MASTER'S THESIS

Determining SPHINCS+ Readiness for Standardization of SLH-DSA Signature

Student:
Jessica Ancillotti
jna1172@rit.edu

Co-Chairs
Professor Stanisław Radziszowski
Professor T.J. Borrelli
Observer
Professor Richard Lange

Department of Computer Science
May 30, 2025

Abstract

As quantum computing advances, public-key cryptographic algorithms risk becoming obsolete, requiring the development and implementation of quantum-resistant alternatives. This thesis evaluates SPHINCS+, a stateless hash-based digital signature scheme recently standardized by NIST under the FIPS 205 standard named Stateless Hash-Based Digital Signature Algorithm (SLH-DSA), which was selected for being a conservative and robust choice due to its reliance solely on well-understood cryptographic primitives. In the context of the growing need for quantum-resistant cryptographic solutions, determining the readiness of SPHINCS+ involves assessing its practical viability across different application domains and evaluating how well it meets today's and future needs. To achieve this, we analyze SPHINCS+ in the following areas.

Our analysis begins by benchmarking SPHINCS+ against classical and post-quantum signature schemes such as the Elliptic Curve Digital Signature Algorithm (ECDSA) and ML-DSA, measuring key generation, signing, and verification times, as well as signature sizes across multiple variants. The scheme's performance is evaluated in constrained environments such as Vehicle-to-Vehicle (V2V) communication, simulating its integration into mock blockchain transactions and TLS-like protocols to assess feasibility and scalability. In addition, this work investigates hybrid cryptographic models that combine SPHINCS+ with classical schemes to ensure backward compatibility and cryptographic agility, as a transitional solution during this post-quantum migration period.

The findings suggest that, while SPHINCS+ offers strong security against quantum computers and has been standardized by NIST, its substantial performance drawbacks raise critical concerns about its practicality. These limitations indicate that SPHINCS+, despite its conservative design, may not be suitable for many real-world applications. As such, we believe it is imperative that an alternative signature scheme be developed and standardized alongside the current selection to ensure feasibility across all types of applications.

Contents

List of Abbreviations

ADRS Address; page 22
AES Advanced Encryption Standard; page 15
BPP Bounded-error Probabilistic Polynomial Time; page 70
BQP Bounded-error Quantum Polynomial Time; page 35
DSA Digital Signature Algorithm; page 12
ECDSA Elliptic Curve Digital Signature Algorithm; page 9
FALCON Fast-Fourier Lattice-based Compact Signatures over NTRU; page 8
FIPS Federal Information Processing Standard; page 14
FFT Fast Fourier Transform; page 36
FFS Fast Fourier Sampling; page 36
FORS Forest of Random Subsets; page 8
FN-DSA FALCON Digital Signature Algorithm; page 8
Haraka Lightweight AES-based Hash Function; page 15
HORS Hash to Obtain Random Subset; page 14
LWE Learning With Errors; page 34
ML-DSA Module-Lattice-Based Digital Signature Algorithm; page 8
Module-LWE Module Learning With Errors; page 34
Module-SIS Module Short Integer Solution; page 34
NDSS Network and Distributed System Security Symposium; page 74
NIST National Institute of Standards and Technology; page 8
NTRU Number Theory Research Unit; page 36
NTT Number Theoretic Transform; page 36
OTS One-Time Signature; page 14
P2P Peer-to-Peer; page 50
PoS Proof of Stake; page 50
PoW Proof of Work; page 50
PQC Post-Quantum Cryptography; page 10
QC Quantum Computer/Quantum Computing; page 8
QRC Quantum-Resistant Cryptography; page A
RSA Rivest, Shamir, Adleman; page 8
SHA Secure Hash Algorithm; page 12
SHA2-256 Secure Hash Algorithm 2 Family; page 12
SHA3 & SHAKE Secure Hash Algorithm 3 Family; page 12
SHAKE256 SHA-3 Extendable Output Function (256-bit); page 12
SIS Short Integer Solution; page 34
SLH-DSA Stateless Hash-Based Digital Signature Algorithm; page 14
SVP Shortest Vector Problem; page 34
TLS Transport Layer Security; page 10
TX Transaction; page 51
UTXOs Unspent Transaction Outputs; page 51
V2V Vehicle-to-Vehicle; page 10
VM Virtual Machine; page 45
WOTS & WOTS+ Winternitz One-Time Signature (+ variant); page 14
XMSS eXtended Merkle Signature Scheme; page 14
ZKPoK Zero-Knowledge Proof of Knowledge; page 19

List of Figures

List of Tables

1 Introduction

In recent years, quantum computing (QC) has garnered significant research interest due to its potential to tackle complex mathematical problems that are infeasible for classical computers to solve at scale. These advancements, while promising for innovation, pose a serious threat to modern cryptographic systems. Should a general-purpose, large-scale, fault-tolerant quantum computer become a reality, foundational cryptographic protocols such as RSA [33], Diffie-Hellman, and Elliptic Curve Cryptography [33] would be rendered insecure, necessitating the urgent development of quantum-resistant alternatives [38]. More details on quantum computers can be found in Appendix A.1. In response to this potential threat, the National Institute of Standards and Technology (NIST) has proactively launched an initiative to identify and standardize quantum-resistant public-key cryptographic algorithms [42].

SPHINCS+ is a stateless hash-based signature framework, submitted to the NIST search in 2017, and has since become standardized [47]. SPHINCS+ has made several contributions, including tweakable hash functions and the development of the few-time signature scheme, Forest of Random Subsets (FORS). NIST also selected CRYSTALS-Kyber (ML-KEM) [45] and HQC [36] as the public-key encapsulation mechanisms and two other digital signature schemes for standardization: CRYSTALS-Dilithium (ML-DSA) [44] and FALCON (FN-DSA) [16]. Among these, SPHINCS+ stands out as the only scheme not based on the computational hardness of problems involving structured lattices [1].

2 Thesis Overview

This thesis, titled “Determining SPHINCS+ Readiness for Standardization,” investigates the viability of SPHINCS+ as a post-quantum cryptographic standard in light of its recent endorsement by NIST [50]. The study is motivated by the looming threat posed by quantum computing to classical public-key cryptographic systems, necessitating the development and adoption of quantum-resistant solutions. SPHINCS+ represents a hash-based, stateless signature scheme, designed to provide long-term security and resilience against quantum attacks.

The research is structured around four primary objectives. First, SPHINCS+ is compared against other NIST post-quantum cryptography finalists, analyzing its ease of integration, compatibility with existing systems, and practicality for broad adoption. Second, the thesis explores hybrid cryptographic schemes, which combine classical and quantum-resistant algorithms, as a transitional solution to ensure robust security during the transition to post-quantum systems. Drawing on insights from prior work in post-quantum authentication for vehicle-to-vehicle (V2V) communications [54], which found SPHINCS+-SHA2-256s (Robust) to be impractical due to its large signature size and slow verification times, this study further evaluates the SPHINCS+-SHA2-256f (Simple) variant.

Building on this analysis, the third objective focuses on evaluating SPHINCS+ within blockchain systems, specifically as a potential replacement for traditional signature schemes like the Elliptic Curve Digital Signature Algorithm (ECDSA) in Bitcoin. This segment assesses the performance, scalability, and compatibility of SPHINCS+ in critical blockchain protocols, positioning it as a candidate for securing future blockchain infrastructures.

Finally, this thesis investigates the limitations of SPHINCS+ in constrained environments, where factors can impose strict operational requirements. By examining its performance in V2V and TLS settings, the study identifies key scenarios where SPHINCS+ may fall short and where alternative schemes may be more appropriate.

Overall, this thesis concludes that while SPHINCS+ is cryptographically robust and necessary as a conservative fallback, it is not yet practical for widespread adoption due to severe performance and size limitations. Its standardization should be accompanied by the development of a more efficient, non-lattice-based alternative to ensure security, scalability, and post-quantum resilience across multiple applications.

2.1 Objectives

2.1.1 Comparative Analysis of SPHINCS+

Compare SPHINCS+ with other NIST post-quantum cryptography finalists, focusing on size constraints and performance across key generation, signing, and verifying messages of varying sizes. The analysis revealed that lattice-based schemes like FALCON and ML-DSA provide favorable performance in terms of signing and verification times, with FALCON excelling in compactness and speed. In contrast, SPHINCS+, while offering strong cryptographic assurances rooted in conservative, non-algebraic assumptions, exhibited significantly higher signing costs and large signature sizes, particularly in its robust variant.

2.1.2 Hybrid Cryptographic Schemes

To assess the potential benefits of hybrid schemes in transitional periods where both classical and post-quantum cryptographic solutions are necessary for secure systems. This evaluation demonstrated that hybrid signatures, such as ECDSA combined with SPHINCS+, can enhance cryptographic resilience during the quantum transition. However, the substantial increase in signature size, driven primarily by SPHINCS+, renders such schemes impractical for bandwidth-constrained applications like DSRC-based V2V communication. The analysis confirms there is an urgent need for compact, non-lattice-based alternatives.

2.1.3 SPHINCS+ in Blockchain Systems

Building upon the assessment of hybrid schemes, this objective narrows in on evaluating SPHINCS+ as a candidate to replace traditional cryptographic algorithms (e.g., ECDSA in Bitcoin) within blockchain systems. As blockchains are increasingly viewed as critical infrastructures, transitioning to quantum-safe algorithms like SPHINCS+ is essential to protect against future quantum threats. This objective determines whether SPHINCS+ offers a feasible alternative to current standards regarding performance, scalability, and compatibility with blockchain protocols. Our findings reveal that while SPHINCS+ meets post-quantum security requirements, its exceptionally large signature sizes introduce severe scalability and cost challenges. When applied in systems like Bitcoin, SPHINCS+ drastically reduces transaction throughput and inflates fee structures beyond practical limits, making it unsuitable for mainstream blockchain adoption without significant modifications to protocol constraints.

2.1.4 Bandwidth and Latency Constraint

The paper "When Cryptography Needs a Hand: Practical Post-Quantum Authentication for V2V Communications" [54] provides valuable insights into the limitations of SPHINCS+ for use in bandwidth-limited, real-time applications. Specifically, it concludes that SPHINCS+ is unsuitable for V2V communications due to its large signature sizes and longer processing times, incompatible with the stringent latency and spectrum constraints of V2V environments. This finding prompts further investigation into whether SPHINCS+ may face similar challenges in other critical areas requiring post-quantum cryptographic (PQC) solutions. Our investigation confirms that SPHINCS+ is infeasible for constrained environments like C-V2X, where its performance causes unacceptable latency and bandwidth overhead. However, in more flexible contexts like TLS 1.3, SPHINCS+ remains viable, though far less efficient compared to FALCON and ML-DSA. These findings reinforce SPHINCS+'s role as a fallback rather than a first-choice option for post-quantum deployment while still showcasing the urgent need for a non-lattice-based alternative.

3 History

3.1 Properties of Cryptographic Hash Functions

A cryptographic hash function is a publicly known function that compresses an input message of arbitrary length into a short, fixed-length, random-looking output called a message digest [49]. Cryptographic hash functions have certain requirements they must meet in order to meet the NIST standards. The cryptographic hash functions must be:

1) Collision Resistant: It is computationally infeasible to find any two distinct message inputs, $m_1 \neq m_2$, such that $\text{Hash}(m_1) = \text{Hash}(m_2)$ [49]. The birthday paradox illustrates why collisions are possible even when hash functions are well-designed. In a space of 2^n possible outputs, randomly selecting about $2^{n/2}$ distinct inputs leads to a high probability of at least one collision [49]. This phenomenon defines the birthday attack and imposes a lower bound on digest sizes for security. For instance, a 256-bit hash function like SHA-256 offers about 128 bits of collision resistance. A security level of 128 bits means that an attacker would need to perform roughly 2^{128} operations to find a collision, which is currently considered computationally infeasible.

2) Preimage Resistant: Given a hash output h, it must be computationally infeasible to find any input message m such that $h = \text{Hash}(m)$. In other words, given only the digest, one cannot derive any message that maps to it.

3) Second Preimage Resistant: Given an input m_1, it should be computationally infeasible to find another input $m_2 \neq m_1$ such that $\text{Hash}(m_1) = \text{Hash}(m_2)$. This guards against generating a new input that produces the same hash as a known message [15].

4) Efficient: $\text{Hash}(m)$ is relatively easy to compute [39].

5) Arbitrary Message Size: $\text{Hash}(m)$ can be applied to messages m of varying length, up to a maximum determined by the specific hash function's design [34]. For instance, SHA-256 supports inputs up to $2^{64}-1$ bits, while SHA-512 allows up to $2^{128}-1$ bits. Functions like SHA-3 or SHAKE256 have similar support [34].

6) Fixed Output Size: $\text{Hash}(m)$ produces a hash value z of fixed length, regardless of the size of the input message m [39]. For example, SHA-256 always outputs a 256-bit digest, while SHA-512 outputs 512 bits [34].

NIST currently recommends hash functions from the SHA-2 and SHA-3 families for cryptographic applications. SHA-2 includes SHA-224, SHA-256, SHA-384, and SHA-512, all based on the Merkle–Damgård construction [34]. SHA-3, based on the Keccak sponge construction, offers similar digest lengths but improved resistance to certain attack vectors and supports variable length output through SHAKE128 and SHAKE256 functions [35].

3.2 Digital Signatures

Digital Signatures provide a way for a receiver to verify that a message being sent to them is from the sender. This is done by digitally signing the message before it is sent to the receiver. We can visually see this in Figure 1. Alice would create her private and public keys, sign the message using the private key, and then send the signed message and the signature to Bob. Bob is then able to verify the message using Alice's public key and can confirm the validity of the signature. Digital Signature Algorithm (DSA) is one popular method that provides message integrity, authentication, and non-repudiation [38]. DSA utilizes a hashing function component to create a message digest that is signed with SHA. There are some important properties needed when using a hash function in a cryptographic scheme.

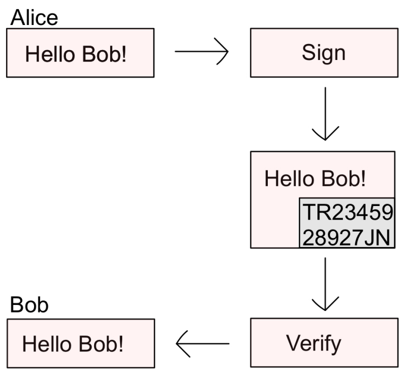


Figure 1: Digital Signature Example

3.3 Digital Signatures with Hash Functions

To illustrate how digital signatures use hash functions, we can look at Figure 2. The sender, Alice, first transforms the message to be sent into a digested form using a hash function such as SHA-256. This condensed output is known as a "message digest." Alice then encrypts this message digest with her private key. This encrypted digest serves as her digital signature. Both the original message and the signed digest are then sent to the recipient, Bob. Upon receiving them, Bob can use Alice's public key to decrypt the signature, recovering the message digest, similar to what we described in Figure 1. Bob then independently hashes the original message and compares his result to the decrypted digest; if they match, Bob is confident that the message is authentic and has not been tampered with.

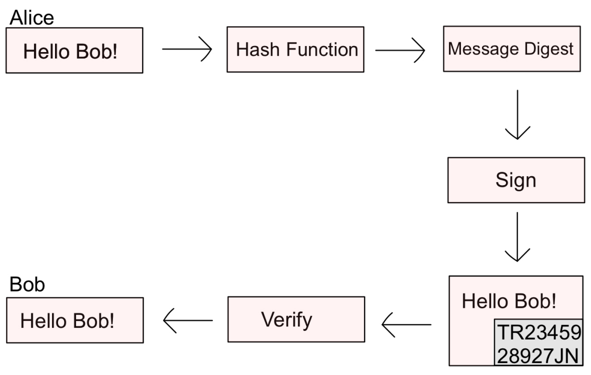


Figure 2: Digital Signature w/ Hash Example

3.4 SPHINCS

Originally, SPHINCS [6] was designed by Bernstein, Hopwood, Hulsing, Lange, Niederhagen, Papachristodoulou, Schneider, Schwabe, and Wilcox-O'Hearn as a stateless hash-based signature scheme. In the original design of SPHINCS, there are some components that SPHINCS+ tries to improve upon. SPHINCS uses Hash to Obtain a Random Subset, HORS/HORST to sign messages, whereas SPHINCS+ uses FORS. See figure 5 on page 28, which illustrates the difference between HORS and FORS, which will be addressed later in this thesis. SPHINCS uses Winternitz One-Time Signature (WOTS) which SPHINCS+ further develops in WOTS+. WOTS+ introduces tweakable hash functions. SPHINCS+ also introduces verifiable index selection [19]. Part of this comes from the change of HORST/HORS to Forest of Random Subsets (FORS). In HORST, the index representing the keypair for signing the message was generated pseudo-randomly. Due to an inability to verify this index, someone, say Oscar, could easily attack by using one hashing computation on the HORST instances and exploit this. The index is now computed using the message digest, which is hashed with the public key and the pseudo-randomly generated R value. All of these together are connected directly to the FORS instance and only that instance. This helps enhance the security of the scheme.

3.5 State vs Stateless

Hash-based digital signature schemes fall into two main categories, stateful and stateless. A stateful hash-based signature, like XMSS, requires the signer to keep track of internal state, specifically ensuring that each one-time signature (OTS) key is used once [13]. The reuse of an OTS key can lead to signature forgery. However, stateful schemes offer smaller signature and key sizes, making them attractive for constrained environments where state management is feasible [13].

A stateless hash-based scheme, like SPHINCS+, eliminates the need for managing signing state by incorporating more redundancy and randomness into each signature [47]. This makes them inherently more secure against misuse but comes at the cost of significantly larger signatures and longer signing times. We will examine this cost throughout the later sections.

3.6 SPHINCS+ vs SLH-DSA

SLH-DSA, the standardized version of SPHINCS+, introduces several key changes aimed at improving clarity, consistency, and long-term security, which are described at the end of the FIPS 205 standard. We go into detail on the scheme as a whole in Section 5. Two new address types, called $WOTS\_PRF$ and $FORS\_PRF$, were added to improve domain separation during secret key generation for WOTS+ and FORS, respectively. Additionally, the public seed, $PK.seed$ was incorporated into the pseudo-random function, PRF, to mitigate multi-key attacks and enhance protection in multi-user environments [47].

