `CK_SIGN_ADDITIONAL_CONTEXT` structure. This mechanism parameter enabled applications to specify optional context strings (limited to 255 bytes) and select

between deterministic and hedged signing modes. The context string, when provided, was incorporated into the message representative according to the FIPS 204 and FIPS 205 specifications for `ML-DSA` and `SLH-DSA` respectively.

By default, the implemented mechanisms operate in hedged mode (`CKH_HEDGE_PREFERRED`), where signing operations incorporate fresh randomness to provide robustness against side-channel attacks. Applications requiring deterministic signatures for reproducibility can explicitly request deterministic operation (`CKH_DETERMINISTIC_REQUIRED`), while `CKH_HEDGE_REQUIRED` ensures that hedging is always applied regardless of application preferences.

The signing and verification operations were structured as multi-phase processes, following the standard `PKCS#11` pattern of initialization, update, and finalization. This design accommodated both single-part signatures, where the entire message is provided at once, and multi-part signatures, where the message is provided incrementally. The implementation maintained internal state to track the operation's progress and accumulated message data until finalization.

Parallel to QUBIP Firefox, vendor-defined mechanism identifiers and key types were established under the NISEC namespace (`0x4E534543`). The mechanism identifiers were derived by combining the vendor-defined base constant `CKM_VENDOR_DEFINED` with the NISEC vendor identifier, resulting in `CKM_NISEC`. Similarly, key type identifiers were defined using `CKK_NISEC`. An overview of all the algorithms currently supported by QUBIP Firefox and `qryptotoken` is presented later in Section 5.2.4.

5.2.3 Validation and Testing

To ensure correct integration at the `PKCS#11` interface level, unit tests were conducted to validate basic token functionalities. These tests verified the proper operation of core `PKCS#11` mechanisms including key generation, key import, signing, and verification operations for all supported algorithm variants. The unit tests ensured that the object factories correctly validated key material, that mechanism implementations properly handled operation contexts, and that error conditions were appropriately detected and reported through standard `PKCS#11` return codes.

On the other hand, to validate the correctness of the cryptographic implementations, an extensive testing framework was developed based on the **Wycheproof** project's test vectors as outlined in Section 4.2.3. The framework systematically evaluated `ML-DSA` Adapters against known-answer tests, edge cases, and adversarial inputs designed to detect common implementation vulnerabilities.

To facilitate the use of Wycheproof test vectors, upstream contributions were made to the `wycheproof-rs` crate [81], a Rust library that deserializes Wycheproof JSON-formatted test vectors and provides type-safe access to cryptographic test cases. The `wycheproof-rs` library automates the continuous integration of new test vectors, including their deserialization and the decoding of hexadecimal and Base64-encoded data, thereby providing a robust foundation for testing cryptographic implementations in Rust. These contributions enabled the integration of Wycheproof test vectors not only in `qryptotoken` but also in other QUBIP projects, such as the `Aurora Provider`, which follows the same shallow Adapter-based architecture. This unified testing approach ensured consistency across the QUBIP ecosystem and facilitated the validation of different cryptographic backends through a common interface.

The testing bench covered both positive test cases and negative test cases. For negative tests, the framework verified not only that operations failed as expected but also that the specific error conditions matched the anticipated failure modes indicated by the test vectors' flags. This level of granularity enabled the detection of subtle implementation defects that might otherwise remain undetected. For instance, the `ML-DSA` Adapters wrapping the libcrux library demonstrated vulnerabilities in the detection of malformed private keys during decoding and import operations, specifically when either the `s1` or `s2` vector exceeded the valid range as defined by the internal construction specification in FIPS 204 [6].

The Wycheproof integration was implemented through parameterized test functions that operated generically across different algorithm variants. Type-level abstractions allowed the same test logic to be applied to all the parameter sets without code duplication, improving maintainability and ensuring consistent test coverage across all supported parameter sets.

5.2.4 Targeted Post-Quantum and Hybrid Algorithms Supported

This section reports all the currently supported post-quantum/hybrid algorithms exposed by latest release of `qryptotoken` (v0.4.1), and actively engaged by QUBIP Firefox in the TLS 1.3 handshake and certificate chain validation.

Table 5.1 summarizes the supported digital signature algorithms alongside their respective `PKCS#11` identifiers and corresponding TLS 1.3 signature scheme codepoints, as registered in the IANA TLS SignatureScheme[3] registry. For completeness, the key exchange groups supported and addressed by prior work relevant to this thesis are reported in Table 5.2.

Currently, `x25519mlkem768` is natively supported in Firefox as the only PQ/T hybrid key exchange group in the TLS 1.3 handshake. The hybrid key exchange is handled internally by NSS, with its own application-level combiner implementation to delegate the underlying operations (i.e., traditional `X25519` and post-quantum `ML-KEM-768`) to `PKCS#11` tokens. The IANA TLS Supported Groups[4] identifier for this group is `0x11EC (4588)`, and upstream Firefox versions already advertise this codepoint in the TLS 1.3 Client Hello.

In terms of `PKCS#11` support, `ML-KEM-768` is defined in upstream Firefox (and NSS) as a vendor-defined mechanism in the NSS namespace and exposed by `qryptotoken` under the same namespace to be actively used in the key exchange process.

[3] https://www.iana.org/assignments/tls-parameters/tls-parameters.xhtml#tls-signaturescheme

[4] https://www.iana.org/assignments/tls-parameters/tls-parameters.xhtml#tls-parameters-8

Algorithm	OID	IANA TLS Sig. ID	CKM_* (Hex)	CKM_*_KEY_PAIR_GEN	CKK_*(Hex)	CKP_* (Hex)	PQ/T Hybrid
ML-DSA-44	2.16.840.1.101.3.4.3.17	0x0904 (2308)	0xCE534546	0xCE534545	0xCE534544	0x00000001	Pure-PQC
ML-DSA-65	2.16.840.1.101.3.4.3.18	0x0905 (2309)	0xCE534546	0xCE534545	0xCE534544	0x00000002	Pure-PQC
ML-DSA-87	2.16.840.1.101.3.4.3.19	0x0906 (2310)	0xCE534546	0xCE534545	0xCE534544	0x00000003	Pure-PQC
MLDSA65-Ed25519	1.3.6.1.5.5.7.6.48	0x090B (2315)	0xCE53454C	0xCE53454B	0xCE53454A	None	Composite
SLH-DSA-SHAKE-128S	2.16.840.1.101.3.4.3.26	0x0917 (2327)*	0xCE534549	0xCE534548	0xCE534547	0x00000002	Exempt
SLH-DSA-SHAKE-128F	2.16.840.1.101.3.4.3.27	0x0918 (2328)*	0xCE534549	0xCE534548	0xCE534547	0x00000004	Exempt
SLH-DSA-SHAKE-192S	2.16.840.1.101.3.4.3.28	0x0919 (2329)*	0xCE534549	0xCE534548	0xCE534547	0x00000006	Exempt
SLH-DSA-SHAKE-192F	2.16.840.1.101.3.4.3.29	0x091A (2330)*	0xCE534549	0xCE534548	0xCE534547	0x00000008	Exempt
SLH-DSA-SHAKE-256S	2.16.840.1.101.3.4.3.30	0x091B (2331)*	0xCE534549	0xCE534548	0xCE534547	0x0000000A	Exempt
SLH-DSA-SHAKE-256F	2.16.840.1.101.3.4.3.31	0x091C (2332)*	0xCE534549	0xCE534548	0xCE534547	0x0000000C	Exempt

Table 5.1. Post-quantum and composite digital signature algorithms mapped to their corresponding object identifiers (OIDs), IANA TLS SignatureScheme identifiers, and `PKCS#11` constants. In the `PKCS#11` columns, the symbol * denotes a placeholder that is replaced by the specific algorithm name for which the constant is defined. The final column indicates whether each algorithm is a pure post-quantum scheme or a hybrid/composite construction. `SLH-DSA` entries marked with * indicate that are currently disabled in TLS.

Algorithm	CKM_* (Hex)	CKM_*_KEY_PAIR_GEN	CKK_* (Hex)	CKP_* (Hex)	Notes
ML-KEM-768	0xCE534381	0x0xCE534380	0xCE534356	0xCE534352	NSS vendor-defined

Table 5.2. Post-quantum key exchange groups mapped to their corresponding `PKCS#11` constants, which are vendor-defined within the NSS namespace.

5.3 Results of Integration and Interoperability Testing

This section presents the results of the integration and interoperability tests described in Section 4.2.4. The experiments assessed the correct operation of `qryptotoken` within QUBIP Firefox through end-to-end tests performed in the Quantum-Secure Internet Browsing (IB) Pilot, as well as through interoperability tests conducted against external PQC-enabled test servers. Both sets of experiments focused on establishing a full PQ/T hybrid TLS 1.3 connection.

Both test scenarios followed an identical initial configuration procedure. A clean Fedora 42 virtual machine was provisioned with a preconfigured `qryptotoken` database and its compiled shared object file (`libqryptotoken_pkcs11.so`). QUBIP Firefox was deployed through a **Flatpak** remote repository. Following each incremental production-ready update of QUBIP Firefox, a Flatpak archive was constructed from the QUBIP Firefox source-compiled binaries and installed using the Flatpak package manager pre-installed on Fedora.