For parameter sets using SHA2 at higher security categories, 3 and 5, the standard replaced SHA-256 with SHA-512, which addressed vulnerabilities related to achieving full Category 5 strength [47] in the following functions:

- H_{msg}, which is used to generate the digest of the message being signed.
- PRF_{msg}, which is the pseudo-random function, PRF, that generates the randomizer, R, for the randomized hashing when the message is being signed.

- T_l, which is the hash function that maps the $ln - byte$ message to an $n - byte$ message.
- H, which takes a $2n - byte$ message and behaves similar to T_l.

Furthermore, the inclusion of both R and $PK.seed$ in the $MGF1$ input when computing H_{msg} strengthens resistance against multi-target, long-message second preimage attacks, reinforcing the scheme's long-term robustness.

A notable adjustment included the formalization of domain separation between signing raw messages and signing message digests. This modification, prompted by public feedback on the draft FIPS 205 standard released in August 2023, ensures that these two use cases are handled distinctly within the algorithm. The adjustment is reflected in the signature generation and verification procedures, where the input structure was revised to make this separation explicit. This reduces the risk of misuse or ambiguity in implementation.

Additionally, this standard modified the method of extracting FORS indices from the message digest. This change, made to align with the reference implementation, differs from the method described in earlier specifications, which contained ambiguities and inconsistencies. Line 6 of both the *wots_sign* and *wots_pkFromSig* procedures was also updated to match the reference code, correcting potential off-by-one errors in cases where $\log w \neq 4$.

Another significant change in SLH-DSA is the removal of the Haraka hash function as an option for the underlying cryptographic primitive.

Haraka is a lightweight cryptographic hash function specifically designed for high-speed hashing of short, fixed-length inputs [24]. Unlike general-purpose hash functions like SHA-3, which are optimized for long inputs and prioritize collision resistance, Haraka focuses solely on achieving strong preimage and second preimage resistance while sacrificing collision resistance to gain performance. By using hardware-accelerated Advanced Encryption Standard New Instructions (AES-NI) available on modern CPUs, Haraka achieves extremely low latency, in one CPU cycle per byte, even for inputs as short as 256 or 512 bits. Its construction uses a Davies-Meyer compression function built from multiple rounds of AES followed by lightweight mixing, with well-chosen round constants to avoid structural weaknesses that affected earlier versions of Haraka [24].

Haraka, while lightweight and efficient on constrained hardware, does not have a fully vetted security proof within the same framework as the other hash functions used in SPHINCS+, such as SHA-256 and SHAKE256. Its omission simplified the standard and avoids potential future concerns about its security margin. By focusing on well-analyzed hash functions with broader trust and support, SLH-DSA strengthens its position as a conservative and reliable choice for post-quantum digital signatures [24].

4 The Quantum Threat to Classical Cryptography

The emergence of quantum computing poses a fundamental threat to classical cryptographic systems. Algorithms once considered secure are vulnerable to quantum attacks. This section outlines the technical basis for these vulnerabilities by examining key quantum algorithms, including Shor's and Grover's, and their implications on classical cryptography. It also introduces the efforts led by NIST to define and standardize post-quantum cryptographic solutions in response to these unprecedented challenges.

4.1 Classical Collision Detection Attacks

Before the quantum era, various classical techniques were used to detect weaknesses in cryptographic functions. One such technique is Floyd's cycle detection algorithm, also known as the Tortoise and Hare algorithm [27]. Originally developed to detect cycles in sequences generated by iterated functions, it has been applied in cryptanalysis to identify collisions in hash functions or pseudo-random generators [37].

In the context of hash functions, Floyd's cycle detection algorithm exploits their deterministic nature by identifying when a repeated value occurs in a hash chain, thereby signaling a cycle [27]. This classical technique can uncover potential collisions or structural weaknesses in hash-based constructions. Although Floyd's algorithm runs in polynomial time and is highly space-efficient, its practical application is limited to reduced-round or weakened variants of cryptographic primitives due to the exponential complexity of real-world hash outputs.

Nevertheless, Floyd's algorithm exemplifies how classical tools can still be valuable in analyzing and stress-testing cryptographic designs. Its usage underscores the critical need for hash functions with long cycle lengths and strong structural resilience, attributes found in standardized families such as SHA-2 and SHA-3, which SPHINCS+ utilizes in its design under SHA-2 and SHAKE.

4.2 Shor's Algorithm and Public-Key Cryptography

Shor's algorithm, proposed by Peter Shor in 1994, enables efficient solutions to two key mathematical problems: integer factorization and discrete logarithms. These problems form the hardness assumptions of widely used public-key cryptosystems [39].

- The **RSA** algorithm relies on the computational difficulty of factoring a large composite number $N = pq$, where p and q are large primes [39].
- The **ECDSA** is based on the hardness of the Elliptic Curve Discrete Logarithm Problem, ECDLP, i.e., finding an integer k such that $Q = kP$, given points P and Q on an elliptic curve [39].

Shor's algorithm can solve both problems in *polynomial time* on a quantum computer. For instance, it can factor an n-bit integer in $O(n^3)$ time using a quantum circuit with $O(n^2)$ qubits. This renders RSA and ECDSA insecure against quantum adversaries, as the cryptographic assumptions they rely on no longer hold.

4.3 Grover's Algorithm and Symmetric Cryptography

Grover's algorithm, introduced by Lov Grover in 1996, provides a quadratic speedup for unstructured search problems [39]. In the context of symmetric cryptography, this translates to an attack model where a brute-force key search over a space of size 2^n can be performed in approximately $O(2^{n/2})$ time on a quantum computer.

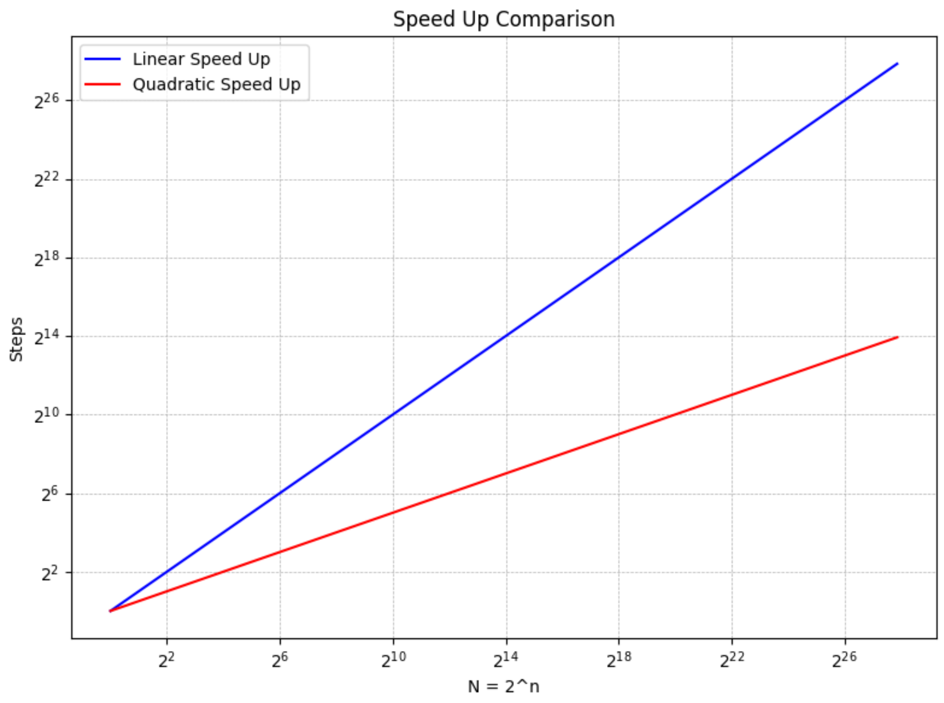


Figure 3: Grover's Speedup

As a result, symmetric algorithms such as AES [32] are not completely broken but their effective security is halved [39]. For example, a 128-bit key provides only 64 bits of quantum security. A 256-bit key is recommended to maintain a post-quantum security level equivalent to 128 classical bits.

This effect is a direct consequence of Grover's algorithm's complexity [21]:

$$\text{Quantum Search Complexity:}\quad O(\sqrt{N}) = O(2^{n/2})$$

Although Grover's algorithm weakens symmetric cryptography, it does not render it unusable. Increasing key lengths can effectively mitigate this threat, but 192 and 256-bit keys are included in the AES standard. Importantly, it has been shown that no quantum algorithm can perform unstructured search more efficiently than Grover's algorithm [5]. If the search space has structure, as in factoring integers or solving certain algebraic problems, then more efficient quantum algorithms, Shor's in particular, will outperform Grover's. But for truly unstructured problems, Grover's is optimal.

Together, these two algorithms pose threats:

- **Public-key cryptography** RSA, ECDSA is completely broken by Shor's algorithm.
- **Symmetric cryptography** AES is weakened by Grover's algorithm but remains viable with longer key lengths.

Impact on Hash Functions Grover's algorithm also generalizes to preimage search in cryptographic hash functions. Normally, finding an input m such that $H(m) = h$ requires $O(2^n)$ operations for an n-bit output h. Grover's algorithm reduces this to $O(2^{n/2})$. The same applies to second preimage attacks, finding $m_2 \neq m_1$ such that $H(m_1) = H(m_2)$.

- A 256-bit hash (e.g., SHA-256) offers 128-bit quantum preimage resistance.
- A 512-bit hash (e.g., SHA-512) is needed to achieve 256-bit quantum preimage resistance.

4.4 NIST PQC Search and On-Ramp

In 2017, NIST initiated a global effort to identify and standardize post-quantum cryptographic algorithms, anticipating the eventual threat posed by quantum computers to classical public-key systems like RSA and elliptic curve cryptography. As part of this effort, NIST established a security level framework based on symmetric-key benchmarks, shown in Table 1. These levels compare the computational difficulty of breaking a proposed PQC scheme to known symmetric cryptographic primitives, such as AES and SHA. This approach ensured that PQC algorithms would provide a consistent measure of strength ranging from general-purpose applications at Level 1 to highly sensitive systems requiring maximal protection at Level 5.

The requirements for standardization can be seen in Table 1. While NIST formally defines five distinct post-quantum cryptographic security levels, we believe that levels 2 and 4 are largely redundant. This is because a scheme designed to meet Level 3 (comparable to AES-192) also inherently satisfies Level 2, which is based on the weaker security benchmark of collision resistance for SHA-256. Similarly, a Level 5 scheme (AES-256 equivalent) exceeds the requirements of Level 4, based on SHA-384 collision resistance. Levels 2 and 4 are effectively subsumed by Levels 3 and 5. Our critique is that the inclusion of all five levels introduces unnecessary complexity. A more streamlined framework with only three meaningful levels, corresponding to the symmetric key strengths of 128, 192, and 256 bits, would have sufficed to guide both implementers and evaluators.

Level	Security Benchmark Description
1	Comparable to brute-force key search on a 128-bit symmetric cipher (e.g., AES-128)
2	Comparable to collision search on a 256-bit hash function (e.g., SHA-256 or SHA3-256)
3	Comparable to brute-force key search on a 192-bit symmetric cipher (e.g., AES-192)
4	Comparable to collision search on a 384-bit hash function (e.g., SHA-384 or SHA3-384)
5	Comparable to brute-force key search on a 256-bit symmetric cipher (e.g., AES-256)

Table 1: NIST Post-Quantum Cryptographic Security Levels [42]

In September 2022, NIST released a request for additional digital signatures, particularly ones that have short signatures and fast verification that are not lattice-based, though it would still consider lattice submissions. The NIST IR 8528 report outlines the progress of the First Round of the Additional Digital Signature Schemes for the NIST Post-Quantum Cryptography Standardization Process, culminating in the selection of 14 candidates to advance to the Second Round, shown in Table 2 [1].

Table 2: Second-round digital signature candidates organized by category

Code-Based	**Lattice-Based**	**MPC-in-the-Head**	**Multivariate**
CROSS	Mirath (MIRA/MiRiTH)	UOV	
LESS		MQOM	MAYO
		PERK	QR-UOV
Symmetric-Based	**Isogeny-Based**	RYDE	SNOVA
FAEST	SQIsign	SDitH	

Code-Based

Among the code-based submissions, CROSS stands out for its compact public keys and efficient performance compared to existing standards like SPHINCS+. CROSS uses a zero-knowledge proof of knowledge based on restricted syndrome decoding problems, while LESS focuses on the linear equivalence problem. LESS signature sizes are smaller than SPHINCS+ but suffer from a much larger public key. In both cases, the designs demonstrate promising trade-offs between performance and security, although their underlying assumptions still require more scrutiny from the community.

MPC-in-the-Head

Several schemes rely on the MPC-in-the-Head, MPCitH, paradigm, including Mirath, MQOM, PERK, RYDE, and SDitH. These approaches enable interactive zero-knowledge proofs compiled into non-interactive signature schemes via the Fiat-Shamir transform.

Mirath, a combination of MIRA and MiRitH, is based on the hardness of the MinRank problem. Both MIRA and MiRitH generate signatures by applying the Fiat-Shamir transform to a zero-knowledge proof of knowledge, ZKPoK, for solving MinRank. They use a hypercube optimization that enables parallel computation and reduces signature size, though it increases computation time. Their key sizes fall between those of SLH-DSA and FALCON, and their signature sizes are similar to SLH-DSA. Signing and verification speeds are faster than SLH-DSA and close to ML-DSA and FALCON.

MQOM is a signature scheme based on the MPC-in-the-Head, MPCitH approach, relying on the difficulty of solving random multivariate quadratic equations with equal numbers of variables and equations. Although recent research may slightly affect the parameters needed for its target security, the scheme remains grounded in well-studied hardness assumptions. MQOM features very small public keys and signature sizes that are between those of ML-DSA and SLH-DSA.

PERK security relies on the Permuted Kernel Problem, which is believed to be hard over finite fields. PERK uses MPC-style techniques to build a ZKPoK, where randomness is derived from hashing the message. While early versions of PERK relied on a possibly stronger assumption, updates have improved the proof system and significantly reduced the signature size. Despite these improvements, PERK remains slower than ML-DSA and closer in speed to SLH-DSA, with a signature size smaller than SLH-DSA but larger than ML-DSA.

Ryde is based on the hard Rank Syndrome Decoding problem, which involves solving a system of linear equations over a finite field for a low-rank solution. In RYDE, the signer proves knowledge of such a solution using zero-knowledge techniques, with interaction removed via the Fiat-Shamir transform. It offers two signature variants using different MPCitH techniques: the hypercube method with additive secret sharing and Threshold-in-the-Head with linear threshold secret sharing.

SDitH (Syndrome Decoding in the Head) is a digital signature scheme based on the MPC-in-the-Head approach and the hardness of the syndrome decoding problem. It comes in two variants: the hypercube variant, which produces smaller signatures but requires more computation, and the threshold variant, which reveals fewer internal values for verification. Although the original version slightly overestimated its security, updated parameters now ensure stronger guarantees. SDitH has very small key sizes and signature sizes that are between the "small" and "fast" options of SLH-DSA. While it isn't as fast as ML-DSA or FALCON, it performs better than SLH-DSA and is seen as a competitive option among similar schemes, though it could benefit from further security analysis.

Multivariate

In the multivariate category, UOV and its variants MAYO, QR-UOV, and SNOVA represent a mature class of schemes rooted in solving systems of quadratic equations. Unbalanced Oil and Vinegar, UOV, is the oldest unbroken multivariate digital signature scheme, built on a system of quadratic equations with a hidden structure that makes solving it difficult. It follows the hash-and-sign paradigm and provides strong EUF-CMA security. UOV is known for its very short signatures and extremely fast signing and verification times, making it ideal for use cases where speed and small signature size are critical. However, its major drawback is the large size of its public keys, even in configurations optimized for faster verification. Although UOV is a well-established scheme, a recent attack slightly weakened some parameter sets, prompting NIST to advise continued scrutiny of its security.

MAYO is a variant of the UOV signature scheme designed to reduce public key size by transforming a small quadratic map, called mini-UOV, into a larger one with UOV-like structure. This approach retains UOV's small signature size while significantly improving key size efficiency. Although MAYO is not as fast as UOV, it remains highly efficient overall. Some recent research on solving underdetermined multivariate systems may slightly affect its security, but it needs further analysis.

QR-UOV is a variant of the UOV signature scheme that reduces public key size by using quotient rings instead of standard finite fields. In QR-UOV, matrix elements are drawn from a field extension represented by a quotient ring, allowing each n x n block to be expressed with just n coefficients instead of n^2. This results in public keys that are roughly 50% smaller than those in UOV.

SNOVA is a signature scheme based on UOV that introduces extra structure to reduce public key size, using matrices over a noncommutative ring, square matrices over a finite field. It builds on the NOVA and SNOVA designs and achieves significantly smaller public keys that are comparable to or even smaller than those in MAYO, while maintaining relatively fast performance. Though slightly slower than MAYO, SNOVA remains efficient. However, it has faced multiple attacks during the First Round, with some parameter sets broken and others surviving. The unbroken sets, which also have the smallest public keys, remain promising.

Lattice Based

HAWK is a lattice-based hash-and-sign digital signature scheme closely related to FALCON. Like FALCON, it uses structured lattices and a secret basis to produce signatures by finding a lattice vector close to a hashed message. However, unlike FALCON, which uses floating-point arithmetic and the Fast Fourier Transform to protect its secret key, HAWK avoids floating-point operations by relying on two alternative hardness assumptions, the one-more Shortest Vector Problem (omSVP) and the search Module Lattice Isomorphism Problem (smLIP). These problems are less studied than FALCON's underlying assumptions, which is why NIST encourages further analysis of HAWK's security.