The `qryptotoken` module was loaded into QUBIP Firefox through the *Security Devices* settings by selecting the corresponding shared object file. Upon entry of the PIN configured during token initialization, QUBIP Firefox's NSS should be able to recognize and keep track of the post-quantum cryptographic capabilities advertised by `qryptotoken`. At this point, QUBIP Firefox should be capable of establishing TLS 1.3 connections with server authentication using post-quantum/hybrid algorithms and, as demonstrated in the IB Pilot, should also be able to authenticate itself using post-quantum/hybrid credentials. The tests conducted were designed to empirically validate these assumptions.

Quantum Secure Internet Browsing (IB) Pilot

As previously introduced, the Quantum Secure IB Pilot component relevant to the TLS use-case comprised two scenarios. The only difference between **UC01** and **UC02** lied in the optional client authentication requested by the server in UC02. In both scenarios, an NGINX web server instance was configured to support `x25519mlkem768` group for key exchange, and `MLDSA65-Ed25519` signature algorithm for authentication.

The certification path of the server certificate was established through the QUBIP Certification Authority (CA). At the time these tests were conducted, the public key infrastructure (PKI) adopted for the experiments was rooted in `SLH-DSA-SHAKE-256S` (QUBIP Root CA), which was used to sign the intermediate TLS CA certificates. The TLS intermediate CA, which in turn issued end-entity certificates, such as the one used by the server in these scenarios, employed `composite ML-DSA`-based signatures.

Figure 5.2 shows a successful connection established using `x25519mlkem768` key exchange group and `MLDSA65-Ed25519` signature algorithm against the test server running on port 22001 of `fi.ib.qubip.eu`. This test completed successfully, demonstrating that QUBIP Firefox, with `qryptotoken` loaded, was capable of negotiating post-quantum and hybrid algorithms in a connection with server-side authentication only, while also recursively verifying all signatures involved in the server's certificate trust chain. The server deployed for the IB pilot utilized an NGINX module developed by Tampere University to dynamically retrieve information related to the established connection and embed it on the HTML page rendered by Firefox following a successful handshake. Previously, connection details were inspected using Firefox's Developer Tools.

Figure 5.3 shows a successful connection established with client authentication using the same algorithms against the test server running on port 22002 of `fi.ib.qubip.eu`. In this scenario, an additional step was required. During client authentication, the user was prompted to select a client certificate for authentication from the list of certificates accessible to Firefox. The available certificates were identified by their serial numbers, with their corresponding details displayed in the selection dialog.

The server was configured to accept only certificates rooted in the trusted QUBIP Client TLS PKI. The client certificate used in this experiment was issued by the QUBIP TLS Certification Authority, following the submission of a certificate signing request (CSR) to the QUBIP TLS CA. The certificate was then loaded into `qryptotoken` as a certificate object, with an identifier

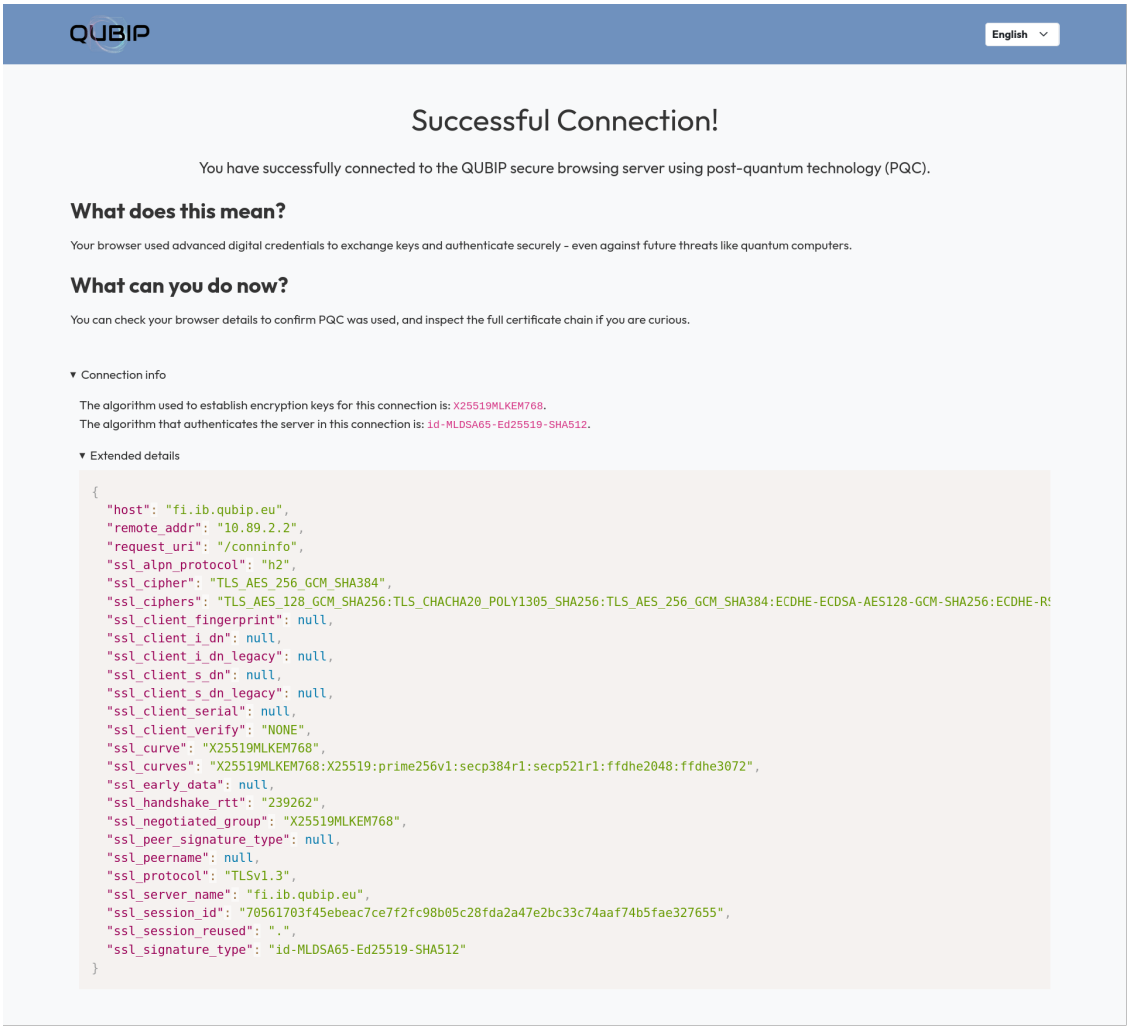

Figure 5.2. Successful PQ/T hybrid TLS 1.3 connection using `x25519mlkem768` for key exchange and `MLDSA65-Ed25519` for authentication against the IB Pilot test server running on port 22001.

matching that of the corresponding private key. This identifier correspondence ensured that QUBIP Firefox prompted the user only with certificates stored in the token for which the user actually possessed the associated private key.

The successful connection demonstrated the ability of QUBIP Firefox to perform TLS 1.3 client authentication using post-quantum/hybrid cryptographic algorithms.

Finally, Figure 5.4 illustrates the server certificate chain up to the trusted QUBIP Root CA, as displayed in the enhanced Certificate Viewer component of QUBIP Firefox (i.e., `certviewer`).

Interoperability Tests

The Open Quantum Safe (OQS) project provides over 700 publicly accessible test servers offering a wide range of post-quantum cryptographic configurations, combining different key exchange and signature algorithms to evaluate both standardized and experimental implementations. The interoperability tests conducted against this external PQC-enabled infrastructure aimed to assess the same hybrid key exchange mechanism, `x25519mlkem768`, and pure `ML-DSA` signature algorithm across all supported parameter sets within the TLS 1.3 handshake.

All `composite ML-DSA` configurations currently offered by OQS are based on earlier revisions of the specification and therefore do not conform to the draft version adopted in this thesis. For this reason, testing of `MLDSA65-Ed25519` was excluded from the interoperability tests. It is

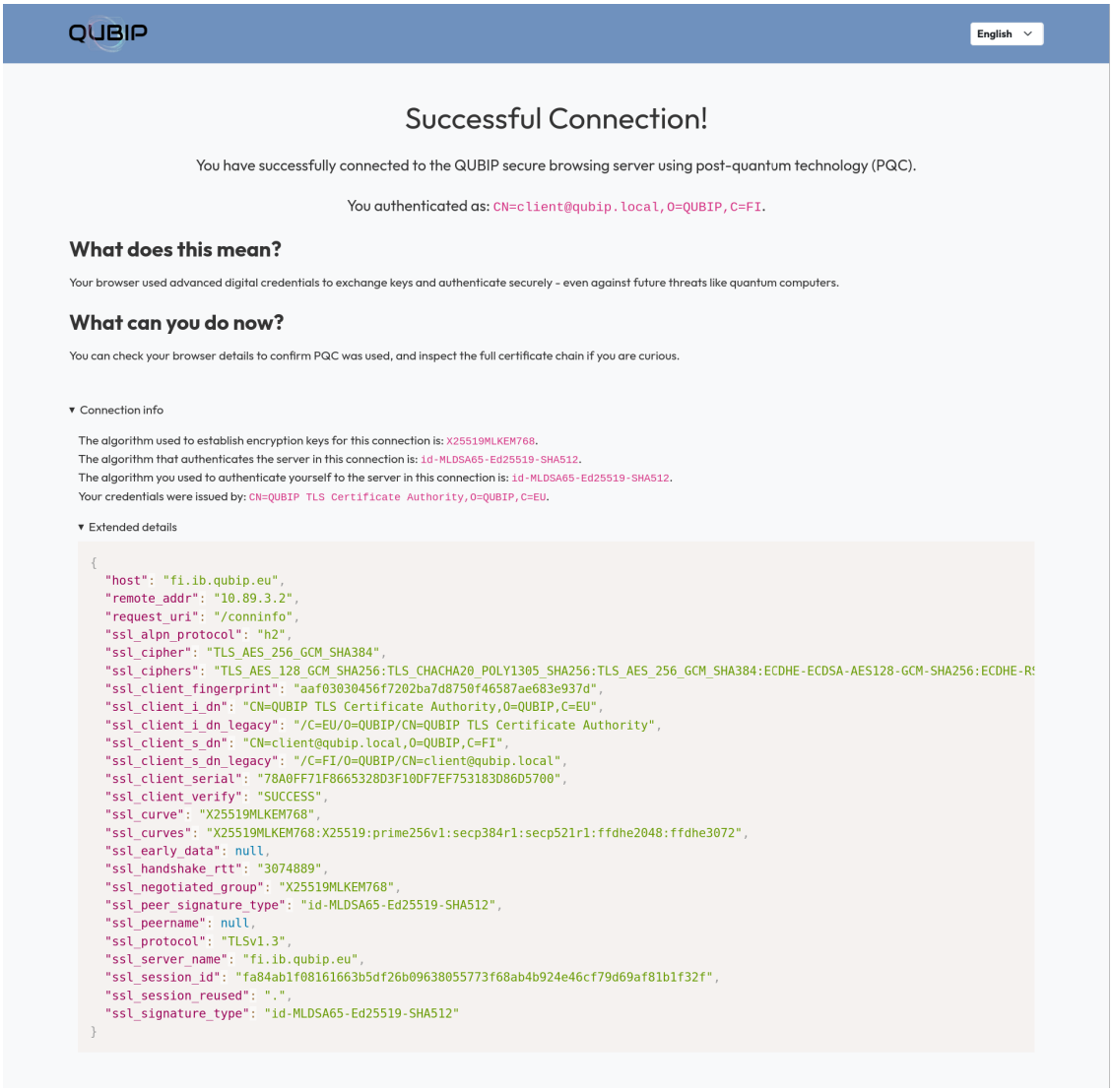

Figure 5.3. Successful mutual PQ/T hybrid TLS 1.3 connection using `x25519mlkem768` for key exchange and `MLDSA65-Ed25519` for authentication against the IB Pilot test server running on port 22002.

worth noting that the OQS test infrastructure currently supports only server-side authentication scenarios and does not provide configurations for mutual TLS authentication. Consequently, the evaluation focused exclusively on server-only authentication.

All conducted tests were successful, demonstrating the capability of QUBIP Firefox to establish TLS 1.3 connections using also pure post-quantum cryptographic implementations for authentication. Figure 5.5 illustrates an example of a successful TLS 1.3 connection established using `x25519mlkem768` for key exchange and `ML-DSA-65` for authentication. Although not shown in the figure, all other supported parameter sets yielded equally successful results. The OQS server certificate was rooted in an RSA-based public key infrastructure. However, the intermediate Certification Authority employed a pure `ML-DSA` signature scheme. This configuration also enabled the evaluation of QUBIP Firefox's ability to verify pure `ML-DSA` signatures in the server certificate chain, thereby successfully testing the interoperability of the `mozpkix` certificate validation component.

The final interoperability test was conducted using **Cloudflare Research's Post-Quantum**

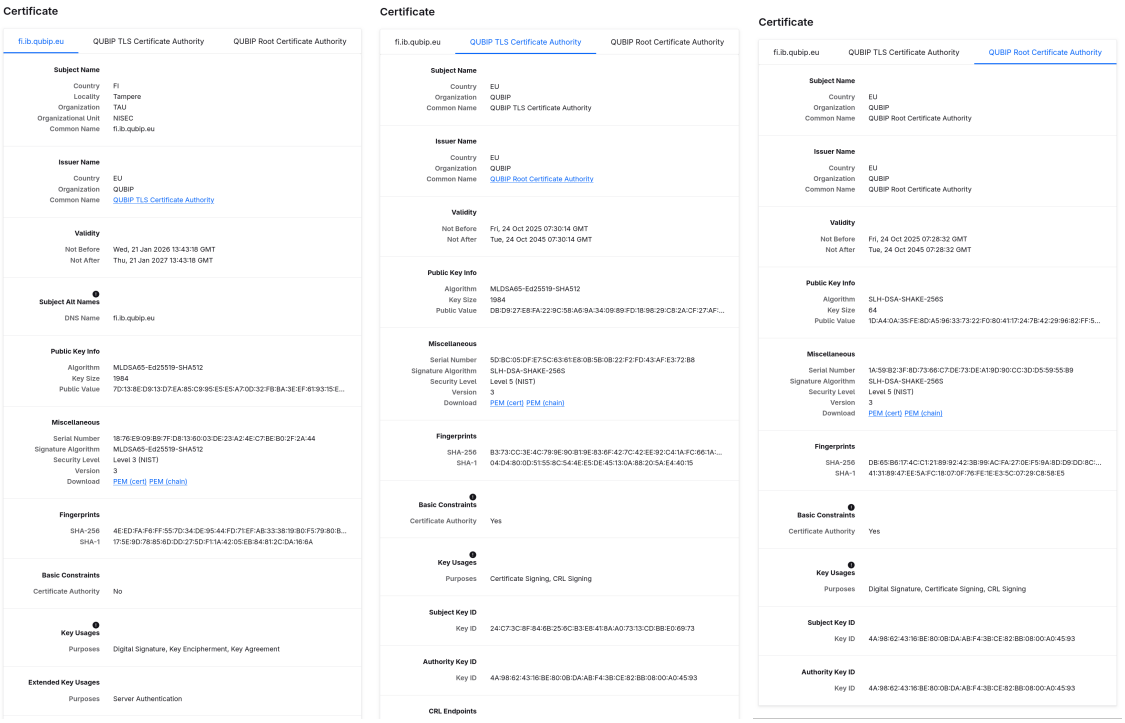

Figure 5.4. Complete certificate chain presented by an IB Pilot test server. From left to right: the end-entity (EE) certificate, signed by the intermediate CA using `MLDSA65-Ed25519`; the intermediate certificate, signed by the root CA using `SLH-DSA-SHAKE-256S`; and the root certificate, self-signed with `SLH-DSA-SHAKE-256S`.

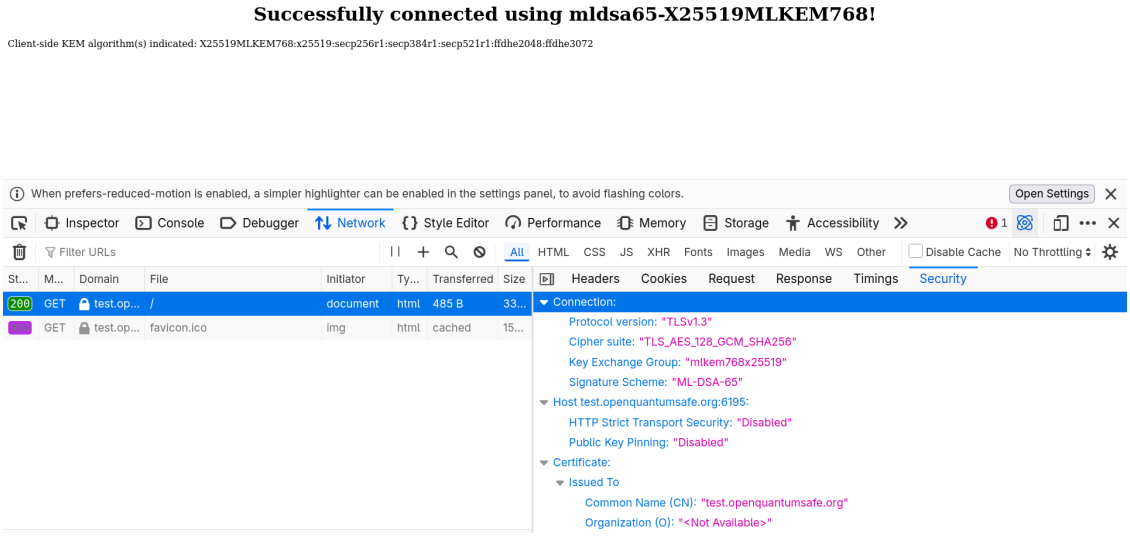

Figure 5.5. Successful PQ/T hybrid TLS 1.3 connection using `x25519mlkem768` for key exchange and `ML-DSA-65` for authentication against the OQS test server. Connection details are displayed through Firefox's Developer Tools.