Symmetric-Based

FAEST is based on the VOLE-in-the-Head, VOLEitH, technique, a recent approach related to MPC-in-the-Head. Its security relies only on symmetric-key cryptography, using AES. The scheme constructs a ZKPoK using shared computations over the signing key, which is then made non-interactive via the Fiat-Shamir transform. Thanks to VOLEitH, FAEST achieves very small signatures and public keys, as small as 32–64 bytes, along with competitive signing and verification speeds. While not as fast as lattice-based schemes like ML-DSA, FAEST significantly outperforms the hash-based scheme SLH-DSA.

Isogeny-Based

SQIsign is built on isogeny graphs of elliptic curves, relying on the hardness of computing secret isogenies rather than traditional number-theoretic or lattice-based problems. Like many other candidates, it uses the Fiat-Shamir transform to create a non-interactive signature from a zero-knowledge proof. Although isogeny-based cryptosystems have recently faced major setbacks, particularly the complete break of SIKE [10], SQIsign avoids those same vulnerabilities by not revealing extra information used in key establishment. SQIsign stands out for having the smallest combined public key and signature size among all first-round candidates, including ML-DSA and FALCON. However, it is slower in terms of computation, especially for signing, but verification remains faster.

In 2024, NIST published its initial draft for transitioning to PQC, outlining clear timelines for the deprecation and disallowance of digital signature algorithms that are vulnerable to quantum attacks [30]. As shown in Table 3, the commonly used algorithms, ECDSA, Edwards Curve Digital Signature Algorithm (EdDSA), and RSA, regardless of bit strength, are scheduled to be deprecated after 2030 and disallowed entirely after 2035. This transition plan emphasizes the urgency of adopting quantum-resistant alternatives now and not later.

Table 3: Quantum-vulnerable digital signature algorithms

Digital Signature Family	**Parameters**	**Transition**
ECDSA [FIPS186]	112 bits of security strength	Deprecated after 2030 Disallowed after 2035
	$\geq$ 128 bits of security strength	Disallowed after 2035
EdDSA [FIPS186]	$\geq$ 128 bits of security strength	Disallowed after 2035
RSA [FIPS186]	112 bits of security strength	Deprecated after 2030 Disallowed after 2035
	$\geq$ 128 bits of security strength	Disallowed after 2035

5 Detailed Review of SPHINCS+

5.1 Key Components of SPHINCS+

The components that make up SPHINCS+ are WOTS+, Extended Merkle Signature Scheme (XMSS), Hypertrees, and Forest of Random Subsets (FORS).

5.2 Tweakble Hash Function

SPHINCS+ employs *tweakable hash functions* to ensure domain separation and secure computation throughout its stateless hash-based signature framework. A tweakable hash function is defined as $\mathrm{Th} : P \times T \times \{0,1\}^{\alpha} \to \{0,1\}^{n}$, where P denotes public parameters (e.g., the public seed `PK.seed`), T is a structured tweak, and $M \in \{0,1\}^{\alpha}$ is the message input [7]. This abstraction enables each hash invocation within the SPHINCS+ structure, spanning WOTS+, FORS, and Merkle trees, to be uniquely contextualized by its position and role. By embedding the address, ADRS, as a tweak and applying a bitmask derived from the public seed, SPHINCS+ ensures that no two hash function calls collide across different components, even when applied to the same message input. This construction not only facilitates a modular security reduction but also supports multi-target attack resistance under post-quantum security models. SPHINCS+ specifically adopts instantiations of these functions using either SHA-256 or SHAKE256 and organizes them by input length into functions such as $F = \mathrm{Th}_1$ and $H = \mathrm{Th}_2$, streamlining their application across the hypertree and few-time signature components [7].

5.3 Winternitz One-Time Signature Plus (WOTS+)

Winternitz One-Time Signature Plus, WOTS+, is a one-time signature scheme that has two main parameters, n, w. WOTS+ maintains the core ideas of WOTS, using a certain number of function chains that start from a random input, a secret key, and the end of chains are the public key [20].

5.3.1 Lamport OTS

Before we dive deeper into WOTS, it's important to have an understanding of Lamport. A signature scheme is a one-time signature scheme if it is secure when only one message is signed. That is, OTS can only be used once but can be verified multiple times. Lamport's basic definition is a message to be signed is a binary k-tuple of a fixed length and each bit is signed individually [49]. Let the message length be l. Then for all i,j where $1 \leq i \leq l$ and $j \epsilon (0,1)$, then $y_{ij} = f(z_{ij})$ A drawback is that this requires a very large key size, where the public key is 2k values to sign a $k - bit$ message.

Example of Lamport OTS from Stinson:
Let $p = 7879$, and the primitive element of Z_p be 3.
Then $f(x) = 3^x (\mod 7879)$ would follow as [49]

$$y_{1,0} = 5831, z_{1,0} = 2009$$
$$y_{1,1} = 735, z_{1,1} = 3810$$
$$y_{2,0} = 803, z_{2,0} = 4672$$
$$y_{2,1} = 2467, z_{2,1} = 4721$$
$$y_{3,0} = 4285, z_{3,0} = 268$$
$$y_{3,1} = 6449, z_{3,1} = 5731$$

If the message to sign is, x = (0,0,1).
The signature would be $(y_{1,0}, y_{2,0}, y_{3,1}) = (5831, 803, 6449)$

This can be verified using $f(x)$:

$$3^{5831} \bmod 7879 = 2009$$
$$3^{803} \bmod 7879 = 4672$$
$$3^{6449} \bmod 7879 = 5731$$

5.3.2 WOTS Example

To address the key size issues from Lamport OTS, Robert Winternitz proposed an improved scheme, now known as the Winternitz One-Time Signature, WOTS [55]. WOTS reduces the signature size by dividing the message digest into chunks and encoding each chunk using a base-w representation. Rather than signing each bit independently, it applies a hash chain to each chunk, which significantly compresses the key and signature sizes while maintaining security. Using the WOTS example below, which is from Stinson [49], we can show a basic outline of WOTS. Here, 3 bits are assigned, $w = 3$, and f is our secure hash function, which is collision-resistant. Since $k = 9$, three hash chains will be necessary, according to the formula $l = k/w$, where l is the number of hash chains. To construct the hash chains, one would follow the formula: $y^j = f^j(y_0)$ for $i \leq j \leq 7$ and $z = f(y_7)$. The 7 comes from the formula, 2^{w-1}, where $y^0, y^1, ..., y^{2w-1}$. Each hash chain is then constructed to produce the public key, in this case, (z_1, z_2, z_3). To verify the signature, all one needs to check is $f^{2w-x_1}(a_i) = z_i$.
There is a security issue with this process. The signature values are released, and Oscar now knows an element of the hash chain and, therefore, can compute the other values as well. This security issue is later fixed using the addition of a checksum to the message and signing process [49].

WOTS Example: Suppose that $k = 9$, $l = 3$, $w = 3$. Three hash chains:

$$y_1^0 \to y_1^1 \to y_1^2 \to y_1^3 \to y_1^4 \to y_1^5 \to y_1^6 \to y_1^7 \to z_1$$
$$y_2^0 \to y_2^1 \to y_2^2 \to y_2^3 \to y_2^4 \to y_2^5 \to y_2^6 \to y_2^7 \to z_2$$
$$y_3^0 \to y_3^1 \to y_3^2 \to y_3^3 \to y_3^4 \to y_3^5 \to y_3^6 \to y_3^7 \to z_3$$

Message to sign: 011101001
Signature: $x_1 = 011 = 3$, $x_2 = 101 = 5$, $x_3 = 001 = 1$
Released values: $a_1 = y_1^3$, $a_2 = y_2^5$, $a_3 = y_3^1$

Verify the signature:

$$f^5(a_1) = z_1$$
$$f^3(a_2) = z_2$$
$$f^7(a_3) = z_3$$

5.3.3 WOTS vs WOTS+

WOTS+ is an improvement of WOTS. The basic idea is to take the plain hash function used in WOTS for every hash chain, and instead use a tweaked hash function. There is an update made with SHAKE and SHA, so in addition to the input value will also take a public seed and an address. The hash function ensures there is domain separation between each member of the hash function family [7]. This allows you to use the same hash function in different areas, which helps to ensure a different result even if the message repeats. WOTS+ also adds randomization to WOTS by having the additional public key and the hash function take in multiple inputs instead. The tweakable hash function allows for easier analysis of the hash-based scheme. It's also able to limit attacks by not using l-trees for compression.

5.3.4 WOTS+ Construction

The first parameter is n, the length of the message in bytes. The second parameter is w, which is the Winternitz parameter and originally submitted with multiple potential values, element of the set $\{4, 16, 256\}$ [7]. The w parameter was limited to the set of $\{4, 16, 256\}$ because it was found that these values yield optimal trade-offs and are easy to implement. However, in the official standardization, we would like to note that the parameters only use 16 for w and fail to mention the other elements for the rest of their submission or why they were omitted.

The n parameter also includes the length of the private key, public key, and signature element in bytes. It determines the length of the message that WOTS+ can process and sign. The parameters that are recommended for n are shown in Table 4. These two parameters are then used to compute the len, or number of n-byte string elements in the WOTS+ private key, public key, and signature(1,2) [7].

$$len_1 = [8n/log(w)] \tag{1}$$

$$len_2 = [log_2(len_1 * (w - 1))/log(w)] + 1 \tag{2}$$

The WOTS+ construction is broken down into the chaining function, public key generation, signature generation, and verification using the signature. Private key generation happens within the chaining functions and public key generation and is not discussed much outside of these.
The chain function takes an $n - byte$ string, X, number of steps, s, and a starting index, i, as input. It also takes in ADRS as input, which is a 32-byte hash function address that can identify the position of the hash function call, the value being computed [7]. ADRS is used throughout all of the algorithms that make up SPHINCS+ and is meant as a way to keep track of where you are within all the trees. Lastly, the chain function takes in $PK.seed$, which is a public seed that was added later into the NIST submission as an input to help mitigate multi-key attacks and assist in domain separation between different key pairs [47]. This algorithm for the chaining function will produce the value of F iterated s times on X. F is the tweakable hash function that takes an n-byte message as input and produces an n-byte output [47]. Referring to the WOTS+ example, this is what the chaining function is trying to replicate, by making the hash chains. The chaining function will be used in the other algorithms for WOTS+.

Public Key Generation in WOTS+ will take in $Sk.seed$, $PK.seed$, and $ADRS$ and outputs a compressed version of the WOTS+ public key, pk. The algorithm will also compute the secret value and public value. For the generation of the public value, it will generate a corresponding secret value and use the chaining algorithm from above to compute the end value of the chain of length w [47]. Once these are computed, they will be compressed into a single n-byte value [49]. This compression is a bit different from the last nodes of the WOTS+ chains are not compressed using an L-tree but using a single tweakable hash function call. This function call, like the ones before it, receives an address and a public seed to key this call and to generate a bitmask as long as the input [19].

WOTS Signature Generation will take in the message, M, $SK.seed$, $PK.seed$, and $ADRS$ and return the WOTS+ signature. This operates by converting the n-byte message to $base_w$ and computing the checksum over M and appending it to the $base_w$ value found. The $base_w$ is essentially integers that belong to the set {0 to $w-1$}. The selected nodes are then concatenated to form the signature. Computing the checksum is just a basic checksum evaluation with the addition of w, eq(3,4). The checksum is important to help avoid forgery, as stated earlier, is an issue with basic Lamport.

$$for(i = 0; i < len_1; i++)\{ \tag{3}$$

$$csum = csum + w - 1 - msg[i]; \} \tag{4}$$

Verifying a WOTS+ signature can be done by computing a public key value from the signature. This is similar to the public key generation steps. The verifier would need to recompute the checksum, derive the chain lengths, and apply the hash function F to complete each chain to its full length [7]. This will then return the public key, which can be checked against the known public key. However, since WOTS+ is not just used on its own, the output value will be used in the grand scheme to verify the SPHINCS+ public key.

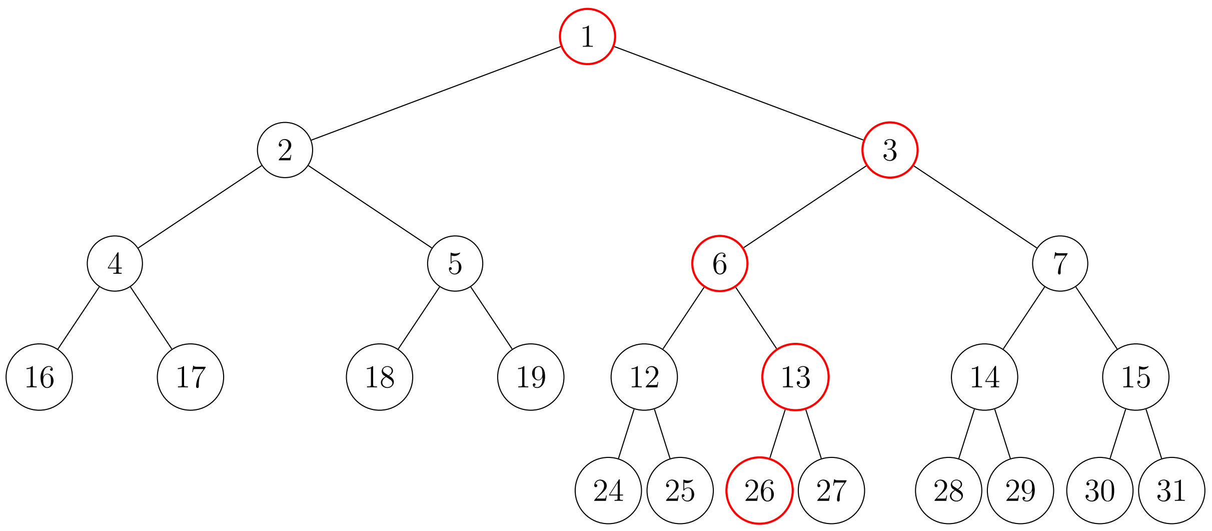
5.4 Merkle Signature Scheme (MSS) and More

Extended Merkle Trees, XMSS, are an important part of SPHINCS+. Merkle Signature Schemes are a useful method of extending a one-time scheme so it can be used with a larger number of signatures without increasing the public key size [26]. The idea behind the MSS is to create a binary tree, the Merkle Tree, by hashing combinations of various public keys of OTS schemes. The MSS purpose is to authenticate public keys, while WOTS+ is used for creating signatures.

5.4.1 An example of MSS

The example from Stinson [49] shown in the MSS Example shows creating a signature for the message 11, m_{11}. Node 26 is our first node because for our leaf nodes, it is number 11. From there, we follow the path up to the root note, 1. The public key, k_i, and private key s_i are included in the final signature of the MSS, as well as the siblings of the path nodes. The sibling nodes are represented as $V(i + 2^d - 1)$ where the value at each node, $V(j)$ is represented by the secure hash function value. The validation chain will be equal to K in the end if everything is correct and Oscar has not tampered with anything [3].

MSS Example Suppose $d = 4$, create a signature for message m_{11}:



Relevant path nodes: 26, 13, 6, 3, 1
Sibling nodes: $V(27)$, $V(12)$, $V(7)$, $V(2)$

Authenticate key K_{11}:

1. Compute $V(26) = h(K_{11})$
2. Compute $V(13) = h(V(26)\|V(27))$
3. Compute $V(6) = h(V(12)\|V(13))$
4. Compute $V(3) = h(V(6)\|V(7))$
5. Compute $V(1) = h(V(2)\|V(3))$
6. Verify that $V(1) = K$

Signature: $K_{11}, s_{11}, V(27), V(12), V(7), V(2)$

5.4.2 Extended Merkle Trees (XMSS)

XMSS is just an extension of the basic Merkle Signature Scheme. XMSS helps sign a larger number of messages. XMSS contains a few components, WOTS, PRF, which is a pseudo-random function that can generate a randomizer called R, which is then used for the hashing of the message to be signed, and also part of the tweakable hash function. XMSS also contains H, part of the tweakable hash function, and H_{msg}, which is used to generate the digest of the message. Each of the public and private key pairs is connected with a perfect binary tree [18].

The XMSS signature consists of the *height* + *the length* $*n$ *bytes*, WOTS+ signature and the authentication path. The authentication path is an array of nodes from each level that allows the verifier to compute the root of the tree when combined with the WOTS+ public key [47]. The authentication path is similar to that in the MSS Example example using sibling nodes. The sibling nodes will be those that are siblings to the nodes on the path from the WOTS+ key used up to the root. Once the authentication path is created, the n-byte message M is signed with the corresponding WOT+ key [49].

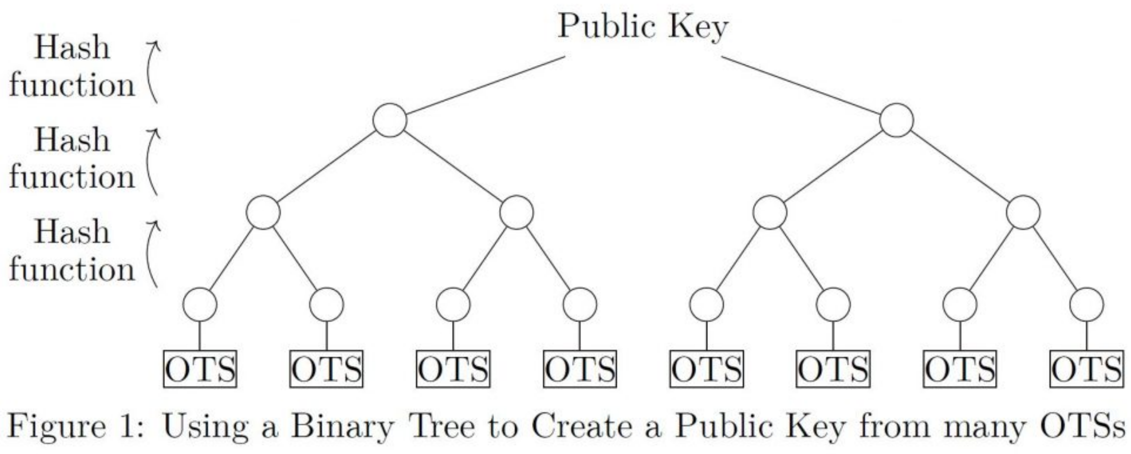


Figure 4: Merkle Tree with OTS [3]

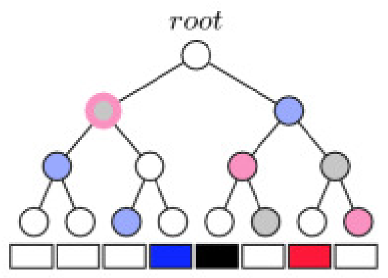
5.4.3 Connection to WOTS

In the example in Figure 4, h is the hash function, which is collision resistant. The hash function for us is part of the tweakable hash functions mentioned. To compute a parent node, you compute the hash of the two children nodes using a version of the tweak. The leaf nodes of XMSS are the WOTS+ public key. We can see it better by referring to Figure 3. Each leaf node is the hash output of the WOTS+ public key [47]. Every other node in the tree is the hash of the two child nodes, to create the parent node. The root node is the public key.