Key Agreement test page. This server supports only post-quantum/hybrid key exchange, specifically the `x25519mlkem768` group, while all digital signature algorithm involved relied on traditional algorithms, with the server's certificate chain rooted in `ECDSA-P256-SHA256`. Although the key exchange functionality had been previously implemented by the previous contributor, the

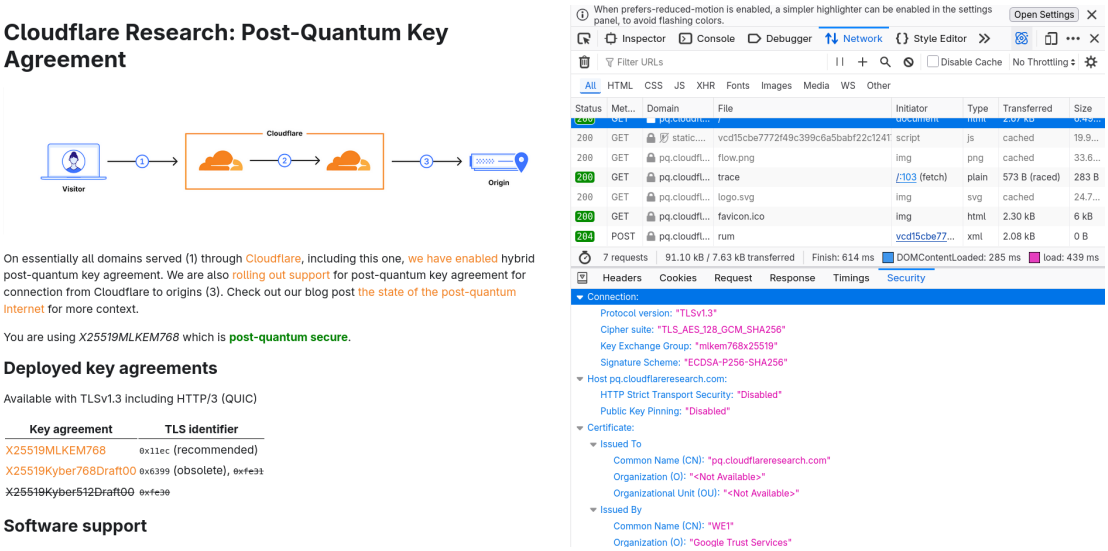

Figure 5.6. Successful PQ/T hybrid TLS 1.3 connection using `x25519mlkem768` for key exchange against the Cloudflare Research's test server. Connection details are displayed through Firefox's Developer Tools.

successful outcome of this test demonstrated that QUBIP Firefox is capable of correctly negotiating hybrid key exchange mechanisms and properly delegating the post-quantum component (`ML-KEM-768`) to the externally developed and loaded `qryptotoken` module. Figure 5.6 illustrates the successful PQ/T hybrid TLS 1.3 connection.

5.4 Challenges and Limitations

While the results presented in this thesis demonstrate a successful integration of post-quantum cryptography into Mozilla Firefox following the principle of cryptographic agility, a few challenges and limitations were encountered throughout the implementation process. Beyond the inherent challenges of interfacing with the NSS codebase, which suffers from inadequate documentation and the persistence of legacy code that complicates the introduction of new features, this section summarizes the main architectural criticalities encountered and discusses their implications for the final solutions.

Thread Stack Size Limitations Regardless of which `PKCS#11` module is loaded, Firefox performs cryptographic operations within dedicated worker threads. Firefox uses NSPR's thread API (`PR_CreateThread`[5]), which accepts an explicit `stackSize` parameter. Typically, this parameter is set to zero, allowing NSPR to select a machine- and OS-specific default stack size for newly created threads. Although the default thread stack size in most modern Linux systems is typically around 8 MiB, this limit can nevertheless be exceeded in practice when executing post-quantum cryptographic algorithms.

In particular, during certificate chain validation or signature verification in server authentication scenarios, the verification path reaches the `PKCS#11` token implementation. In the case of `qryptotoken`, these operations are delegated to external libraries such as `libcrux`, which provides the `ML-DSA` implementation. The `libcrux` library relies extensively on large stack-allocated data structures, including matrices and vectors of polynomial ring elements, as well as deeply nested call graphs involving sampling routines, SHAKE-based expanders, number-theoretic transforms

[5] https://firefox-source-docs.mozilla.org/nspr/reference/pr_createthread.html

(NTT), and SIMD/AVX-optimized operations. These components collectively require tens to hundreds of kilobytes of stack space within a single call chain, further amplified by compiler-generated register spill space and strict alignment requirements imposed by vectorized instructions.

When executed within Firefox's verification threads, whose default stack allocation may be insufficient for such computationally intensive workloads, the stack pointer can cross guard pages, resulting in stack exhaustion and a consequent segmentation fault (`SIGSEGV`) due to invalid memory access. This issue arose during the initial development phase, manifesting as unexpected segmentation faults during early attempts to verify `ML-DSA` signatures in TLS handshakes and certificate chains.

To mitigate this limitation, `qryptotoken`'s Adapters were modified to explicitly spawn dedicated threads with increased stack sizes prior to invoking `libcrux` operations. This approach ensured sufficient stack space for the heavy stack-based allocations required by the underlying library, preventing stack overflows and ensuring reliable execution during cryptographic operations.

Firefox's Constraints on Third-Party PKCS#11 Modules The adoption of `qryptotoken` into Firefox revealed significant architectural limitations inherent to the browser's handling of cryptographic material. Although Firefox provides considerable flexibility in loading external `PKCS#11` modules to delegate cryptographic operations, certain cases exist where it maintains exclusive control through its default internal module integrated within NSS, formally known as `softtoken`. As introduced in Section 3.2.3, one of the tokens exposed by this module is the **NSS Certificate DB**, which stores certificates and private keys in proprietary database files.

A critical limitation emerged with respect to `PKCS#12` material import operations. `PKCS#12`, as defined in RFC 7292 [82], is a standard transfer syntax for personal identity information that enables the secure exchange of private keys, X.509 certificates, and other cryptographic material between platforms under various privacy and integrity modes. When importing `PKCS#12` files containing certificate and private keys, Firefox exclusively targets the internal NSS Certificate DB token, providing no mechanism to redirect this operation to third-party `PKCS#11` modules. The `PKCS#12` import process obtains a reference to the internal NSS Certificate DB slot during initialization and maintains this reference throughout the entire operation. Consequently, all cryptographic material extracted from `PKCS#12` files is automatically stored in NSS's proprietary certificate and key databases, bypassing any third-party tokens that may be loaded in the browser.

This design constraint is further reinforced during the unwrapping of encrypted private keys encoded in `PKCS#8` format [83]. The unwrap operation employs a non-standard attribute specific to NSS that mandates the creation of persistent objects within the NSS database. This architectural decision effectively prevents the migration of existing cryptographic material from `PKCS#12` file formats to third-party `PKCS#11` tokens through Firefox's native interface.

Regarding export operations, Firefox theoretically supports extracting private keys and certificates to `PKCS#12` format through its backup functionality. However, this operation succeeds only if the private key object possesses the `CKA_EXTRACTABLE` attribute. The majority of hardware security tokens and smart cards deliberately prevent private key extraction in any form, aligning with the fundamental security principle that private keys should never leave the secure boundary of the cryptographic device. Notably, even if extraction were possible, the resulting `PKCS#12` file could only be reimported into Firefox's internal database, making the export operation circular and of limited utility for migration purposes.

In developing `qryptotoken`, the design philosophy deliberately aligned with that of hardware secure tokens, following the principle that private keys should never leave the token boundary once generated. Despite being a software-based implementation of a `PKCS#11` token, `qryptotoken` enforces the non-extractability of private keys. `PKCS#12` files are particularly important when dealing with client certificates for TLS client authentication and the absence of this functionality in third-party token workflows clearly imposes operational limitations that necessitated alternative solutions

This architectural decision required the development of alternative workflows for key and certificate lifecycle management. Moreover, in the context of client authentication in TLS, the

generation of Certificate Signing Requests (CSRs) for obtaining a client certificate from a Certification Authority, as well as the import of the issued certificate, are not functionalities natively provided by upstream Mozilla Firefox. Following the operational model of commercial hardware tokens, key generation, CSR creation, and certificate import operations were implemented through dedicated external tools rather than through Firefox's built-in interfaces.

The modifications introduced in QUBIP Firefox's NSS theoretically enable the use of NSS's native tools to interact with `qryptotoken` for these operations. However, NSS contains hard-coded assumptions for interacting exclusively with its internal token, following different database conventions that complicate integration with general third-party tokens. Moreover, building QUBIP Firefox's NSS as a separate standalone library to leverage its tools for the management of `qryptotoken` proved impractical, despite being tested and functioning to some extent. For these reasons, customized versions of the `p11tool` and `certtool` utilities from the **GnuTLS** library [84] were developed to support post-quantum and hybrid signature algorithms. These tools were actively used to store client certificates in `qryptotoken` and to evaluate their usage in TLS client authentication scenarios, as described in the previous section for the IB Pilot use case 2 (UC02). Overall, this approach maintains complete separation between key lifecycle operations and Firefox's internal certificate management, while ensuring that all operations remain within the open-source ecosystem.

Given the strict limitations imposed by Firefox on the import of `PKCS#12` files to third-party tokens, and the limitations posed by non-extractability of private keys imposed by `qryptotoken`, an alternative migration strategy leveraged the inherent security architecture of the `qryptotoken` database itself. As discussed in Section 3.2.1, `qryptotoken` is built upon Kryoptic, which implements a single-file database design with encryption of sensitive material.

This architecture enables database-level migration: the entire token state, including encrypted private keys, certificates, and all cryptographic material, can be transferred between systems by simply copying the database file. Unlike `PKCS#12` export, which Firefox restricts to its internal token, database migration operates entirely outside Firefox's control. Upon loading the migrated database file in a new environment, users authenticate with the same PIN credentials to access the encrypted material. This approach effectively circumvents Firefox's `PKCS#12` limitations while maintaining the security guarantees established by `qryptotoken`'s layered encryption architecture.

Chapter 6

Conclusion

The transition to post-quantum cryptography (PQC) is not merely a replacement of public key algorithms, but a complex, ecosystem-wide migration that demands careful planning, architectural flexibility, and sustained interoperability. This thesis, conducted within the European QUBIP Project, addressed the critical challenge of transitioning Mozilla Firefox and its underlying Network Security Services (NSS) cryptographic library toward post-quantum cryptography through a modular, crypto-agile framework. Building upon prior work conducted within the QUBIP consortium, this research extended support for post-quantum and hybrid digital signatures, enabling fully quantum-resistant TLS 1.3 connections. By implementing a crypto-agile `PKCS#11` software-based token for integrating post-quantum algorithms, this work contributed to the broader objective of preparing the web browsing ecosystem for a quantum-secure future.

6.1 Answering the Research Questions

This section revisits the research questions formulated in Section 1.3 and provides conclusive answers based on the findings and implementations detailed in the preceding chapters.

RQ1: What is the methodology required to effectively guide the transition of Mozilla Firefox and its cryptographic library, NSS, toward post-quantum cryptography to enable a quantum-secure Internet browsing?

The transition of a complex, widely deployed system like Mozilla Firefox to post-quantum cryptography (PQC) requires a methodology that prioritizes cryptographic agility, backward compatibility, and incremental adoption. Based on the implementation work and results presented in this thesis, an effective methodology must address several interdependent concerns: maintaining operational continuity during the transition, preserving interoperability with existing infrastructure, and establishing flexible mechanisms for algorithm replacement or evolution as the post-quantum landscape matures.

The core principle underlying the methodology is the exploitation of existing modular architectural patterns rather than the introduction of disruptive core modifications. NSS's adherence to the `PKCS#11` standard provides a built-in extension point for introducing post-quantum capabilities. NSS relies on `softoken` as its internal token, which ultimately wraps cryptographic primitives implemented in the FreeBL library. Rather than directly integrating post-quantum algorithms into FreeBL, the methodology advocates the use of external cryptographic modules that expose post-quantum capabilities through the standardized Cryptoki interface. This approach maintains clear separation between Firefox's NSS and algorithm implementations, enabling each component to evolve independently.

A critical aspect of this modular strategy is the adoption of a *shallow* loadable module. The external module should function as a lightweight delegation layer rather than embedding cryptographic implementations directly within its codebase. By routing operations to independently

maintained, conditionally pluggable post-quantum libraries through well-defined abstract interfaces, the module achieves practical cryptographic agility. This design ensures that cryptographic algorithm implementations remain decoupled from both Firefox's NSS and the module itself, facilitating straightforward replacement as standards evolve or implementation flaws are discovered. This architectural separation proved essential during this research, particularly as the IETF `composite ML-DSA` specification evolved across multiple Internet Draft versions, as discussed in Chapter 5. The Adapter-based architecture within `qryptotoken` enabled concurrent support for different specification versions through conditional compilation, exemplifying how this design pattern can effectively manage evolving standardization processes while enabling evaluation in live real-world deployment scenarios.

However, module development alone is insufficient. NSS itself must be extended to recognize and correctly process post-quantum algorithms. This requires systematic extension of multiple internal components, each serving a distinct architectural role. The object identifier management system must be augmented to associate post-quantum algorithm identifiers with their corresponding `PKCS#11` mechanisms and policy flags. The key type system must be extended to represent post-quantum key structures, including their specific characteristics such as parameter set definitions for algorithms like `ML-DSA` and `SLH-DSA`. In addition, the `PKCS#11` abstraction layer must be enhanced to discover, query, and invoke post-quantum mechanisms exposed by external modules.

Critically, the high-level cryptographic interfaces require adaptation to accommodate fundamental operational differences between post-quantum and traditional algorithms. Post-quantum signature schemes such as `ML-DSA` and `SLH-DSA` perform message hashing internally rather than following the conventional hash-and-sign paradigm, necessitating modifications to signing and verification interfaces to support algorithms that do not require external digest computation. These changes must propagate from low-level mechanism invocation to the high-level APIs exposed to protocol implementations.

Protocol-level integration introduces additional challenges that extend beyond cryptographic primitive support. The TLS implementation must be enhanced to negotiate post-quantum algorithms during the handshake, requiring updates to signature scheme enumeration, preference ordering, and algorithm selection logic. More significantly, the certificate validation infrastructure requires careful extension to verify post-quantum signatures within X.509 certificate chains. This involves not only algorithm recognition but also creation of proper new verification APIs to correctly route verification operations through the `PKCS#11` interface to external modules.

An essential methodological consideration is the adoption of conservative deployment strategies during the transitional period. The methodology should distinguish between pure post-quantum algorithms and hybrid constructions that combine classical and post-quantum primitives. Hybrid approaches provide defense-in-depth by remaining secure unless both components are compromised, which is particularly valuable given the relative immaturity of post-quantum implementations compared to the extensive cryptanalysis applied to classical schemes.

The methodology therefore advocates for policy-driven deployment that enables hybrid constructions by default while treating pure post-quantum algorithms as experimental capabilities requiring explicit user opt-in. This approach aligns with general EU recommendations, as well as other European security agencies such as France's ANSSI and Germany's BSI, which advocate hybrid implementations as a precautionary measure during the migration phase. Policy enforcement must operate at multiple architectural layers: at the cryptographic library level through algorithm policy frameworks that control permitted usage contexts, and within the application through configuration mechanisms that allow users to manage trade-offs between conservative security postures and experimental algorithm adoption. The layered policy architecture implemented across NSS and Firefox, described in Section 5.1.6, reflects this approach.

Validation methodology is equally critical. The transition strategy must incorporate comprehensive testing that extends beyond unit-level verification to encompass integration testing in realistic deployment scenarios and interoperability testing against external implementations. Integration testing should evaluate complete protocol flows, including both server-only and mutual authentication scenarios, within controlled environments that replicate production PKI structures. Interoperability testing against independently developed implementations provides essential validation that the system correctly adheres to evolving standards and can establish secure

connections across the heterogeneous ecosystem that will characterize the transition period. The results presented in Section 5.3, demonstrating successful connections against both the QUBIP Internet Browsing (IB) Pilot and external test servers including Open Quantum Safe servers and Cloudflare Research's Post-Quantum Key Agreement server, provide empirical evidence of this methodology's effectiveness.

The methodology should also address practical deployment challenges that extend beyond support of new algorithms. Certificate and key management workflows, particularly for client authentication, require careful handling when external cryptographic modules are used. The architectural limitation identified in Section 5.4, where Firefox's `PKCS#12` import functionality is restricted to the internal NSS certificate database, highlights the need for complementary tooling and alternative workflows to support full lifecycle management of cryptographic material stored in external tokens.

In summary, the methodology developed in this research demonstrates that transitioning Firefox and NSS toward post-quantum cryptography requires a holistic approach spanning architectural extension, modular algorithm integration, protocol-level adaptation, policy-driven deployment, and comprehensive validation. By leveraging NSS's existing modular design principles and adherence to the `PKCS#11` standard, this methodology enables incremental adoption of post-quantum algorithms while maintaining operational continuity, backward compatibility, and the cryptographic agility necessary to adapt to an evolving post-quantum landscape.

RQ2: How can the proposed shallow-loadable module approach enhance cryptographic agility and facilitate the integration of post-quantum algorithms within NSS?

The shallow-loadable module approach represents a conceptual shift from traditional cryptographic library design, in which algorithm implementations are tightly integrated within the library itself. NSS already provides a limited form of cryptographic agility by supporting the loading of external `PKCS#11` modules to delegate cryptographic services. While `PKCS#11` modules inherently offload cryptographic services from relying applications, conventional tokens still embed implementations directly within their codebase or maintain tight dependencies on specific cryptographic libraries. Consequently, although NSS can switch between different `PKCS#11` modules at runtime, the selected token ultimately remains bound to a monolithic cryptographic backend, reproducing the same architectural constraint at a different layer. This model therefore offers only minimal crypto-agility, as adapting cryptographic implementations in a truly flexible manner remains impractical, particularly in the context of the evolving, heterogeneous, and still uncertain post-quantum cryptographic landscape.

Recalling from Chapter 4, the approach developed in this thesis achieves cryptographic agility through deliberate decoupling of three distinct concerns: the standardized Cryptoki interface, the Adapter-based architecture that bridges this interface to external implementations, and the cryptographic primitives themselves. The term "shallow" captures this essential characteristic: the module maintains minimal internal complexity functioning as a lightweight translation layer that maps `PKCS#11` operations to corresponding functions in external third-party cryptographic libraries.