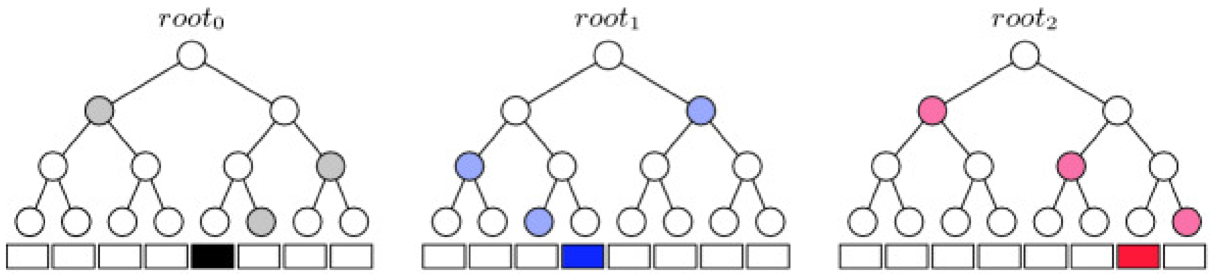
5.5 Hypertree

A hypertree is introduced to help with the signing of the FORS keys. A hypertree is simply a tree of XMSS trees. The lowest layer of these trees is used to sign FORS public keys, and the keys at the other layers are used to sign the XMSS public keys. The hypertree signature, which is used for the signing process in the end, has as input, the message M, $SK.seed$, $PK.seed$, and the index of the leaf that is being used to sign the message. After the signing algorithm is complete, it will return the $HTSignature$. The HT signature consists of the stack of XMSS signatures that were computed earlier. This is really all the signature from the WOTS+ steps to build up the tree. Essentially, everything is iteratively signed to create the HT signature.

The hypertree will consist of d layers, with the top layer being $d-1$, and consisting of a single XMSS tree. The XMSS tree height is represented by h', and in turn, the total height of the hypertree is $h = d * h'$. We can see this represented in the parameter set shown in Table 4. The second layer, $d-2$, has $2^{h'}$ trees and WOTS+ keys and the lowest layer contains $2^{h-h'}$ XMSS trees. At each layer from layer 0 to layer $d-2$, the public key for each XMSS key is signed by the XMSS key at the next layer higher than it. This is why an XMSS key consists of 2^h WOTS+ Keys. These WOTS+ keys are then used to sign the FORS public key in the key pair [47].



(a) HORS signature within a binary tree construction



(b) FORS signature within κ binary trees construction

Figure 5: HORS vs FORS Trees [14]

5.6 Forest of Random Subsets (FORS)

An improvement to SPHINCS is the addition of a Forest of Random Subsets, FORS. In the original SPHINCS, you would randomly pick one of the HORST trees to sign your message, which created better chances of forgery. However, with SPHINCS+, the mapping of the message is connected to both FORS leaves and the FORS signature. This gives less of a chance for someone, say, Oscar, to forge a message.

FORS is a few-time signature that can sign the digests of the actual message [47]. It's called a few times because it can sign a message a few times, but each time, information is exposed, which reduces the security of the key being used. FORS construction is similar to HORS/HORST, however, unlike HORS/HORST where there is just one tree constructed from the message digest, FORS has several, where each tree has a root that is all hashed to create the FORS public key, see Figure 5, [14].

Some key parameters for FORS are k, the number of trees, a, the height of k trees, and $t = 2^a$. Each k set is formed into a Merkle tree, and as stated earlier, the roots of these trees are hashed together to form the FORS public key. This can be seen in Figure 5 and even in Figure 6, which will be broken down in the next section.

In the generation of the FORS part of the Merkle Hash Tree, the FORS public values and signature will be generated. This is similar to the construction of the hypertree with WOTS nodes, except this time the leaf nodes are the FORS secret values instead of the WOTS+ public keys [47].

The algorithm works by taking in the input as $SK.seed$ $PK.seed$, the target node at index i, the target node height, z, and the $ADRS$. Each node in this tree is the root of a subtree, and this follows normal tree creation algorithms by computing right and left nodes and hashing them together to create the parent node. The lead nodes will return the secret key [47]. Again, this looks similar to Figure 5 FORS tree.

The creation of a FORS signature, again is similar to other signature creations. This time it will sign the message digest, which is generated using the Randomizer function, R. This algorithm will take as input, $SK.seed$ $PK.seed$, $ADRS$, and the message digest. Similarly to Fig 5, the message digest is split into k a bit strings, and similar to the WOTS+ $byte_w$, the integers are in this case between 0 and $t-1$ Each of these integers will then be mapped to a FORS private key, and then the authentication path is computed and the message is signed [7].

To verify a FORS signature, the verifier would need to compute the public key using the message digest and the signature. This is done similarly to WOTS+ verification. The FORS public key is always held within the bottom layer of the tree, layer 0, The verifier would need to compute the roots of each Merkle Tree. The authentication path that was generated in the signing process will be used to do this, along with the hash values of the private keys. Once the verifier has computed the roots, they can hash all the roots together to obtain the FORS public key [47].

6 SPHINCS+ High Level Overview

SPHINCS+ uses hypertree and FORS to create the stateless hash-based signature scheme. The public key for SPHINCS+ is made up of the *PK.seed*, and the *PK.root*, the root of the hypertree. The private key is made up of the *SK.seed*, *SK.prf*, *PK.seed*, and *PK.root*. *PK.seed*, *SK.seed*, and *SK.prf* are all randomly generated using a NIST-approved bit generator. *SK.seed* is what we used above to generate the WOTS+ and FORS private key elements. *SK.prf* is used to generate the randomizer, R, for the hashing of the message [47]. The R value is generated by taking the *SK.prf*, the message, and an n-byte value called OptRand, which is a 256 bit value that is defaulted to 0 [3]. This allows for deterministic signing to be optional, since you can fill the OptRand value with random bits using TRNG. Deterministic signing is not always wanted, so having the option to turn it off makes SPHINCS+ appealing. R is also not just a random value, as that could lead to bad random value being generated and making the signature susceptible to attacks. This R is part of the solution to having a verifiable index that was addressed in the beginning of this thesis. The key pair for SPHINCS+ is the first component to be created before signing. The randomly generated elements must be created first before the *PK.root* can be created.

The signature for SPHINCS+ contains 3 components. The randomizer, R, FORS signature, and the hypertree signature. To sign a message, you need to create the message digest using R. Then, extract the bits from the message, pair them with the appropriate FORS key, and sign the message using the hypertree signatures.

It can be difficult to see how these all connect, so we will refer to Figure 6 for a simplified breakdown. The squares represent the WOTS+ public keys, the circles represent the interior nodes of the hash tree, and the diamonds represent the FORS private key.

Starting from the bottom, say Alice wants to send a message to Bob. Alice's message will be hashed using the randomizer information, and the message digest will be mapped to the black diamonds on the bottom row. These black diamonds are the FORS private key that will all be hashed together to create the FORS root, this is the red circle that represents the FORS public key. From here, there is a mini gap in between the green square and the red circle. This is where the WOTS+ signature comes in. It will sign the FORS public key. The signature will use the info from the WOTS+ public key in the green square. Following the MSS construction, the nodes are hashed to create the parent nodes up to the next root, colored in yellow. In each of these yellow roots, there is a corresponding green WOTS+ public key square. These intermediate areas, again, are where the WOTS+ signature comes in and signs the yellow root, using the info from the corresponding public key leaf node. It repeats all of this, iteratively goes up the tree until the *PK.root* is reached and the signature has been created and can be sent off to Bob.

Bob can then verify it using the verification process, which is very similar to what was just described above. The only difference is that Bob has the message and signature and is essentially unpacking the signature to generate the public key that can be found at the root, and then compares the two to make sure they match. If they do, then it's verified; if not, then the message has been tampered with.

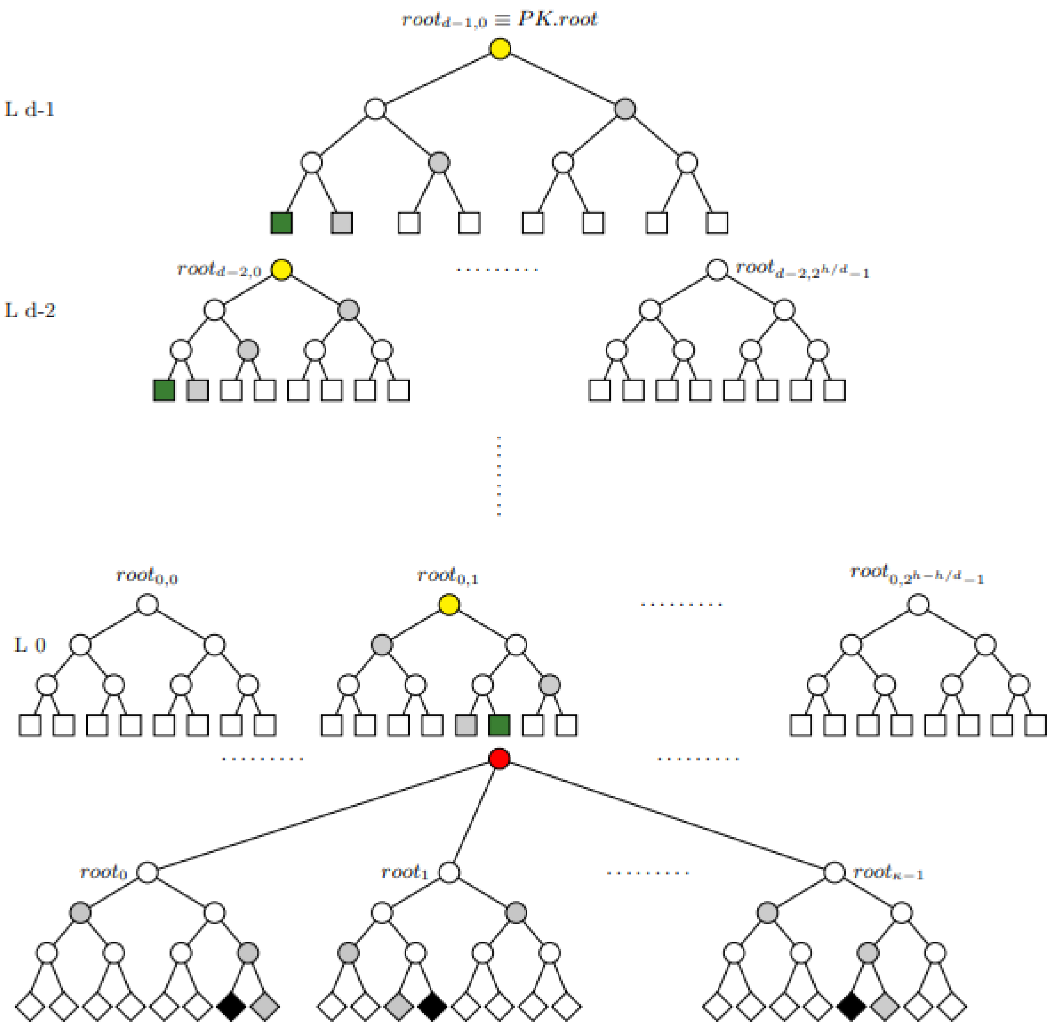


Figure 6: SPHINCS+ Breakdown [57]

6.1 Parameters

Table 4 presents the twelve approved parameter sets for the SLH-DSA scheme, as described in the NIST specification for post-quantum digital signatures [48].

Method This column lists the full name of the parameter set. Each method name includes the hash function used, SHA2 or SHAKE, which is under the SHA3 family, the target security strength (128, 192, or 256 bits), and a variant suffix: `s` denotes the “robust” variant with reduced signature size, while `f` indicates the “simple” variant optimized for signing speed.

n The length (in bytes) of the hash function output, and also the primary security parameter for the scheme. It influences the size of keys and internal state.

h The total height of the hypertree used in SLH-DSA. A higher tree height supports more signatures per key pair but also increases signature size and signing time.

d The number of layers in the hypertree.

h' The height of each XMSS tree for each layer d. Wehre $\frac{h}{d} = h'$

k The number of FORS trees used. A larger number of trees can provide higher security but increases signature size.

a The height of each FORS tree. $a = \log t$, where t is the number of leaves of a FORS tree.

$\log_2 w$ The base-2 logarithm of the Winternitz parameter w, where $w = 16$.

m The length (in bytes) of the digested message to be signed.

Security Level The NIST-defined post-quantum security category that the parameter set is designed to meet.

PK Bytes The size of the public key in bytes. Notably, SLH-DSA parameter sets keep the public key size fixed across all variants for implementation simplicity.

Sig Bytes The total size of the digital signature produced using the respective parameter set. “Robust” (`s`) variants yield more compact signatures, while “Simple” (`f`) variants produce larger signatures but improve signing throughput.

Table 4: SLH-DSA Parameter Sets [48]

Method	**n**	**h**	**d**	h'	**a**	**k**	$\log_2 w$	**m**	**Security Level**	**PK Bytes**	**Sig Bytes**
SLH-DSA-SHA2-128s	16	63	7	9	12	14	4	30	1	32	7,856
SLH-DSA-SHAKE-128s	16	63	7	9	12	14	4	30	1	32	7,856
SLH-DSA-SHA2-128f	16	66	22	3	6	33	4	34	1	32	17,088
SLH-DSA-SHAKE-128f	16	66	22	3	6	33	4	34	1	32	17,088
SLH-DSA-SHA2-192s	24	63	7	9	14	17	4	39	3	48	16,224
SLH-DSA-SHAKE-192s	24	63	7	9	14	17	4	39	3	48	16,224
SLH-DSA-SHA2-192f	24	66	22	3	8	33	4	42	3	48	35,664
SLH-DSA-SHAKE-192f	24	66	22	3	8	33	4	42	3	48	35,664
SLH-DSA-SHA2-256s	32	64	8	8	14	22	4	47	5	64	29,792
SLH-DSA-SHAKE-256s	32	64	8	8	14	22	4	47	5	64	29,792
SLH-DSA-SHA2-256f	32	68	17	4	9	35	4	49	5	64	49,856
SLH-DSA-SHAKE-256f	32	68	17	4	9	35	4	49	5	64	49,856

6.2 Fast and Small

In the parameter set, as shown in Table 4, there are small and fast versions for each, leading to 6 instances of this framework. Small, denoted with an s, was often referred to as Robust in earlier versions of the SPHINCS+ proposal, [3], and Fast, denoted with an f, was referred to as Simple. The Fast/Simple version has the advantage of better speed; however, the drawback of a larger signature. The Small/Robust version has a much smaller signature size, but is slower since it contains all the components of SPHINCS+ [47]. The Fast/Simple version omits the bitmasks part, which means that no bitmasks need to be generated and XORed with the message input of the tweakable hash functions [47].

6.3 Security

Since SPHINCS+ is designed to remain secure even in the presence of quantum-capable adversaries, it derives its security not from number-theoretic assumptions like ML-DSA and FALCON, but from the properties of its underlying hash functions. As we demonstrated in Section 5, SPHINCS+ builds a hypertree structure that is composed of many Merkle subtrees, where each layer leverages cryptographic hash functions like SHA-256, SHA-512, and SHAKE256. These functions provide preimage and second preimage resistance of approximately 2^n and collision resistance of $2^{n/2}$, where n is the output length in bits of the underlying cryptographic hash functions. As we saw in Section 4 with Grover's algorithm reducing the security strength of hash functions, by choosing SHA-256, $n = 256$, therefore SPHINCS+ maintains approximately $128 - bit$ quantum security.

7 Analysis of SPHINCS+ vs NIST PQC Finalists

NIST selected three digital signature schemes to be standardized: ML-DSA, FALCON, and SPHINCS+ [46]. These candidates differ significantly in their design principles, cryptographic assumptions, performance profiles, and implementation characteristics. A deeper comparative analysis reveals trade-offs between efficiency, security assumptions, and implementation complexity.

ML-DSA is a lattice-based signature scheme built on the Module Learning with Errors (Module-LWE), which is believed to be hard to solve even for quantum algorithms [44]. ML-DSA signatures are relatively short and fast to verify, making the scheme well-suited for general-purpose use. Protecting ML-DSA implementations against side-channel attacks is essential to ensure the confidentiality of private keys. To mitigate such threats, the design incorporates blinding, which introduces randomness into secret-dependent computations to obscure data-dependent patterns, and constant-time implementations, which ensure that execution time does not vary based on secret values. These techniques help prevent timing and power analysis attacks. The design also emphasizes simplicity and robustness by avoiding operations such as floating-point arithmetic, which can lead to implementation inconsistencies and potential vulnerabilities [44].

FALCON is also based on lattice cryptography, specifically on the NTRU lattice and relies on the Fast Fourier Sampling technique for efficient signature generation [16]. FALCON is notable for its compact signature size and high verification speed, especially in high-security settings. However, it requires floating-point operations and careful implementation to maintain numerical precision and side-channel resistance [28].

SPHINCS+ differs fundamentally from FALCON and ML-DSA in that it is a hash-based signature scheme [48]. Its security relies solely on the security of cryptographic hash functions, making it the most conservative choice among the finalists. SPHINCS+ offers strong post-quantum guarantees and avoids reliance on more recent algebraic hard problems, given reasonable assumptions such as the continued preimage and second-preimage resistance of SHA-2 and SHAKE256. These functions are expected to withstand quantum adversaries, with Grover's algorithm offering only a quadratic speedup, as discussed in Section 4. However, this conservative design comes at a cost: SPHINCS+ signatures are significantly larger, and the scheme is slower than lattice-based alternatives, something we will prove to you through these objectives. Despite this, its stateless and hash-based nature makes it a fallback option where long-term security guarantees outweigh efficiency concerns.

7.1 Lattice Foundations

ML-DSA and FALCON are both based on lattice problems, which are believed to be resistant to quantum attacks. ML-DSA leverages the Module Learning With Errors (Module-LWE) and Module Short Integer Solution (Module-SIS) problems, making it a derivative of the more general Ring-LWE framework.

The LWE problem is a foundational hard problem in post-quantum cryptography. It can be described as follows: Given a matrix A and a vector z, it is computationally hard to recover the pair $(\mathbf{y}, \mathbf{e})$ such that $A\mathbf{y} + \mathbf{e} = \mathbf{z} \bmod p$, where $\mathbf{y}$ is a secret vector and $\mathbf{e}$ is a small error vector. This hardness persists even for quantum adversaries, making LWE an attractive basis for constructing quantum-resistant cryptographic primitives, including digital signature schemes. The problem is closely related to lattices. Each valid

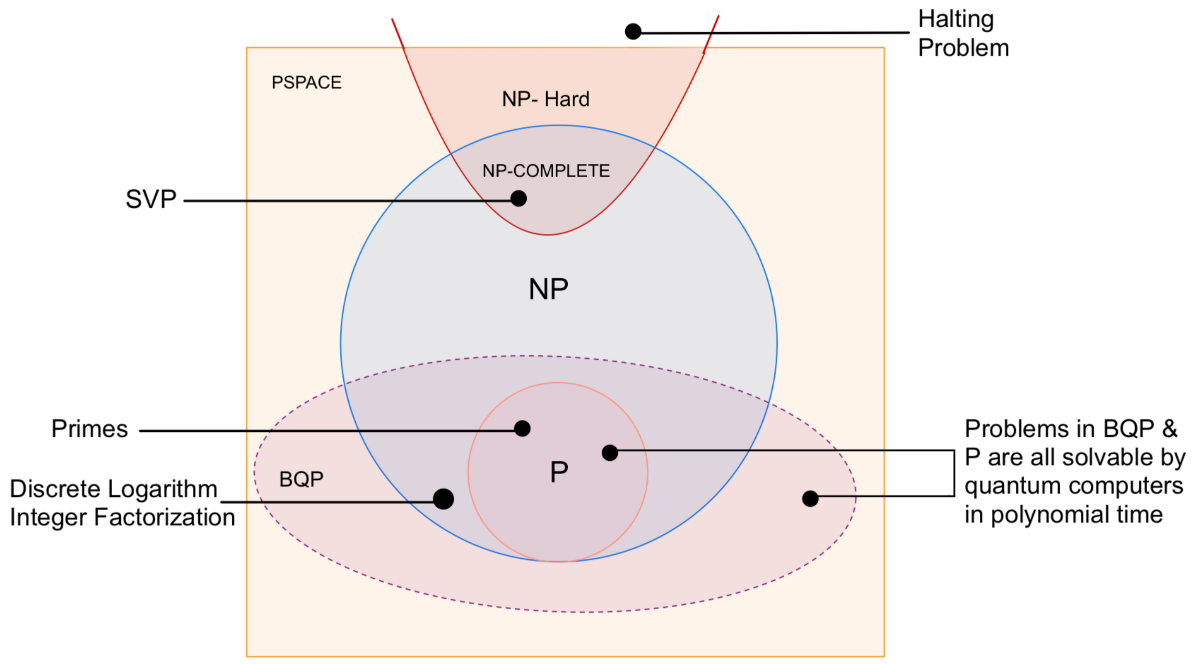


Figure 7: Computational Complexity

pair $(\mathbf{y}, \mathbf{e})$ corresponds to a point in a shifted lattice. Recovering the original secret $\mathbf{y}$ involves finding the lattice point closest to a given target, which is known as the Bounded Distance Decoding (BDD) problem, an NP-hard problem [9].

The security of lattice-based cryptographic systems is fundamentally rooted in the presumed hardness of two core problems, the Learning With Errors (LWE) problem we just saw, and the Short Integer Solution (SIS) problem. The SIS problem is closely related to the well-known Shortest Vector Problem (SVP), as both involve finding short vectors within a lattice structure. A key commonality among LWE, SIS, and SVP is that they are all believed to be computationally hard problems. Importantly, many of these problems are known to be NP-hard, and no quantum algorithm is currently known to solve them efficiently. In fact, the complexity class BQP (Bounded-error Quantum Polynomial time), which captures problems efficiently solvable by quantum computers, is not known to contain any NP-hard problems, see Figure 7. More information on BQP can be found in Appendix A.2. If cryptographic parameters are chosen appropriately, both LWE and SIS are conjectured to be intractable even for quantum adversaries.

Though FALCON is a lattice-based system that also relies on the hardness of the SIS and SVP problems, FALCON's components utilize the NTRU problem and use Fast Fourier Sampling (FFS) techniques to generate signatures.

NTRU was introduced by Hoffstein, Pipher, and Silverman and is believed to be derived from the pun Number Theorists 'R' Us, or now stands for Number Theory Research Unit. NTRU is based on lattice mathematics, specifically leveraging the hardness of SVP and Closest Vector Problem (CVP). The scheme operates over truncated polynomial rings, where messages and keys are represented as polynomials with small integer coefficients [9]. A major advantage of NTRU is its speed and compact key sizes relative to other post-quantum schemes, though it can experience rare decryption errors and is vulnerable to certain lattice-based attacks if not properly parameterized.

NTRU Example

Suppose Bob wants to receive a secure message from Alice [9]. He begins by selecting small secret polynomials f and g such that f has a multiplicative inverse modulo of both p and q. Using these, he computes his public key:

$$h = (f^{-1} \bmod q) \circledast g \bmod q$$

Bob publishes the tuple (N, p, q, h) as his public key.
Alice wants to send a message m to Bob. She encodes her message as a polynomial m with small integer coefficients. Then, she randomly selects a small polynomial ϕ and computes the ciphertext:

$$y = p\phi \circledast h + m \bmod q$$

Alice sends the ciphertext y to Bob.

For decryption, NTRU requires a center modular reduction, or better known as center-lifted [9]. Center-lifting is a technique that maps an integer modulo p to an equivalent value b such that:

$$a \bmod p = b \text{ if } a \equiv b(\bmod p) \text{ and } -\frac{p-1}{2} \leq b \leq \frac{p-1}{2} \tag{5}$$

Center-Lifting Example: Let $p = 5$.

$$\frac{5-1}{2} = 2 \quad \Rightarrow \quad \text{centered range} = \{-2, -1, 0, 1, 2\}$$

This range contains 5 values, just like $\mathbb{Z}_5$.
Continuing with NTRU decryption, Bob will then perform the following steps:

1. Compute:
$$a = f \circledast y \bmod q$$
2. Center-lift the coefficients of a to fall within the range $\left[-\frac{q-1}{2}, \frac{q-1}{2}\right]$.
3. Recover the original message:
$$m = f^{-1} \circledast a \bmod p$$

FALCON then introduces trapdoor sampling techniques to construct compact and efficient signatures with tight security proofs, which leverage the Fast Fourier Transform (FFT) sampling to efficiently sample short lattice vectors from a discrete Gaussian distribution. The authors of FALCON call this technique Fast Fourier Sampling (FFS) [16].

FFT is efficient at computing the Discrete Fourier Transform (DFT), which converts a signal from the time domain to the frequency domain. This transformation is fundamental in signal processing, enabling analysis of frequency components within data. While the naive computation of the DFT requires $O(n^2)$ operations, the FFT reduces this complexity to $O(n \log n)$, making it vastly more efficient for large datasets. Importantly, the Fourier Transform is a bijection, meaning it is invertible, given a transformed signal $f' = \text{fft}(f)$, the original signal can be perfectly reconstructed using the inverse FFT: $f = \text{inv\_fft}(f')$ [9].

Number Theoretic Transform (NTT) is a specialized variant of DFT that operates over finite fields, typically Z_q, where q is a carefully chosen prime, rather than complex numbers. It enables fast polynomial multiplication modulo a prime number, which is essential in many lattice-based cryptographic schemes. FALCON uses NTT to accelerate public key operations and key pair generation, leveraging arithmetic over the finite ring Z_q [16]. However, private key operations, including signature generation, rely on Fast Fourier Sampling (FFS) over complex numbers, and thus depend on FFT [16].

SPHINCS+ diverges from this algebraic paradigm by relying solely on the hardness of cryptographic hash functions, such as SHA-256 or SHAKE256. It uses a hypertree of hash-based structure of WOTS+ and FORS, which are combined in an XMSS tree to construct a large but stateless signature scheme. We previously explored the foundations of SPHINCS+ in Sections 5 & 6.

7.2 Size Comparison

The following three tables provides a detailed comparison of classical and post-quantum digital signature schemes, categorized by their public and private key sizes, signature sizes, and targeted security levels. This comparative analysis serves to highlight the trade-offs between efficiency and security across different cryptographic foundations, including lattice-based, hash-based, multivariate, and symmetric cryptographic systems.

Table 5: Comparison of Classical Signature Schemes by Key and Signature Size in Bytes

Method	**Public Key**	**Private Key**	**Signature**	**Type of Scheme**	**Security Level**
RSA-2048	256	256	256	Integer	<1 (128-bit)
ECC 256-bit	64	32	64	Elliptic Curve	<1 (128-bit)

In Table 5, RSA-2048 and ECC-256 represent the baseline for 128-bit classical security [43]. RSA features relatively large key and signature sizes (256 bytes), whereas ECC achieves the same level of security with much smaller keys and signature size. ECC's compact nature makes it attractive for constrained environments, but neither RSA nor ECC is considered secure in a post-quantum setting.

Table 6: Comparison of Rejected PQC Schemes by Key and Signature Size in Bytes

Method	**Public Key**	**Private Key**	**Signature**	**Type of Scheme**	**Security Level**
Picnic 3 Full	49	73	71,179	Symmetric	3 (192-bit)
GeMSS 128	352,188	16	33	Multivariate	1 (128-bit)
GeMSS 192	1,237,964	24	53	Multivariate	3 (192-bit)
Rainbow Level Ia	161,600	103,648	66	UOV	1 (128-bit)
Rainbow Level IIIa	861,400	611,300	164	UOV	3 (192-bit)
Rainbow Level Vc	1,885,400	1,375,700	204	UOV	5 (256-bit)

Looking at Table 6, Rainbow signatures, which is based on the based on the Unbalanced Oil and Vinegar scheme invented by Jacques Patarin, was a multivariate signature scheme, where its security relied on the hardness of solving a large system of multivariate quadratic equations over a finite field [11]. Rainbow offered an exceptionally small signature size however, this scheme suffered an enormous key size, with public keys ranging from 161 KB to over 1.8 MB depending on the security level. This key size overhead presented a significant practical limitation for real-world deployment, but despite that NIST still moved it to Round 3 of its search. However, in 2020, Ward Beullens discovered two attacks that ultimately broke Rainbow, eliminating it from Round 3 [51]. Alongside Rainbow, Picnic and GeMSS were also ruled out. NIST stated that Picnic was not mature enough to be standardized, and GeMSS had a much larger public key and slower signing time compared to others in Round 3 [29].

Table 7: Comparison of Standardized PQC Schemes by Key and Signature Size in Bytes

Method	Public Key	Private Key	Signature	Type of Scheme	Security Level
ML-DSA-44	1,312	2,560	2,420	Lattice	2 (128-bit)
ML-DSA-65	1,952	4,032	3,309	Lattice	3 (192-bit)
ML-DSA-87	2,592	4,896	4627	Lattice	5 (256-bit)
FALCON-512	897	1,281	690	Lattice	1 (128-bit)
FALCON-1024	1,793	2,305	1,330	Lattice	5 (256-bit)
SLH-DSA-SHA2-128f Simple	32	64	17,088	Hash-based	1 (128-bit)
SLH-DSA-SHA2-128s Robust	32	64	7,856	Hash-based	1 (128-bit)
SLH-DSA-SHA2-192f Simple	48	96	35,664	Hash-based	3 (192-bit)
SLH-DSA-SHA2-192s Robust	48	96	16,224	Hash-based	3 (192-bit)
SLH-DSA-SHA2-256f Simple	64	128	49,856	Hash-based	5 (256-bit)
SLH-DSA-SHA2-256s Robust	64	128	29,792	Hash-based	5 (256-bit)

Looking at Table 7, FALCON offers compact signatures, as small as 690 bytes for Level 1 security, while ML-DSA exhibits slightly larger signature and key sizes. At higher security levels, both schemes exhibit modest growth in size while maintaining efficiency and strong resistance against quantum attacks. While SPHINCS+ key sizes are minimal, ranging from 32 to 64 bytes, SPHINCS+ still suffers from large signature sizes, exceeding 17 KB at Level 1 and growing to nearly 50 KB at Level 5 for the fast variant. The small variants offer reduced signature sizes but at the expense of slower performance, which can be seen later in Table 8. Despite this, SPHINCS+ remains a strong candidate for applications that prioritize long-term security and can tolerate slower signing times.

7.3 Setting up the Environment

To compile and run the benchmarking suite, detailed in Appendix B, a properly configured build environment is required. The Appendix section outlines the necessary tools, libraries, and installation steps to successfully build the project.

7.4 Performance Results

Table 8 presents the benchmark results that we discovered for a selection of digital signature schemes, including post-quantum algorithms, SPHINCS + Robust, SPHINCS + Simple, FALCON and ML-DSA, as well as the classical ECDSA algorithm. The benchmarks include the average of key generation, signing, verifying times, measured in microseconds (ms), along with the average CPU time spent signing and verifying over the course of 1000 iterations on a message size of 1024-bytes.

The results highlight significant performance variation across post-quantum and classical digital signature schemes when signing 1024-byte messages. As expected, SPHINCS+ Robust exhibits the highest signing time, averaging over 201 milliseconds, reflecting the computational cost of its stateless hash-based construction. Even its verification time, though under one millisecond, remains higher than other schemes, at 0.231 ms.

In contrast, FALCON demonstrates exceptional performance, with average signing and verification times of 0.175 ms and 0.039 ms, respectively. Its verification time significantly outpaces both SPHINCS+ variants and even classical ECDSA. This makes FALCON a strong candidate for real-time applications that demand minimal latency.

The fast variant, SPHINCS+ Simple, reduces average sign time to approximately 9.78 ms and verification time to 0.60 ms, offering a trade-off between robustness and efficiency. Meanwhile, ML-DSA shows promising sub-millisecond performance for both signing and verification, achieving 0.20 ms and 0.034 ms, respectively.

Table 8: Performance results for digital signature schemes on 1024-byte messages in ms

Scheme	**Key Gen**	**Sign**	**Verify**	**Sign CPU**	**Verify CPU**
ECDSA	0.068269	0.077516	0.105865	7.7e-05	0.000105
SPHINCS+ 128 Robust	13.4258	201.734	0.231223	0.201229	0.00023
SPHINCS+ 128 Simple	0.43128	9.78411	0.600149	0.009724	0.000596
FALCON-512	5.53196	0.17545	0.038626	0.000174	3.8e-05
ML-DSA-44	0.098982	0.199791	0.033798	0.000198	3.3e-05

To evaluate how each signature scheme scales with increasing message sizes, we benchmark key pair generation, signing, and verification times for various cryptographic schemes across multiple message lengths in bytes, $[4, 32, 64, 128, 512, 1024, 2500, 4096]$. These include classical (ECDSA), lattice-based (FALCON), hash-based (SPHINCS+ Simple and Robust variants), and a hybrid scheme combining ECDSA with SPHINCS+ Robust. The tests were conducted using a uniform benchmarking framework described in Appendix B. Key generation times, shown in Figure 8 remained largely constant across

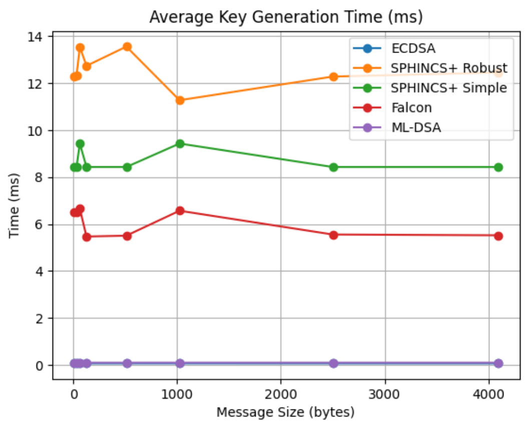
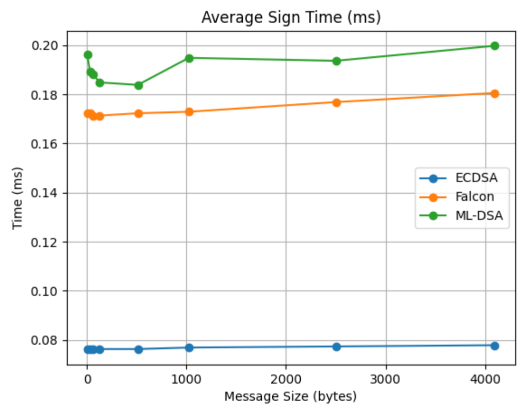


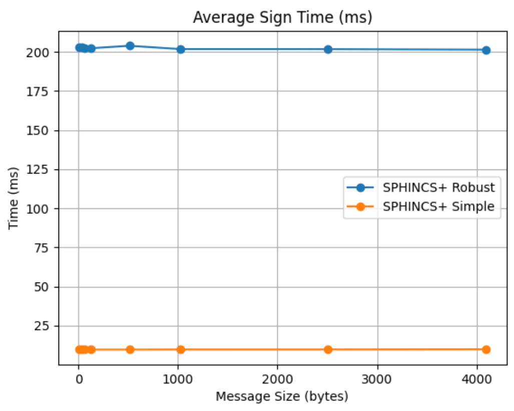
Figure 8: Key Generation

all message sizes, as expected. ML-DSA and ECDSA consistently produced keys in under 1ms, their lines being almost overlapping in the figure. On the other end, FALCON and SPHINCS+ Simple were consistently just under 10ms. SPHINCS+ Robust exhibited significantly higher key generation times, averaging over 13 milliseconds.