The shallow design enhances cryptographic agility simultaneously within the module and within NSS. Within the module, algorithm implementations can be replaced by updating external libraries or switching Adapters without disrupting the `PKCS#11` interface. Within NSS, adding new post-quantum algorithms requires only that the module expose appropriate `PKCS#11` mechanisms and that NSS recognize the corresponding algorithm identifiers, extending the `PKCS#11` abstraction layer and high-level cryptographic interfaces as needed. In this thesis, such extensions were primarily confined to components directly used by Firefox, such as the TLS implementation and the Mozilla PKIX library. However, in general, supporting new algorithms requires the systematic addition of recurring scaffolding code across these layers.

From NSS's perspective, this approach dramatically reduces the complexity of supporting diverse post-quantum implementations, particularly when evaluating algorithms for which standardization processes are still ongoing. Rather than maintaining algorithm implementations directly, which would require tracking developments across multiple algorithm families, managing

dependencies on specialized mathematical libraries, and validating cryptographic correctness, NSS can delegate these concerns to the broader cryptographic library ecosystem. Consequently, the library's role remains focused on protocols integration, policy enforcement, and correct exploitation of cryptographic services through the `PKCS#11` interface.

RQ3: What are the critical challenges in adapting PKI components to support post-quantum algorithms, ensuring interoperability within Firefox and NSS?

The integration of post-quantum cryptography into Public Key Infrastructure components within Mozilla Firefox revealed challenges that extend beyond algorithm support to encompass certificate validation logic, trust establishment mechanisms, and policy enforcement frameworks.

Certificate chain validation, implemented through NSS's `mozpkix` library, required careful extension to support post-quantum signature verification. As detailed in Section 5.1.5, the validation engine's algorithm identification system needed enhancement to recognize certificates containing DER-encoded post-quantum public keys and digital signatures, and correctly route verification operations through new appropriate APIs tailored for post-quantum signature algorithms. A significant semantic difference emerged as the targeted post-quantum signature schemes perform message hashing internally rather than following the conventional hash-and-sign paradigm. This necessitated the introduction of a "no digest" concept, enabling the validation engine to bypass external hash computation and delegate the complete verification operation to the `PKCS#11` token.

Trust anchor management presents particularly complex challenges during the post-quantum transition. Root Certification Authority certificates typically possess validity periods exceeding tens or twenty years, creating substantial inertia in PKI trust hierarchies. The migration to post-quantum trust anchors cannot occur instantaneously. Instead, a phased transition must accommodate clients with varying levels of post-quantum support. As discussed in Section 2.12.3, this necessitates parallel trust hierarchies or cross-signing arrangements where post-quantum intermediate CAs are signed by both classical and post-quantum roots. Firefox's certificate validation infrastructure, through its integration with the CKBI (Built-in Root Module), successfully processed certificate chains rooted in post-quantum CAs during testing, as demonstrated by the QUBIP PKI infrastructure that employed `SLH-DSA-SHAKE-256S` for the QUBIP root CA.

However, this research did not exhaustively evaluate scenarios involving fully parallel trust hierarchies anchored simultaneously in classical and post-quantum roots. Backward compatibility is nonetheless preserved, as classical trust hierarchies continue to function as expected, which reflects Firefox's standard behaviour against single classically rooted hierarchies. While the concurrent processing of multiple certificate chains and the enforcement of distinct security policies is technically feasible within NSS, achieving correct and predictable behaviour requires well-defined policy controls, particularly at the Firefox level. Determining the "desired level of security" in these heterogeneous environments remains a critical challenge, as the browser must discern when and how to prioritize post-quantum validation over classical fallbacks without compromising the user's security posture. This work primarily assumed post-quantum signatures at the trust anchor level. Nevertheless, NSS's existing architecture readily accommodates cross-signed certificate chains anchored in classical root CAs, as demonstrated by interoperability testing with the Open Quantum Safe (OQS) test server, where post-quantum certificate chains remained rooted in classical algorithms. Composite certificate scenarios were evaluated at the intermediate CA level, but the same mechanisms are applicable to root CA certificates, indicating that NSS can support mixed trust models during the transitional period.

The revocation and transparency infrastructure components present additional considerations that were not exhaustively evaluated in this work. Certificate Transparency (CT) proofs, OCSP responses, and CRLs must all eventually support post-quantum signatures as these primitives become embedded throughout the PKI hierarchy. While this thesis did not explicitly test CT proofs signed with post-quantum algorithms, nor OCSP responses or CRLs employing post-quantum signatures, extensions to the `TrustDomain` interface should effectively support these scenarios. As described in Section 5.1.5, the Firefox's `NSSCertDBTrustDomain` implementation was enhanced to incorporate post-quantum signature verification paths that route operations through the newly added verification APIs in `mozpkix`. This component serves as the central validation mechanism

through which OCSP responses, CRL signatures, and Certificate Transparency verification are processed. Consequently, the infrastructure modifications implemented to support post-quantum certificate chain validation should naturally extend to these auxiliary PKI components, though empirical validation of these specific scenarios remains a direction for future work.

Certificate size implications pose practical challenges for protocol implementations and infrastructure components. `ML-DSA` signatures, while substantially smaller than their `SLH-DSA` counterparts, nevertheless exceed traditional `ECDSA` or `RSA` signatures by significant margins. When multiple signatures appear in a single certificate chain, including the end-entity certificate signature, intermediate CA signatures, and potentially embedded OCSP responses or Certificate Transparency SCTs, the cumulative size can approach or exceed protocol message limits. The TLS 1.3 implementation in NSS required careful testing to ensure that handshake message buffers could accommodate post-quantum certificate chains without truncation or failure. However, significant challenges emerged at the `PKCS#11` integration layer. Specifically, as detailed in Section 5.4, the primary obstacle was not protocol message truncation, but rather thread stack size limitations within the browser's execution environment. Post-quantum algorithms like `ML-DSA`, when executed through external libraries such as libcrux within `qryptotoken`, utilize extensive stack-allocated structures for polynomial operations and matrices. These requirements often exceeded Firefox's default worker thread stack allocations, leading to stack exhaustion and segmentation faults during certificate validation and signature verification.

Despite these challenges, the results presented in Chapter 5 show that PKI components can be adapted to support post-quantum algorithms while preserving interoperability with existing infrastructure. The successful validation of certificate chains containing pure `ML-DSA`, `SLH-DSA`, and composite `MLDSA65-Ed25519` signatures, together with the establishment of end-to-end TLS 1.3 connections using these credentials, provides empirical evidence that the technical barriers to post-quantum PKI deployment are manageable. The remaining challenges are primarily operational, and require coordinated efforts across the broader ecosystem, including the definition of standardized migration strategies, the update of trust anchors, and the optimization of infrastructure components to account for the increased certificate sizes introduced by post-quantum cryptography.

6.2 Contributions and Future Directions

This thesis primary contributions include the development of `qryptotoken` as a shallow-loadable `PKCS#11` module implementing an Adapter-based framework for cryptographic operation delegation, extensive modifications to Firefox and NSS enabling recognition and processing of post-quantum algorithms, and the successful demonstration of end-to-end PQ/T hybrid TLS 1.3 connections in realistic deployment scenarios. The work conducted contributes directly to the QUBIP project's strategic objectives concerning cryptographic libraries transition through loadable modules and the advancement of Firefox toward post-quantum security. More broadly, the research validated the shallow module design pattern as an effective strategy for achieving cryptographic agility, demonstrating applicability beyond the specific context of Firefox and NSS to the wider domain of cryptographic `PKCS#11` tokens. Moreover, contributions to upstream NSS related to `ML-DSA` support highlight the broader applicability of this work beyond the QUBIP ecosystem. Nevertheless, several directions for future work emerge from this thesis and are presented below.

Performance Evaluation Prior work [12] defined comprehensive performance evaluation criteria that remain in part unaddressed, specifically on systematic measurements of TLS handshake latency, traffic overhead, and connection establishment success rates in post-quantum and hybrid configurations. Introducing browser-level instruments to capture metrics such as Time To First Byte (TTFB) and First Contentful Paint (FCP) would enable quantitative assessment of user-perceived performance impacts, and would be essential for guiding development strategies and optimization opportunities.

Extended Composite Algorithms Support Current support of `composite ML-DSA` in QUBIP Firefox is limited to `MLDSA65-Ed25519`, reflecting the experimental state of composite algorithm standardization during the development of this work. Future efforts should extend support to additional composite variants. Achieving this requires an architectural restructuring within NSS to introduce a generalized composite algorithm framework, rather than hardcoded support for individual variants. Such changes would need to be coordinated with NSS maintainers and aligned with evolving `PKCS#11` standard specifications, enabling modular representations similar to the parameter set differentiation used for `ML-DSA` and `SLH-DSA`.

PKI Components Validation Although architectural support for post-quantum signatures in revocation mechanisms (i.e., OCSP and CRLs) and Certificate Transparency was implemented, empirical validation of these components remains incomplete. Future work should conduct systematic testing of OCSP responses, CRLs, and CT proofs signed with post-quantum algorithms to verify correct behavior under realistic PKI deployment conditions.