Signing time, shown in Figure 9b, revealed the greatest divergence in performance. ECDSA, FALCON, and ML-DSA maintained extremely low and stable signing latencies across all message sizes, typically remaining under 1 millisecond. In contrast, SPHINCS+ signing times were substantially higher, especially for the Robust variant, which exceeded 200 milliseconds on average, with marginal increases as message size grew. For SPHINCS+ Simple, the average was around 10ms, again, with marginal increases as message size grew. For SPHINCS+, especially the Robust variant, the high signing overhead stems from its reliance on stateless hash-based signatures, specifically, hypertree traversal and FORS structures, which must be recomputed for each message. We believe this will cause SPHINCS+ unsuitable for real-time applications demanding high signing throughput.



(a) Signing Time for NIST Finalists and ECDSA



(b) Signing Time for SPHINCS+ variants

Figure 9: Comparison of signing times in ms

Verification times, shown in Table 10, were generally more efficient and less sensitive to message size. ML-DSA and FALCON demonstrated the fastest verification times, averaging under 0.05 milliseconds. SPHINCS+ variants required slightly more time, approximately 0.2 to 0.6 milliseconds, but remained relatively consistent across all message sizes. ECDSA also maintained low verification latency, only slightly slower than ML-DSA and FALCON.

7.5 Analysis Conclusion

This analysis demonstrates that while all three NIST selected signature schemes meet post-quantum security goals, they present sharply different trade-offs. Lattice-based

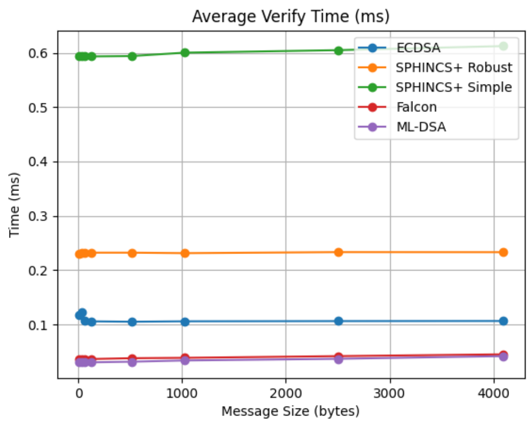


Figure 10: Verify Time

schemes such as ML-DSA and FALCON offer compelling performance characteristics. FALCON, in particular, combines compact signatures, rapid verification, and strong security proofs. ML-DSA, though slightly larger in key and signature size, excels in implementation simplicity by avoiding floating-point arithmetic, and its performance remains fast, which is ideal for many applications.

In contrast, SPHINCS+ represents a fundamentally different design philosophy. By avoiding algebraic assumptions entirely and relying only on the security of hash functions, SPHINCS+ maximizes cryptographic conservatism. This makes it well-suited for use cases demanding long-term security guarantees or resistance to yet-unknown cryptanalytic advances, however, the costs are steep. SPHINCS+ suffers from large signature sizes and substantially higher signing times, particularly in the Robust variant.

Despite improvements in the simple variant of SPHINCS+, performance remains an obstacle for latency-sensitive applications. The results suggest that SPHINCS+ is best reserved for niche deployments where efficiency can be sacrificed for assurance.

These findings suggest that while SPHINCS+ is suitable as a conservative fallback, its practical limitations may prevent it from serving as a primary signature scheme in most applications. Should a breakthrough in lattice cryptanalysis occur, the cryptographic community would urgently need a backup signature scheme rooted in non-algebraic primitives. At present, SPHINCS+ serves as the only standardized non-algebraic alternative. However, as demonstrated in this analysis, its performance characteristics present serious scalability challenges. These concerns underscore the need for continued research and standardization efforts to identify or develop additional digital signature schemes based on non-lattice assumptions.

8 Development and Evaluation Hybrid Scheme

As the cryptographic community prepares for the eventual threat of large-scale quantum computers, hybrid signature schemes have emerged as a transitional solution to bridge the gap between classical and post-quantum security. These schemes combine a well-established classical digital signature algorithm with a post-quantum counterpart, providing a layered defense [8]. This section details the implementation and evaluation of such a hybrid signature scheme, constructed by pairing traditional and quantum-resistant algorithms in a dual-signature format.

A hybrid signature scheme operates by employing two distinct cryptographic algorithms in tandem. First, the signer generates both a classical key pair and a post-quantum key pair. When a message needs to be signed, the signer uses both private keys to generate two independent signatures over the same message. These are then concatenated into a single hybrid signature.

On the verification side, the recipient uses the corresponding public keys to verify both parts of the hybrid signature. Verification succeeds only if both signatures are independently valid. This model ensures that the message's authenticity is preserved even if one of the signature algorithms is broken, provided the other remains secure.

To support this structure in real-world applications, both public keys are typically bundled together, such as through X.509 certificate extensions in TLS. While this approach will enhance security assurance, it does come with some drawbacks. Notably, hybrid signatures, since they combine two schemes, will be significantly larger in size and, as a result, will incur greater computational costs for both signing and verification. Despite these challenges, the enhanced security assurance it brings is worthwhile, especially for environments where a long-term security guarantee is a concern.

8.1 Hybrid Scheme Development

1. Key Generation Let:

- (sk_1, pk_1) be the private, public key pair for the classical signature scheme.
- (sk_2, pk_2) be the private, public key pair for the post-quantum signature scheme.

These key pairs are generated independently:

$$\begin{aligned} sk_1 &\leftarrow \text{KeyGen}_{\text{classical}}(), \qquad & pk_1 &= \text{derive}(sk_1), \\ sk_2 &\leftarrow \text{KeyGen}_{\text{pqc}}(), \qquad & pk_2 &= \text{derive}(sk_2). \end{aligned}$$

2. Signing To sign a message m, compute:

$$\begin{aligned} \sigma_1 &= \text{Sign}_{\text{classical}}(sk_1, m), \\ \sigma_2 &= \text{Sign}_{\text{pqc}}(sk_2, m). \end{aligned}$$

The resulting hybrid signature is the tuple:

$$\sigma = (\sigma_1, \sigma_2).$$

3. Verification To verify a hybrid signature $\sigma = (\sigma_1, \sigma_2)$ on message m, perform:

$$\begin{aligned} \text{Verify}_{\text{classical}}(pk_1, m, \sigma_1) &\rightarrow \text{accept/reject}, \\ \text{Verify}_{\text{pqc}}(pk_2, m, \sigma_2) &\rightarrow \text{accept/reject}. \end{aligned}$$

The hybrid signature is accepted if and only if both components are valid:

$$\text{Valid} = \text{Verify}_{\text{classical}}(pk_1, m, \sigma_1) \wedge \text{Verify}_{\text{pqc}}(pk_2, m, \sigma_2).$$

8.2 Hybrid Scheme Secuirty

A core advantage of hybrid signature schemes is their resilience to partial failure. If one of the two signature algorithms used in the hybrid scheme, either the classical (e.g., ECDSA) or post-quantum (e.g., SPHINCS+) component, is later found to be insecure or faulty, the other algorithm can still provide a layer of protection [8]. In this model, the signature remains valid only if both individual signatures verify successfully; thus, if one fails due to cryptographic weakness or a fault, the entire hybrid signature is considered invalid.

This strict verification requirement provides strong security but also means that a fault in either scheme can cause operational disruptions. For instance, if a future attack breaks ECDSA, all hybrid signatures relying on ECDSA would fail verification unless systems are updated to ignore or replace the broken component. Therefore, the hybrid scheme's design must balance robustness against faults with deployment practicality. We believe that once a vulnerability is discovered in one component, software updates or policy changes would be required to migrate to a new hybrid pairing or rely solely on the still-secure component.

Regardless of its internal construction, a hybrid digital signature scheme must satisfy the standard security notion of existential unforgeability under chosen message attack (EUF-CMA). This requirement ensures that even when combining a classical and post-quantum algorithm, the overall scheme remains secure against adversaries attempting to forge signatures after observing chosen message queries. As discussed by Bindel [8], hybrid designs are intended to preserve security guarantees throughout the transition to post-quantum cryptography and must therefore adhere to the established EUF-CMA model to maintain trust in deployed infrastructures.

NIST has not released any official comments on when or if a hybrid scheme should be disallowed as they had in Table 3. However, it is our opinion that a hybrid scheme's quantum-vulnerable component determines the depreciation of the hybrid scheme. Once the classical signature scheme is broken by quantum advances, any hybrid signature that includes it becomes partially invalid from a trust standpoint, even if the PQC component remains cryptographically sound. In such cases, the signature may still be verifiable and secure under the PQC component, but it fails to meet the original dual-verification standard of the hybrid scheme as described in 8.1. Continuing to rely on the hybrid scheme after half is known to be broken undermines the layered security guarantees these schemes were intended to provide. Therefore, we argue that once the weaker component is deprecated or publicly broken, the hybrid format itself should no longer be used in critical infrastructure. Instead, systems should transition fully to the post-quantum scheme.

8.3 Hybrid Scheme Results

Integrating ECDSA with PQC schemes in a hybrid signature framework naturally results in an increase in overall size for public keys, private keys, and signatures as shown in Table 9. This is due to the concatenation of both classical and quantum-resistant components within the cryptographic artifacts. While such hybrid designs offer transitional security benefits against quantum adversaries, they also impose additional storage overhead, factors that must be carefully considered in constrained environments.

Table 9: Comparison of Hybrid PQC Schemes by Key and Signature Size in Bytes

Method	**Public Key**	**Private Key**	**Signature**	**Type of Scheme**	**Security Level**
ML-DSA-44	1,376	2,592	2,484	Lattice	2 (128-bit)
ML-DSA-65	2,016	4,064	3,373	Lattice	3 (192-bit)
ML-DSA-87	2,656	4,928	4,691	Lattice	5 (256-bit)
FALCON-512	961	1,313	754	Lattice	1 (128-bit)
FALCON 1024	1,857	2,337	1,394	Lattice	5 (256-bit)
SLH-DSA-SHA2-128f Simple	96	96	17,152	Hash-based	1 (128-bit)
SLH-DSA-SHA2-128s Robust	96	96	7,920	Hash-based	1 (128-bit)
SLH-DSA-SHA2-192f Simple	112	128	35,728	Hash-based	3 (192-bit)
SLH-DSA-SHA2-192s Robust	112	128	16,288	Hash-based	3 (192-bit)
SLH-DSA-SHA2-256f Simple	128	160	49,920	Hash-based	5 (256-bit)
SLH-DSA-SHA2-256s Robust	128	160	29,856	Hash-based	5 (256-bit)

The performance results for the hybrid schemes, shown in Table 10, indicate that the combined ECDSA + PQC signatures introduce only a modest increase in runtime compared to the post-quantum schemes in isolation. This outcome is expected, as hybrid schemes execute both the classical and post-quantum signing and verification processes sequentially. This small overhead confirms that the computational cost of hybridization is largely additive, but not prohibitively so. It suggests that in contexts transitional security is necessary, hybrid signatures can be feasibly deployed with only a slight increase in processing time over the standalone PQC scheme. However, the concern remains that SPHINCS+ by itself is already much larger with a costly performance, so even a few ms slower can make it increasingly impractical in constrained environments.

Table 10: Performance results for Hybrid PQC Schemes on 1024-byte messages in ms

Scheme	**Key Gen**	**Sign**	**Verify**
ECDSA + SPHINCS+ 128 Robust	16.256	205.919	0.371659
ECDSA + SPHINCS+ 128 Simple	0.607468	10.79311	0.734677
ECDSA + FALCON-512	6.35757	0.285816	0.18719
ECDSA + ML-DSA-44	0.17314	0.354733	0.235762

8.4 V2V and Post-Quantum Cryptography Integration

The increasing deployment of V2V communication systems presents a critical need for secure authentication mechanisms to prevent spoofing and message tampering. Current V2V standards, such as IEEE 1609.2, rely on the ECDSA to sign safety messages [22]. However, with the anticipated advancement of QC and the expected disallowance of ECDSA by 2035, this places the lifetime of vehicles being manufactured today at risk. This necessitates the integration of PQC into V2V authentication frameworks while maintaining stringent bandwidth and latency requirements.

Unlike traditional security updates, the integration of PQC into V2V communications poses unique challenges due to the following:

1. Stringent Bandwidth Constraints: Payloads must fit within the 2,304-byte limit under the Dedicated Short-Range Communication (DSRC) protocol, making it difficult to incorporate PQC, which as we saw in Section 7, involves much larger key sizes and signatures [54].

2. High Message Frequency: Vehicles broadcast safety messages every 100ms, requiring authentication mechanisms that can operate within milliseconds to avoid delays [54].

3. Backward compatibility: Newly developed quantum-resistant vehicles must still interoperate with older vehicles that rely solely on ECDSA-based authentication [54]. For these vehicles with ECDSA, a manufacturer-issued recall will be necessary to apply protocol updates to prevent security risks.

To address these constraints, a study introduced a Partially Hybrid Authentication Protocol, which provides a practical transition toward post-quantum security without disrupting existing V2V infrastructure [54]. The key ideas involve

1. Continue using ECDSA to sign individual safety messages (BSMs), ensuring compatibility with current systems.

2. Protect the ECDSA public key using a hybrid certificate, where the pseudonym certificate is signed using both the ECDSA and a PQ signature, FALCON, or ML-DSA. This prevents future quantum attacks on the key itself.

3. Optimize certificate transmissions using an AI-driven scheduling mechanism to reduce redundant certificate broadcasts, freeing up spectrum for PQC integration.

This partial hybrid approach ensures that even if a quantum computer becomes capable of breaking ECDSA, the attacker cannot forge new valid certificates, as the PQC signature remains secure. The authors had projected this to be used until 2035 with a transition to fully hybrid right after and lasting until 2042. However, as NIST published disallowance by 2025, shown in Table 3, we argue that full hybrid deployment should be adopted much sooner to proactively mitigate risk and facilitate a smoother transition toward exclusive post-quantum cryptography by 2035.

The authors of this approach omitted SPHINCS+ Simple from their testing altogether and did not fully investigate SPHINCS+ Robust. We decided to analyze both of these schemes first in a pure PQC implementation and determine if this was the correct decision.

8.5 Setting up the environment

To ensure the consistency and reproducibility of the results, it was important to run all tests in the same environment as the paper mentioned earlier. An Ubuntu 22.04 virtual machine (VM) pre-configured to compile and run the artifact was archived in OVA format at doi:10.5281/zenodo.10160535. This virtual machine provided a clean, reproducible environment containing all necessary dependencies and configurations used during experimentation.

The virtual machine was created and tested using VirtualBox. For the setup on my machine some updates needed to be made to the Debian Kernel to allow VirtualBox to run, but afterwards, there was no further issues. More guidance can be found in Appendix C.

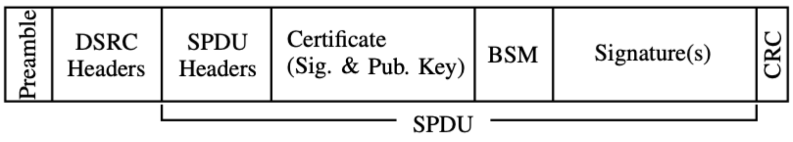


Figure 11: SPDU structure for DSRC Protocol
[**54**]

8.6 V2V Size Constraints

DSRC protocol as defined by the IEEE 1609.2 standard imposes a strict payload size limit of 2,304 bytes per frame, regardless of channel conditions or modulation schemes [22]. This frame limit significantly restricts the feasibility of integrating PQC schemes, such as those that have key and signature sizes that can easily exceed this cap. Before one can begin to evaluate a fully hybrid scheme, we must evaluate the practicality of a pure-PQC scheme.

Each BSM is digitally signed to ensure integrity and authenticity. This signed message, along with the relevant certificate and additional security metadata, is encapsulated within a Secure Protocol Data Unit (SPDU) as defined by IEEE 1609.2, shown in Figure 11 [22]. Once formed, the SPDU is embedded in a DSRC frame and broadcast to nearby vehicles.

Our design of the Secure Protocol Data Unit (SPDU) is guided by a set of assumptions rooted in both the IEEE 1609.2 specification and the practical limitations imposed by the vehicular communication system. Every BSM is assumed to be signed using a digital signature scheme compliant with IEEE 1609.2. This includes not only the message payload but also relevant metadata such as timestamps and location data [54]. Each SPDU contains the sender's certificate, which holds the signature and public key.

Figure 12 presents a comparison of total SPDU frame sizes using various digital signature schemes, including the currently used classical ECDSA and PQC options. Each bar represents a stacked frame composed of three components: the public key, the signature, and the signed BSM plus any protocol overhead which we estimated to be 100 bytes to error on the side of caution. A red dashed line indicates the strict 2,304-byte frame size limit imposed by the DSRC protocol. The results we present below highlight a significant challenge for PQC integration into DSRC-based vehicular networks.

ECDSA signature scheme results in a total SPDU size well within the DSRC limit, as expected. It remains fully compatible with current DSRC specifications and is ideal in this pre-quantum computing era.

The frame size for ML-DSA-based SPDUs exceeds the DSRC limit, primarily due to larger public key and signature sizes. Although more efficient than SPHINCS+, ML-DSA remains infeasible under existing DSRC constraints. FALCON is the most promising post-quantum candidate in terms of compactness. The total SPDU only slightly exceeds the 2,304-byte limit. While not deployable under the current strict DSRC protocol, there is hope that adjustments could be made to the infrastructure by 2035 to accommodate FALCON as a viable PQC choice in V2V systems. Both variants of SPHINCS+ result in prohibitively large frame sizes. SPHINCS+ Simple, in particular, approaches 35,000 bytes, approximately 15 times the DSRC frame limit. These sizes render SPHINCS+ unsuitable for broadcast safety messaging under the current DSRC infrastructure. While there is time for adjustments to be made to the infrastructure before the projected disallowance of classical algorithms by 2035, we believe it is unlikely that SPHINCS+ will

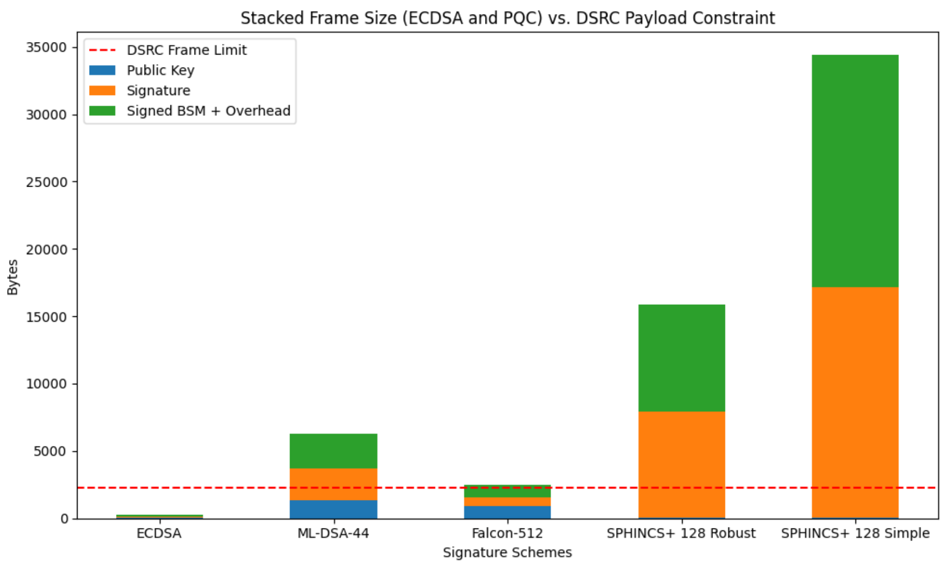


Figure 12: SPDU Packet Size

become a practical candidate for V2V authentication. Its size is fundamentally mismatched with the constrained requirements of the DSRC protocol.

To evaluate a hybrid scheme combining SPHINCS+ and ECDSA, it is important to consider the cumulative size impact. Hybrid signatures as stated earlier in section 8.1, require both the SPHINCS+ and ECDSA signatures to be transmitted together, along with their respective public keys and associated metadata. This would only increase the total frame size further, compounding the already prohibitive size of SPHINCS+ alone. Therefore, while hybrid schemes offer the benefit of transitional security, the use of one using SPHINCS+ is simply impractical. Among the considered candidates, FALCON remains the most promising due to its relatively compact key and signature sizes. Although a FALCON+ECDSA hybrid would still exceed the DSRC frame size limit, a partially hybrid approach, such as the asymmetric validation strategy proposed by Twardokus [54], presents a compelling direction for further investigation. We at least hope this would motivate the process for increasing the DSRC limit sooner.

8.7 V2V Results

Table 11 presents the verification times and throughput capabilities of various digital signature schemes within the context of V2V. In high-density traffic environments, it is estimated that up to 100 vehicles may be transmitting BSMs every 1 ms. Using this info is how we calculated a v_{max} value in the following results. The table shows that traditional ECDSA remains the fastest, with an average verification time of just 0.000157 milliseconds, allowing over 600,000 verifications. While this meets current V2V demands, ECDSA is known to be vulnerable to future quantum attacks.

Among the PQC candidates evaluated, ML-DSA and FALCON both offer excellent performance under these constraints. FALCON, in particular, demonstrates an average verification time of 0.0508 milliseconds, translating to nearly 2,000 verifications which is well within the acceptable limits for real-time vehicular communication. These figures suggest that schemes like FALCON and ML-DSA could serve as a practical post-quantum replacement without requiring specialized hardware acceleration.

In contrast, SPHINCS+ variants exhibit significantly higher verification times. The Robust variant manages 0.226 milliseconds per verification, supporting roughly 431 verifications. The simple variant performs even slower, achieving just 187 verifications. Although both variants technically meet the minimum requirement for V2V authentication, they do so only marginally on standard computing platforms without specialized V2X co-processors or dedicated cryptographic accelerators. This narrow margin raises concerns about scalability in dense traffic environments or under high system load.

Table 11: V2V Our Results in ms

Scheme	**Avg. Verify**	$v_{\max}$	**Acceptable?**
ECDSA	0.000157	636942	Yes
ML-DSA-44	0.030355	3294	Yes
FALCON-512	0.050803	1968	Yes
SPHINCS+ 128 Simple	0.534464	187	Yes*
SPHINCS+ 128 Robust	0.226387	431	Yes*

While SPHINCS+* was shown to meet the minimum verification requirements on a standard computing setup, its performance left little room for scalability or error, especially in high-density traffic scenarios. To explore how hardware acceleration might improve the viability of post-quantum signatures in V2V systems, the referenced paper conducted experiments using the CODHA (Cooperative Driving Hardware Accelerator) platform, a specialized V2X device designed to support secure, low-latency communications.

Table 12 [1] summarizes the results obtained in this hardware-accelerated environment. As expected, ECDSA verification remains extremely efficient, sustaining over 67,000 verifications, an order of magnitude slower than on desktop-class systems, but still well above the V2V threshold. For post-quantum schemes, ML-DSA and FALCON maintain acceptable performance with 529 and 224 verifications, respectively. These figures confirm that such lattice-based signatures remain viable on embedded V2X platforms and could be suitable for future integration into V2V authentication protocols.

The SPHINCS+ Robust variant exhibits a verification time of over 5 milliseconds, reducing throughput to only 18 verifications, far below the operational requirement for V2V systems. The Simple variant was not evaluated in this context, which is why we thought to add it to our own evaluation. However, after seeing its significantly lower performance in the standard environment, we do not believe it would perform any better than SPHINCS+ Robust.

Table 12: V2V Paper Results in ms

Scheme	**Avg. Verify**	$v_{\max}$	**Acceptable?**
ECDSA	0.001	67521	Yes
ML-DSA-44	0.189	529	Yes
FALCON-512	0.446	224	Yes
SPHINCS+ 128 Simple	-	-	-
SPHINCS+ 128 Robust	5.436	18	No

[1] The authors did not test SPHINCS+ Simple, so it is omitted here. We believe this decision came from the size constraints we discussed in Figure 12.

8.8 Objective Two Conclusion

The exploration of hybrid signature schemes reinforces their role as a critical transitional mechanism in the post-quantum migration. By pairing quantum-vulnerable algorithms such as ECDSA with quantum-resistant alternatives like SPHINCS+, these schemes provide layered security assurances. However, this added protection comes at a significant cost, particularly in signature size. SPHINCS+, even on its own, produces excessively large keys and signatures that violate existing V2V message frame constraints. When combined in a hybrid scheme, the cumulative size only adds to this violation of the frame size constraint for DSRC-based systems.

Our evaluation shows that both variants of SPHINCS+ dramatically exceed the 2,304-byte limit mandated by the IEEE 1609.2 standard, with SPHINCS+ Simple reaching almost 35,000 bytes. While SPHINCS+ meets basic throughput thresholds in verification tests, it does so only marginally in standard environments and outright fails under hardware-constrained V2X settings. These results confirm the decision by prior studies to exclude SPHINCS+ from V2V evaluations.

Therefore, although SPHINCS+ remains a valuable conservative fallback in scenarios where size and speed are secondary to cryptographic diversity, it is not suitable for deployment in real-time vehicular communications. This necessitates the identification and development of a non-lattice-based digital signature scheme as a viable backup to lattice schemes like FALCON and ML-DSA.

9 Applicability of SPHINCS+ in Blockchain

This section evaluates the applicability of SPHINCS+ in blockchain systems, focusing on its integration challenges and performance implications. It begins by explaining core blockchain and Bitcoin transaction concepts, including digital signatures, Proof of Work, and fee calculation. It then assesses the threat quantum computers pose to ECDSA-based signatures and outlines a proposed framework for replacing them with post-quantum schemes like SPHINCS+. Finally, the section presents a size and time analysis showing that SPHINCS+ dramatically increases transaction and block sizes and significantly slows down signing and verification, making it impractical for blockchain systems.

9.1 Blockchain

A blockchain is a decentralized and tamper-resistant digital ledger designed to record transactions across a distributed network of computers securely [31]. Instead of relying on a central authority, the blockchain operates on a peer-to-peer (P2P) network where each participant maintains a synchronized copy of the ledger. Transactions are grouped into blocks, each containing a cryptographic hash of the previous block, forming a chronological chain. This structure ensures data integrity and immutability, as once the data is recorded, it cannot be altered without consensus from the entire network [31].

At the core of blockchain is the use of cryptographic primitives such as hashing and digital signatures. These tools ensure the authenticity and integrity of transactions. A consensus mechanism, such as Proof of Work (PoW) or Proof of Stake (PoS), is employed to achieve agreement among distributed nodes on the validity of new blocks [2].

A PoW is a consensus mechanism that ensures only valid transactions are added to the blockchain. It works by requiring participants to solve a computationally difficult puzzle [2]. Before a participant begins solving the puzzle, they verify the digital signature to ensure validity, prevent any invalid blocks and avoid wasting their time. A puzzle involves finding a number called a nonce, a number used only once. When combined with the block header and hashed using SHA-256 twice, the puzzle will produce a hash below a specified difficulty target. This is known as a consensus being reached.

Once consensus is reached, the new block is appended to the chain and propagated across the network. This process guarantees transparency, auditability, and resilience against fraud or unauthorized tampering. Blockchains have seen widespread adoption in areas such as cryptocurrencies, supply chain tracking, decentralized finance, and secure digital identity systems [2].

Cryptocurrencies, such as Bitcoin, are among the most prominent and pioneering applications of blockchain technology. In these systems, the blockchain serves as a distributed ledger that records all monetary transactions in a transparent and verifiable manner. Each transaction represents the transfer of digital currency between users and is digitally signed using the sender's private key to ensure authenticity and non-repudiation.

To prevent issues such as double-spending, where the same digital currency could be fraudulently used more than once, cryptocurrencies employ blockchain's block-based recording mechanism alongside a consensus protocol like PoW defined above [2]. This chaining of blocks ensures that altering any transaction would require redoing the PoW for all subsequent blocks, making tampering computationally impractical. Each block thereby acts as a permanent record of transaction history, with newer blocks reinforcing the integrity of the earlier ones. In essence, cryptocurrencies leverage the immutability, transparency, and decentralized consensus properties of blockchain to establish a trustless digital payment system that allows value to be securely exchanged without the need for centralized financial institutions [31].

9.2 Bitcoin Transaction Overview

Transactions are the fundamental units of value transfer in blockchain systems [31]. In the Bitcoin protocol, a transaction consumes one or more unspent transaction outputs (UTXOs) and creates new UTXOs that can be spent in future transactions. This model forms the basis for Bitcoin's state management and security.

The life-cycle of a transaction begins when a user constructs a message indicating the intent to transfer value. This message includes inputs, references to UTXOs, and outputs, new locking scripts specifying conditions for future redemption. The user signs this message with their private key, generating a digital signature that proves ownership and authorization without revealing the private key itself. The transaction is then broadcast to the network [2].

Once broadcast, the transaction is picked up by participating nodes in the network, which verify its authenticity and validity. This includes ensuring that the referenced UTXOs are unspent, the digital signature is valid, and the transaction adheres to consensus rules [2]. One critical component of this verification process is checking that the public key used to authorize the transaction corresponds to the correct Bitcoin address. This is achieved using a compound hash function called `HASH160`, defined as:

$$HASH160(x) = RIPEMD160(SHA - 256(x)) \tag{6}$$

This function is applied to the public key to produce the public key hash, which is included in the locking script of the output, typically via the OP_HASH160 operation [2]. The use of RIPEMD160 after SHA-256 allows Bitcoin to balance cryptographic strength with shorter address size for efficiency.

In addition, when signing transactions, Bitcoin applies a double SHA-256 hash to the serialized transaction data:

$$Digest = SHA - 256(SHA - 256(tx)) \tag{7}$$

This double hashing is used to generate the message digest that is signed by the user's private key, enhancing security against certain types of cryptographic attacks [2].

Once validated, the transaction enters the memory pool, mempool, where it awaits inclusion in a block. A miner, in POW systems, or a validator, in POS systems, selects a group of pending transactions and constructs a candidate block. Upon solving the consensus puzzle, such as finding a valid nonce under PoW, the block is broadcast to the network and validated by other nodes [31]. If accepted, it is appended to the blockchain.

Each block references the previous one via a cryptographic hash, linking them in a secure and chronological chain [31]. Once a transaction is included in a confirmed block, it becomes a permanent and immutable part of the blockchain ledger.

9.2.1 Transaction Fees

Transaction fees are a critical component of Bitcoin's economic and security model. Although not explicitly defined as a separate field in the transaction structure, fees are inferred as the difference between the total input value and the total output value [2].

$$\text{Fee} = \sum \text{Inputs} - \sum \text{Outputs} \tag{8}$$

This implicit fee is collected by the miner who successfully mines the block containing the transaction. Transaction fees serve two primary purposes: (1) incentivizing miners to include the transaction in the next block, and (2) discouraging spam by attaching a cost to broadcasting transactions on the network [2].

Bitcoin transaction fees are calculated based on the transaction size, not on the amount of bitcoin being transferred. This is due to the bandwidth and storage costs imposed on the network. Larger transactions, such as those with many inputs, require higher fees to be considered by miners. A common unit of fee estimation is satoshis per byte [2], which estimates the fee that will give a transaction a high probability of being selected and included within a certain number of blocks [2].

Bitcoin's average transaction fee recently spiked to \$3.53, an increase of 176.8% from the previous day, May 14, 2025, and a 45.01% increase from last year [56]. This volatility highlights the sensitivity of Bitcoin's fee market to network congestion and limited block space. While this is manageable with compact and fast signature schemes like ECDSA, we believe transitioning to PQC will impose challenges.

9.3 QC Threat to Bitcoin

Bitcoin's reliance on the ECDSA digital signature algorithm introduces a critical vulnerability in the presence of quantum computers. Modern address formats such as Pay To Public Key Hash (P2PKH) and Pay to Witness Public Key Hash (P2WPKH) conceal the user's public key until a transaction is made; the act of spending coins exposes the full public key in the transaction data [2]. This moment of exposure creates a vulnerability window where an adversary could intercept the transaction, derive the private key using Shor's algorithm, and issue a conflicting transaction to steal the funds before the original transaction is confirmed in a block [4].

Bitcoin's older address type, Pay-to-Public-Key (P2PK), presents an even more immediate vulnerability to quantum attacks. In this format, the recipient's full public key is embedded directly in the address and visible on the blockchain at all times, even before any coins are spent [2]. This would mean that a sufficiently powerful quantum computer could, in principle, extract the corresponding private key using Shor's algorithm and steal the funds without waiting for a transaction to be broadcast. Many early Bitcoin holdings, including those believed to belong to Satoshi Nakamoto, remain stored in P2PK addresses, making them especially attractive targets in a post-quantum era [4].

Bitcoin transactions, once broadcast, are typically confirmed in about 10 minutes [53]. This delay stems from the time required for miners to solve a PoW puzzle and produce a new block. The confirmation is not instant like traditional fiat transactions but instead depends on block propagation and consensus among network participants [2]. However, with NIST declaring the disallowance of ECDSA by 2035 [30], it is imperative that Bitcoin begin transitioning to PQC to maintain its long-term security.

However, this transition will raise more concerns with the size constraints on each block. The bitcoin blocks size, though expanding from the 1MB size to a 4MB weight limit, will still have some issues from what we have seen previously with the PQC signature sizes shown in Section 7. The expansion in size stems from the segregated witness

[2]Named after the presumed person(s) who invented Bitcoin.

(SegWit) update, which began back in 2017 [12]. It separates witness data from the base block, aiming to scale the network and fix malleability bugs, resulting in a larger size. We examine the impact PQC signature schemes have on the blocks in Section 9.5.

9.4 Proposed Blockchain PQC Framework

The proposed post-quantum blockchain framework preserves the foundational architecture of classical blockchains, such as Bitcoin, while integrating quantum-resistant digital signature schemes. The process begins with key generation, where users create public-private key pairs using a selected PQC digital signature scheme, ensuring resilience against quantum adversaries leveraging algorithms like Shor's.

When a transaction is initiated, it would then be signed using the user's private PQC key. Miners would then aggregate signed transactions into candidate blocks during the block construction phase. Each transaction includes the PQC public key and its corresponding signature, both of which are necessary for subsequent validation. Network nodes verify these signatures as part of the consensus mechanism, ensuring transaction authenticity before the block is appended to the chain.

9.5 Calculating the Size

To assess the potential impact of adopting NIST post-quantum signature finalist algorithms in Bitcoin, we analyze blockchain data with a focus on transaction volume and block size. These metrics were derived from publicly accessible datasets provided through Bitcoin.org, as described in the sections below.

The two key datasets we used for this analysis:

- `avg-block-size-2.json`: Provides the average daily block size in megabytes.
- `n-transactions-per-block.json`: Reports the average number of transactions per block on each day.