Enhanced Cryptographic Agility and Testing Infrastructure The Adapter-based architecture of `qryptotoken` enables systematic evaluation of multiple cryptographic backends. Future work should extend this framework by developing Adapters for additional third-party post-quantum libraries, thereby supporting more comprehensive comparative analysis of alternative implementations in terms of performance, security properties, and maturity. At present, the Wycheproof-based testing framework validates only `ML-DSA`, due to the limited availability of test vectors for other schemes. As test vectors become available for `SLH-DSA` and composite algorithms, the testing infrastructure should be enhanced to provide broader coverage. Finally, the current Adapter selection mechanism is based on compile-time feature flags, which requires separate builds to evaluate different cryptographic backends. Extending `qryptotoken` to support runtime Adapter selection would significantly improve operational flexibility and enable more practical deployment and experimentation scenarios.

Taken together, these directions address both immediate practical challenges and longer-term architectural concerns, positioning the work presented in this thesis as a foundation for continued progress toward robust post-quantum readiness within Mozilla Firefox.

Bibliography

[1] D. J. Bernstein, J. Buchmann, and E. Dahmen, "Post-Quantum Cryptography", Springer-Verlag, 2009, ISBN: 978-3-540-88701-0

[2] P. Shor, "Algorithms for quantum computation: discrete logarithms and factoring", Proceedings 35th Annual Symposium on Foundations of Computer Science, 1994, pp. 124–134, DOI 10.1109/SFCS.1994.365700

[3] L. K. Grover, "A fast quantum mechanical algorithm for database search", Twenty-Eighth Annual ACM Symposium on Theory of Computing, New York (USA), 1996, p. 212–219, DOI 10.1145/237814.237866

[4] National Institute of Standards and Technology, "NIST Releases First 3 Finalized Post-Quantum Encryption Standards", https://www.nist.gov/news-events/news/2024/08/nist-releases-first-3-finalized-post-quantum-encryption-standards, August 2024, Accessed: 2025-07-30

[5] National Institute of Standards and Technology, "Module-Lattice-Based Key-Encapsulation Mechanism Standard", FIPS 203, August 2024, DOI 10.6028/NIST.FIPS.203

[6] National Institute of Standards and Technology, "Module-Lattice-Based Digital Signature Standard", FIPS 204, August 2024, DOI 10.6028/NIST.FIPS.204

[7] National Institute of Standards and Technology, "Stateless Hash-Based Digital Signature Standard", FIPS 205, August 2024, DOI 10.6028/NIST.FIPS.205

[8] U.S. National Security Agency, "Commercial National Security Algorithm Suite 2.0", tech. rep., National Security Agency (NSA), September 2022. https://media.defense.gov/2025/May/30/2003728741/-1/-1/0/CSA_CNSA_2.0_ALGORITHMS.PDF

[9] National Institute of Standards and Technology, "NIST to Withdraw SP 800-67 Rev. 2", https://csrc.nist.gov/news/2023/nist-to-withdraw-sp-800-67-rev-2, June 2023, Accessed: 2025-07-30

[10] L. Chen, "Considerations for Achieving Cryptographic Agility: Strategies and Practices", NIST CSWP 39 2pd, 2025, DOI 10.6028/NIST.CSWP.39.2pd

[11] IETF Limited Additional Mechanisms for PKIX and S/MIME (LAMPS). https://datatracker.ietf.org/wg/lamps/about/, Accessed: 2025-10-19

[12] D. Luoma, "Enhancing Network Security: Post-Quantum Cryptography through loadable modules in Firefox and NSS: Integrating Future-Proof Cryptographic Techniques", Master's thesis, Tampere University, November 2024. https://trepo.tuni.fi/bitstream/handle/10024/161998/LuomaDaniel.pdf

[13] QUBIP Consortium, "QUBIP: Quantum-oriented Update to Browsers and Infrastructure for the PQ Transition", https://qubip.eu/, 2023-2026

[14] A. J. Menezes, P. C. van Oorschot, and S. A. Vanstone, "Handbook of Applied Cryptography", CRC Press, 1996, ISBN: 0-8493-8523-7

[15] B. Schneier, "Applied Cryptography: Protocols, Algorithms, and Source Code in C", John Wiley & Sons, 2nd ed., 1996, ISBN: 978-0-471-11709-4

[16] National Institute of Standards and Technology, "SHA-3 Standard: Permutation-Based Hash and Extendable-Output Functions", FIPS 202, August 2015, DOI 10.6028/NIST.FIPS.202

[17] "ISO/IEC 19772:2009 Information technology – Security techniques – Authenticated encryption", February 2009, https://www.iso.org/standard/46345.html

[18] M. Dworkin, "Recommendation for Block Cipher Modes of Operation: Galois/Counter Mode (GCM) and GMAC", NIST Special Publication 800-38D, November 2007, DOI 10.6028/NIST.SP.800-38D

[19] A. Langley and N. Modadugu, "ChaCha20 and Poly1305 for IETF Protocols", RFC 8439, June 2018, DOI 10.17487/RFC8439

[20] J. A. Salowey, D. McGrew, and A. Choudhury, "AES Galois Counter Mode (GCM) Cipher Suites for TLS", RFC 5288, August 2008, DOI 10.17487/RFC5288

[21] C. Paar and J. Pelzl, "Understanding Cryptography: A Textbook for Students and Practitioners", Springer Verlag Berlin Heidelberg, 1 ed., November 2010, ISBN: 978-3-642-04100-6

[22] D. Hankerson, A. J. Menezes, and S. Vanstone, "Guide to Elliptic Curve Cryptography", Springer Verlag, 2004, ISBN: 0-387-95273-X

[23] National Institute of Standards and Technology, "Recommendation for Key Management, Part 1: General (Revision 5)", NIST Special Publication 800-57, Part 1, Rev. 5, May 2020, DOI 10.6028/NIST.SP.800-57pt1r5

[24] National Institute of Standards and Technology, "Digital Signature Standard (DSS)", August 2023, DOI 10.6028/NIST.FIPS.186-5

[25] D. Harkins, T. Polk, J. Hodges, and S. Farrell, "Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile", RFC 5280, May 2008, DOI 10.17487/RFC5280

[26] M. Nystrom and B. Kaliski, "PKCS #10: Certification Request Syntax Specification Version 1.7", RFC 2986, November 2000, DOI 10.17487/RFC2986

[27] B. Laurie, A. Langley, E. Kasper, A. Cutter, R. Stradling, and M. S. Melara, "Certificate Transparency Version 2.0", RFC 9162, December 2021, DOI 10.17487/RFC9162

[28] OASIS Open, "OASIS Enhances Popular Public-Key Cryptography Standard, PKCS#11, for Mobile and Cloud", March 2013, https://www.oasis-open.org/news/pr/oasis-enhances-popular-public-key-cryptography-standard-pkcs-11-for-mobile\-and-cloud/ Accessed: 2025-07-28

[29] OASIS PKCS11 Technical Committee, "PKCS#11 Specification Version 3.2", Committee Specification Draft 01, OASIS, April 2025. Latest stage: https://docs.oasis-open.org/pkcs11/pkcs11-spec/v3.2/pkcs11-spec-v3.2.html

[30] OASIS PKCS11 Technical Committee, "PKCS#11 Cryptographic Token Interface Usage Guide Version 3.2", April 2025, Latest stage: https://docs.oasis-open.org/pkcs11/pkcs11-ug/v3.2/pkcs11-ug-v3.2.html

[31] T. Dierks and C. Allen, "The TLS Protocol Version 1.0", RFC 2246, January 1999, DOI 10.17487/RFC2246

[32] T. Dierks and E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.1", RFC 4346, January 2006, DOI 10.17487/RFC4346

[33] T. Dierks and E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.2", RFC 5246, August 2008, DOI 10.17487/RFC5246

[34] E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.3", RFC 8446, August 2018, DOI 10.17487/RFC8446

[35] B. Dowling, M. Fischlin, F. Günther, and D. Stebila, "A Cryptographic Analysis of the TLS 1.3 Handshake Protocol", International Association for Cryptologic Research (IACR), 2020, DOI 10.1007/s00145-021-09384-1

[36] H. Krawczyk and P. Eronen, "HMAC-based Extract-and-Expand Key Derivation Function (HKDF)", RFC 5869, May 2010, DOI 10.17487/RFC5869

[37] J. Preskill, "Quantum Computing in the NISQ era and beyond", Quantum, vol. 2, August 2018, p. 79, DOI 10.22331/q-2018-08-06-79

[38] M. AbuGhanem, "IBM quantum computers: evolution, performance, and future directions", The Journal of Supercomputing, vol. 81, April 2025, p. 687, DOI 10.1007/s11227-025-07047-7

[39] R. Acharya and Google Quantum AI Collaboration, "Quantum error correction below the surface code threshold", Nature, vol. 638, February 2025, pp. 920–926, DOI 10.1038/s41586-024-08449-y

[40] M. Ajtai, "Generating hard instances of lattice problems (extended abstract)", Twenty-Eighth Annual ACM Symposium on Theory of Computing, New York (USA), July 1996, p. 99–108, DOI 10.1145/237814.237838

[41] M. Ajtai and C. Dwork, "A public-key cryptosystem with worst-case/average-case equivalence", Twenty-Ninth Annual ACM Symposium on Theory of Computing, New York (USA), May 1997, pp. 284–293, DOI 10.1145/258533.258604

[42] J. Hoffstein, J. Pipher, and J. H. Silverman, "NTRU: A Ring-Based Public Key Cryptosystem", Third International Symposium on Algorithmic Number Theory, Berlin (Germany), January 1998, pp. 267–288

[43] O. Regev, "On lattices, learning with errors, random linear codes, and cryptography", J. ACM, vol. 56, September 2009, DOI 10.1145/1568318.1568324

[44] R. J. McEliece, "A Public-Key Cryptosystem Based on Algebraic Coding Theory", tech. rep., Jet Propulsion Laboratory, California Institute of Technology, Pasadena, CA, USA, April 1978. https://tda.jpl.nasa.gov/progress_report/42-44/44N.PDF

[45] National Institute of Standards and Technology, "NIST Selects HQC as Fifth Algorithm for Post-Quantum Encryption", https://www.nist.gov/news-events/news/2025/03/nist-selects-hqc-fifth-algorithm-post-quantum-encryption, March 2025

[46] R. C. Merkle, "Secrecy, Authentication, and Public Key Systems", IEEE Symposium on Security and Privacy, Oakland, CA, 1979, pp. 157–167

[47] D. McGrew, M. Curcio, and S. Fluhrer, "Leighton-Micali Hash-Based Signatures", RFC 8554, April 2019, DOI 10.17487/RFC8554

[48] A. Huelsing, D. Butin, S.-L. Gazdag, J. Rijneveld, and A. Mohaisen, "XMSS: eXtended Merkle Signature Scheme", RFC 8391, May 2018, DOI 10.17487/RFC8391

[49] D. Cooper, Q. Dang, S. Fluhrer, S. Gazdag, J. Kelsey, and D. Miller, "Recommendation for Stateful Hash-Based Signature Schemes", October 2020, DOI 10.6028/NIST.SP.800-208

[50] A. Kipnis, J. Patarin, and L. Goubin, "Unbalanced Oil and Vinegar Signature Schemes", Advances in Cryptology – EUROCRYPT '99, January 1999, pp. 206–222

[51] G. Alagic *et al.*, "Status Report on the Third Round of the NIST Post-Quantum Cryptography Standardization Process", July 2022, DOI 10.6028/NIST.IR.8413

[52] W. Castryck and T. Decru, "An Efficient Key Recovery Attack on SIDH", Advances in Cryptology – EUROCRYPT 2023 (C. Hazay and M. Stam, eds.), Cham, April 2023, pp. 423–447, DOI 10.1007/978-3-031-30589-4_15

[53] M. Mosca, "Cybersecurity in an Era with Quantum Computers: Will We Be Ready?", IEEE Security & Privacy, vol. 16, October 2018, pp. 38–41, DOI 10.1109/MSP.2018.3761723

[54] European Telecommunications Standards Institute, "CYBER; Guidance on Migration to Quantum-Safe Cryptography", Tech. Rep. TR 103 619 V1.1.1, ETSI, July 2020

[55] M. Ounsworth, J. Gray, M. Pala, and J. Klaußner, "Composite Signatures For Use In Internet PKI", I-D draft-ietf-lamps-pq-composite-sigs-14, Internet Engineering Task Force (IETF), December 2025. https://datatracker.ietf.org/doc/html/draft-ietf-lamps-pq-composite-sigs-14

[56] C. Hickman, "From Kuala Lumpur to Crypto-Agility: Reflections from the PQC Conference 2025", https://www.keyfactor.com/blog/from-kuala-lumpur-to-crypto-agility-reflections-from-the-pqc-conference-2025/, Accessed: January 1, 2026

[57] The Chromium Project, "Post-Quantum PKI Design", https://www.chromium.org/Home/chromium-security/post-quantum-pki-design/, Accessed: January 12, 2026

[58] D. Benjamin, D. O'Brien, and B. Westerbaan, "Merkle Tree Certificates for TLS", I-D draft-davidben-tls-merkle-tree-certs, Internet Engineering Task Force (IETF), December 2025. https://datatracker.ietf.org/doc/draft-davidben-tls-merkle-tree-certs/09/

[59] Agence Nationale de la Sécurité des Systèmes d'Information, "ANSSI views on the post-quantum cryptography transition", tech. rep., ANSSI, January 2022. https://messervices.cyber.gouv.fr/documents-guides/follow_up_position_paper_on_post_quantum_cryptography.pdf

[60] Federal Office for Information Security (BSI), "Migration to Post-Quantum Cryptography", technical guideline, Bundesamt für Sicherheit in der Informationstechnik, May 2021. https://www.bsi.bund.de/SharedDocs/Downloads/EN/BSI/Crypto/Migration_to_Post_Quantum_Cryptography.pdf

[61] T. Reddy.K, D. Wing, and Y. Rosomakho, "Guidance for Migration to Composite, Dual, or PQC Authentication", I-D draft-reddy-pquip-pqc-signature-migration, Internet Engineering Task Force (IETF), October 2025. https://datatracker.ietf.org/doc/draft-reddy-pquip-pqc-signature-migration/01/

[62] B. Beck, D. Benjamin, D. O'Brien, and K. Nekritz, "TLS Trust Anchor Identifiers", I-D draft-ietf-tls-trust-anchor-ids, Internet Engineering Task Force (IETF), September 2025. https://datatracker.ietf.org/doc/draft-ietf-tls-trust-anchor-ids/

[63] Red Hat, "Fedora project", https://fedoraproject.org/, Accessed: 2026-01-17
[64] QUBIP, "pq-container: Fedora reference container image for post-quantum cryptography", https://github.com/QUBIP/pq-container, Accessed: 2026-01-17
[65] QUBIP, "qubip_aurora: A rust crate providing a framework to build openssl providers for post-quantum cryptography", https://crates.io/crates/qubip_aurora, Accessed: 2026-01-17
[66] A. Mehmood and N. Tuveri, "Integrating PQC in OpenSSL via shallow providers for cryptographic agility", Cryptology ePrint Archive, Paper 2025/2082, 2025, https://eprint.iacr.org/2025/2082
[67] RedHat, "Kryoptic: A PKCS#11 Software Token Written in Rust", https://github.com/latchset/kryoptic, Accessed: 2026-01-17
[68] Cryspen, "libcrux: The formally verified crypto library", https://cryspen.com/libcrux-library/, Accessed: 2026-01-17
[69] StatCounter Global Stats, "Desktop Browser Market Share Worldwide", https://gs.statcounter.com/browser-market-share/desktop/worldwide, Accessed: 24 September 2025
[70] Mozilla, "Security and Networking Components", https://firefox-source-docs.mozilla.org/networking/sec-necko-components.html, Accessed: 2025-09-24
[71] A. Grosskurth and M. Godfrey, "A reference architecture for web browsers", IEEE International Conference on Software Maintenance, ICSM, October 2005, pp. 661–664, DOI 10.1109/ICSM.2005.13
[72] Mozilla, "About NSPR", https://firefox-source-docs.mozilla.org/nspr/about_nspr.html#about-nspr, Accessed: 2025-09-26
[73] Mozilla, "An Overview of NSS Internals", https://nss-crypto.org/reference/security/nss/legacy/an_overview_of_nss_internals/index.html, Accessed: 2025-09-24
[74] Mozilla, "NSS API Guidelines", https://www-archive.mozilla.org/projects/security/pki/nss/nss-guidelines, 2008, Snapshot from April 21, 2008
[75] S. Santesson, M. Myers, R. Ankney, A. Malpani, S. Galperin, and C. Adams, "X.509 Internet Public Key Infrastructure Online Certificate Status Protocol (OCSP)", RFC 6960, June 2013, DOI 10.17487/RFC6960
[76] Mozilla Developer Network, "Mozilla Firefox Release Notes – Firefox 69", https://developer.mozilla.org/en-US/docs/Mozilla/Firefox/Releases/69, 2019, Accessed: 2026-01-17
[77] Mozilla Bugzilla, "Bug 1315460: Remove support for HTML Keygen", https://bugzilla.mozilla.org/show_bug.cgi?id=1315460, 2019, Accessed: 2026-01-17
[78] C2SP, "Project Wycheproof: Tests for Cryptographic Libraries Against Known Attacks", https://github.com/C2SP/wycheproof, Accessed: 2026-01-17
[79] European Union, "A Coordinated Implementation Roadmap for the Transition to Post-Quantum Cryptography", https://digital-strategy.ec.europa.eu/en/library/coordinated-implementation-roadmap-transition-post-quantum-cryptography, June 2025
[80] RustCrypto, "RustCrypto: Cryptographic Algorithms Written in Pure Rust", https://github.com/rustcrypto, Accessed: 2026-01-24
[81] J. Lloyd, "wycheproof-rs: Wycheproof test vectors for rust", https://github.com/randombit/wycheproof-rs, Accessed: 2026-01-22
[82] K. Moriarty, M. Nyström, S. Parkinson, A. Rusch, and M. Scott, "PKCS #12: Personal Information Exchange Syntax v1.1", RFC 7292, July 2014, DOI 10.17487/RFC7292
[83] B. Kaliski, "Public-Key Cryptography Standards (PKCS) #8: Private-Key Information Syntax Specification Version 1.2", RFC 5208, May 2008, DOI 10.17487/RFC5208
[84] GnuTLS Project, "GnuTLS: Secure Communications Library", https://gnutls.org/, Accessed: 2026-01-24