To compute the average size of a transaction, we use the formula:

$$\text{avg\_tx\_size\_bytes} = \frac{(\text{block\_size\_MB} \times 10^6) - 80}{\text{transactions\_per\_block}}$$

where 80 bytes are subtracted to account for the fixed block header size.
We then simulate the effect of replacing the ECDSA public key and signature (128 bytes total) with post-quantum schemes by adjusting the average transaction size:

$$\text{pq\_tx\_size\_bytes} = \text{avg\_tx\_size\_bytes} - \text{ECDSA\_size} + \text{PQ\_sig\_size}$$

Using the new average transaction size, we recalculate the total block size:

$$\text{pq\_block\_size\_MB} = \frac{(\text{pq\_tx\_size\_bytes} \times \text{transactions\_per\_block}) + 80}{10^6}$$

We also estimate how many transactions would fit into a 1 MB block using:

$$\text{max\_txs\_in\_1MB} = \frac{10^6 - 80}{\text{pq\_tx\_size\_bytes}}$$

A summary of the most recent day's metrics, May 6, 2025, is presented in the three bar plots in Figure 13.

9.6 Size Analysis

Figure 13 illustrates the substantial differences in transaction and block sizes caused by post-quantum digital signature schemes, and their corresponding effect on blockchain scalability.

1. **Average Transaction Size (bytes)**: Highlights the signature's impact on transaction size.
2. **Projected Block Size (MB)**: Shows block size under constant transaction counts; includes a red line for the 4 MB SegWit cap.
3. **Transactions per 1 MB Block**: Reflects the throughput efficiency of each signature scheme.

Classical ECDSA signatures result in compact transactions of just 367 bytes. In contrast, post-quantum alternatives, especially hash-based schemes like SPHINCS+, significantly increase this footprint. For example:

- **SPHINCS+ 128f** inflates the transaction size to 17,359 bytes, over a 4700% increase.
- **SPHINCS+ 128s** produces 8,127-byte transactions, over 22 times the ECDSA size.
- **FALCON-512** and **ML-DSA** strike a middle ground at 1,826 and 3,971 bytes, respectively.

Larger transaction sizes directly result in larger block sizes to maintain today's standards, assuming constant transaction counts per block. The current constraints have a block size limit at 4MB [31]. Based on the second figure results on May 6th, 2025:

- **ECDSA** yields a block size of 1.56 MB, well within today's limits.
- **FALCON-512** expands this to 7.78 MB, almost double the current limit.
- **ML-DSA-44** increases this to 16.92MB, just over 4x the limit.
- **SPHINCS+ 128f** and **SPHINCS+ 128s** explode this to 73.96 MB and 81.27 MB respectively, making it impractical under current Bitcoin block size constraints.

A key bottleneck introduced by post-quantum schemes is the reduction in the number of transactions that can fit in a single block:

- **ECDSA:** 4,261 transactions per 1 MB block.
- **FALCON-512:** 855 transactions, 20% of ECDSA throughput.
- **ML-DSA-44:** 393 transactions, 9% of ECDSA throughput.
- **SPHINCS+ 128f:** only 90 transactions, a 95.9% drop.
- **SPHINCS+ 128s:** 192 transactions, a 95.5% drop.

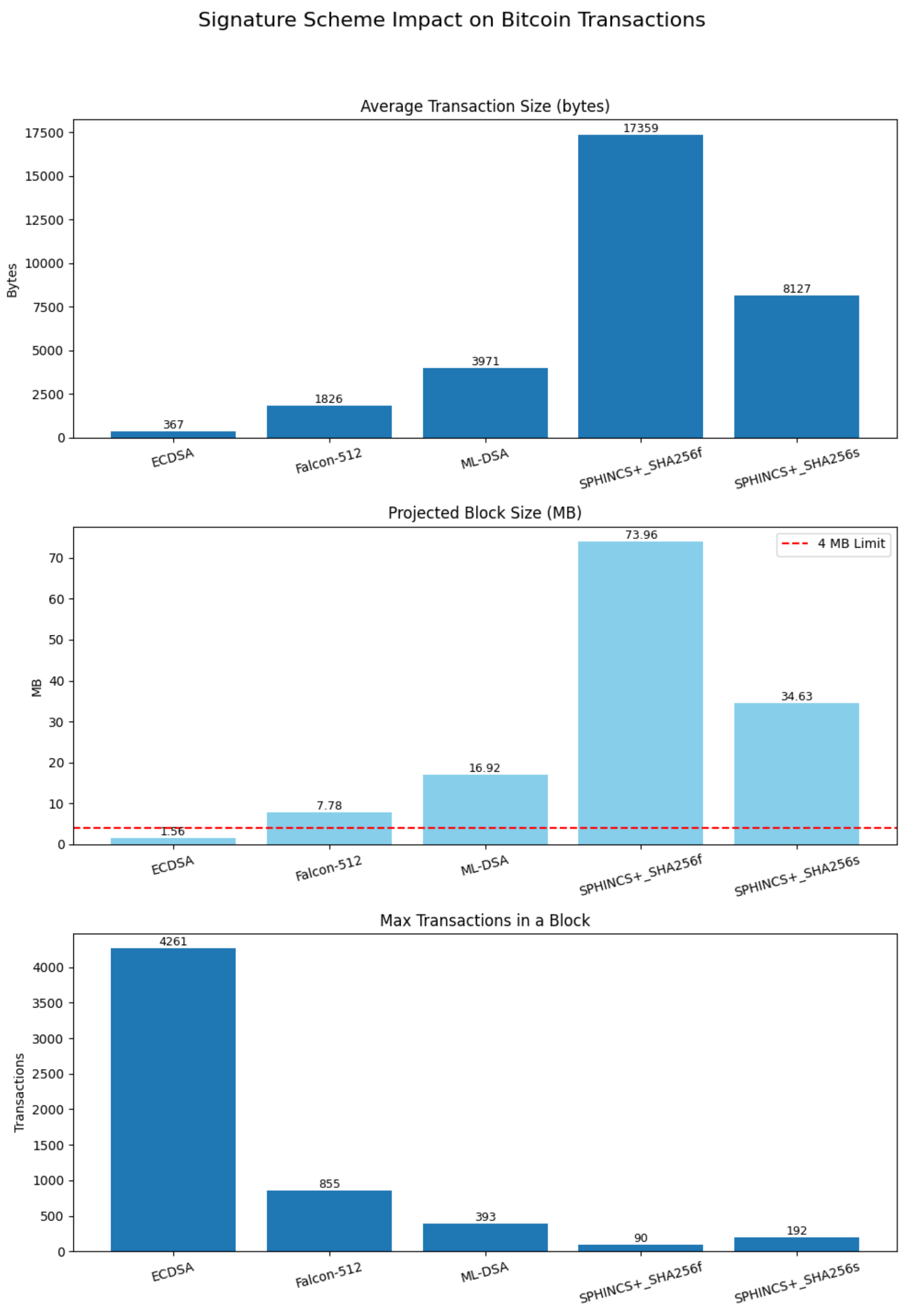


Figure 13: Signature Scheme Comparison

The dramatic increase in transaction sizes introduced by post-quantum signature schemes will have a direct consequence on transaction fees. As stated in Section 9.1.1, fees are calculated based on the size of the transaction, not the value it transfers. As a result, larger transactions will incur proportionally higher fees, since miners prioritize transactions offering higher satoshis per byte.

At a fee rate of 6 satoshis per byte, the estimated fees for various signature schemes are:

- **ECDSA** (367 bytes): 2,202 satoshis (0.000022 BTC) ($2.28)
- **FALCON-512** (1,826 bytes): 10,956 satoshis (0.00011 BTC) ($11.37)
- **ML-DSA-44** (3,971 bytes): 23,826 satoshis (0.00024 BTC) ($24.72)
- **SPHINCS+ 128s** (8,127 bytes): 48,762 satoshis (0.00049 BTC) ($50.59)
- **SPHINCS+ 128f** (17,359 bytes): 104,154 satoshis (0.001 BTC) ($108.05)

These inflated fees create strong economic disincentives for adoption under current conditions. Users and applications would be forced to pay more for each transaction, or risk being deprioritized during periods of congestion. Additionally, wallets and services managing many microtransactions or small-value payments would become economically infeasible with such signature schemes.

9.7 Mock Transaction Setup

To evaluate each digital signature scheme in a blockchain-relevant context, a mock transaction pipeline was implemented that mirrors the core steps in Bitcoin-style transaction signing. More information regarding the setup can be found in Appendix D.

The process begins with the generation of key pairs, followed by the derivation of the address using a HASH160-like function, RIPEMD-160(SHA-256(public key)), to simulate a blockchain-compatible output script. The transaction is then hashed using double SHA-256 to create a digest for signing. Each scheme signs this hash and then verifies the signature to confirm correctness. Throughout this process, the framework records timing metrics for key operations such as key generation, serialization, signing, and verification.

The framework benchmarks six signature schemes: ECDSA, using the Bitcoin-specific curve secp256k1, ML-DSA-44, FALCON-512, SPHINCS+ 128s, SPHINCS+ 128f, as well as three hybrid schemes: ECDSA and SPHINCS+ 128s, ECDSA and SPHINCS+ 128f, ECDSA and FALCON-512, ECDSA and ML-DSA-44. The hybrid configurations follow the setup described in Section 8.1.

Table 13 presents the average runtime (in milliseconds) for each component across six signature schemes. All results are averaged over 100 iterations. Additional implementation details, including hashing utilities, transaction structure, and encoding methods, are provided in Appendix D.

9.8 Time Analysis

The timing data in Table 13 reveals key differences in computational efficiency between classical and post-quantum digital signature schemes. Classical ECDSA remains the most efficient across all phases, with sub-millisecond performance in key generation, signing, and verification. FALCON demonstrates excellent performance, with fast signing and verification, though its key generation step is notably slower. ML-DSA, though a faster key generation step, is just slightly slower than FALCON, but remains close to today's ECDSA standards.

In contrast, SPHINCS+ exhibits significant computational overhead. SPHINCS+s requires over 200 milliseconds to sign a transaction, which is more than 1,800 times slower than ECDSA. Even its faster variant, SPHINCS+f, still takes nearly 10 milliseconds per signature.

Verification time is particularly important for blockchain scalability, as every full node must validate each transaction in a block. While SPHINCS+ Robust maintains a modest verification cost, 0.211 ms, the Simple variant requires 0.581 ms, over 3.8 times that of ECDSA.

The hybrid schemes, shown in Table 14, do inherit much of the overhead that comes with the PQC-specific scheme. This is particularly evident in the signing phase. While the standalone ECDSA operation is highly efficient, the SPHINCS+s hybrid takes over 200 ms to sign, compared to 0.647 ms for SPHINCS+f, 0.261 ms for the FALCON hybrid and 0.331 ms for the ML-DSA hybrid. Verification time also reflects the influence of the PQC algorithm, with SPHINCS+s hybrid requiring 0.36 ms, more than FALCON hybrid's time at 0.17 ms and only slightly worse than ML-DSA hybrid at 0.22 ms, but still better than SPHINCS+f hybrid, which was at .725 ms. When compared to the single schemes, these results show how the post-quantum half of the hybrid scheme will dominate performance.

While hybrid schemes provide transitional trust, since many existing blockchains rely on ECDSA and benefit from its well-established security, combining it with a post-quantum scheme introduces significant overhead. The increased signature size as well will lead directly to higher transaction fees, and the added verification time further strains blockchain scalability. As a result, despite offering quantum resilience, hybrid schemes are not a practical long-term solution for blockchain systems. We therefore do not recommend their use in high-throughput or resource-constrained environments.

Table 13: Timing Comparison (ms) for Signature Schemes

Step	**ECDSA**	**ML-DSA**	**FALCON**	**SPHINCS+s**	**SPHINCS+f**
Key Pair	0.079	0.102	5.775	14.638	0.427
Signing	0.108	0.262	0.169	202.362	9.644
TX Serialization	0.012	0.097	0.057	0.259	0.529
Verification	0.151	0.087	0.028	0.211	0.581

Table 14: Timing Comparison (ms) for Hybrid Signature Schemes

Step	**SPHINCS+s Hybrid**	**SPHINCS+f Hybrid**	**FALCON Hybrid**	**ML-DSA Hybrid**
Key Pair	17.347	0.647	7.422	0.178
Signing	205.352	10.663	0.261	0.331
TX Serialization	0.264	0.634	0.061	0.107
Verification	0.359	0.725	0.166	0.225

9.9 Objective Three Conclusion

While post-quantum digital signature schemes are vital for safeguarding blockchain systems against quantum threats, our findings strongly suggest that SPHINCS+ is not a feasible replacement for ECDSA in Bitcoin under present network conditions. The massive signature sizes lead to an exponential increase in transaction and block sizes, pushing them far beyond the 4 MB SegWit cap. As a consequence, the number of transactions that can fit within a block drops by over 95%, severely throttling throughput and dramatically increasing transaction fees. For instance, a single SPHINCS+ transaction could incur a fee exceeding $100 under typical fee-per-byte estimates, rendering micro-transactions and many applications economically infeasible.

SPHINCS+ performance and size limitations make it impractical for primary deployment in high-volume, fee-sensitive blockchain environments like Bitcoin. This highlights the urgent need for a more efficient, non-lattice-based backup post-quantum signature scheme, one that balances quantum resilience with reasonable signature sizes and computational costs. Until such an alternative emerges, schemes like FALCON-512, offer a more scalable and cost-effective path forward.

10 Feasibility of SPHINCS+

This section further evaluates the suitability of SPHINCS+ for Cellular Vehicle-to-Everything (C-V2X), as well as in more flexible protocols such as TLS 1.3, to determine where its strengths outweigh its limitations.

10.1 Feasibility Constraints in C-V2X Networks

While the limitations of SPHINCS+ have been discussed in the context of V2V systems, these challenges are magnified in C-V2X communication [54]. C-V2X imposes even stricter constraints on message size and timing due to its reliance on LTE and 5G-based physical layers. Unlike DSRC-based V2V, which offers more flexibility in frame size and scheduling, C-V2X must adhere to fixed transmission intervals and tighter latency budgets [54]. The maximum payload size in a standard 10 MHz channel is 437 bytes, less than five times the size supported in DSRC. In such environments, large signatures and public keys can lead to increased transmission delays and potential message drops. We've seen that SPHINCS+ produces extremely large signatures and involves high computational cost, making it an even less viable candidate for this integration. While FALCON or ML-DSA, depending on the size increase for DSRC, are better suited for deployment in real-time vehicular systems, in the C-V2X environment, all PQC schemes are infeasible and would require a complete overhaul of the design. This only confirms that SPHINCS+ is not practicable in all areas of vehicle communication.

10.2 Transport Layer Security (TLS) Overview

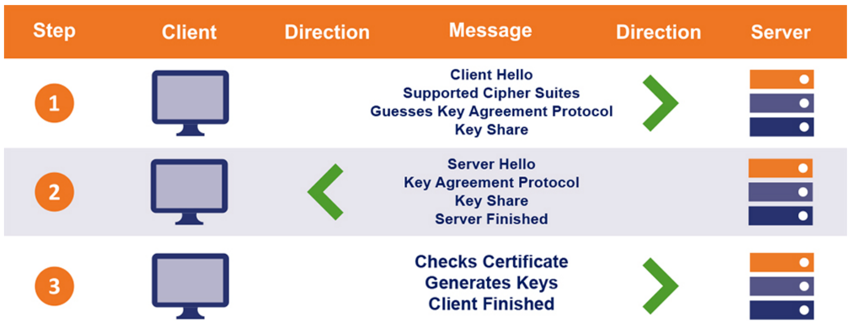
Transport Layer Security (TLS) is the standard protocol used to secure communication over the internet [25]. As the foundational security mechanism behind protocols like HTTPS, TLS ensures that data exchanged between clients and servers is confidential, authenticated, and tamper-resistant. TLS 1.3, specified in RFC 8446, is the latest version of the protocol and introduces significant improvements over its predecessors in terms of both security and efficiency. By removing legacy cryptographic algorithms and streamlining the handshake process, TLS 1.3 addresses many long-standing vulnerabilities in previous versions [41].

10.3 TLS Structure and Protocols

TLS 1.3 operates through two primary sub-protocols: the Handshake Protocol and the Record Protocol. The Handshake Protocol is responsible for negotiating cryptographic parameters, authenticating peers, and establishing shared secret keys. Once the handshake concludes, the Record Protocol takes over, encrypting and authenticating all subsequent application data. This division supports modularity and allows the protocol to be both extensible and secure [40]. Compared to earlier versions, TLS 1.3 encrypts more of the handshake itself to protect metadata and reduces the total number of round-trips required to establish a session [41].

10.4 TLS Security Improvements

TLS 1.3 significantly streamlined and hardened the protocol by removing support for many outdated cryptographic algorithms. RSA-based key exchange has been deprecated



Step	Client	Direction	Message	Direction	Server
1	🖥		Client Hello Supported Cipher Suites Guesses Key Agreement Protocol Key Share	❯	▤
2	🖥	❮	Server Hello Key Agreement Protocol Key Share Server Finished		▤
3	🖥		Checks Certificate Generates Keys Client Finished	❯	▤

Figure 14: TLS Handshake [52]

in TLS 1.3. Instead, the protocol mandates the use of forward-secure key exchanges like ephemeral Diffie-Hellman (DHE) and elliptic-curve variants (ECDHE) to ensure confidentiality even in the event of long-term key compromise [41]. However, as we know from earlier sections, RSA will be disallowed come 2035 and is not safe for use in the age of quantum computers.

10.5 TLS 1.3 Handshake Process

The TLS 1.3 handshake process is designed to establish a secure, authenticated communication channel between a client and a server. It occurs in three primary phases: key exchange, authentication, and handshake finalization. This process is shown in Figure 14 and described below.

In the key exchange phase, the client initiates communication by sending a ClientHello message, which includes a list of supported cryptographic algorithms and a key share for the key exchange part [52]. The server replies with a ServerHello, selecting compatible parameters and returning its own key share. This enables both parties to compute a shared secret.

Next, during authentication, the server presents a digital certificate that includes its public key. The server uses its private key to sign the handshake data, proving ownership of the certificate and thereby authenticating itself. In mutual authentication scenarios, the client may also present a certificate[52]. Finally, both client and server derive symmetric encryption keys from the shared secret and finish the handshake by exchanging Finished messages, which include message authentication codes to confirm that both parties have the same handshake transcript[41].

10.6 Adaptability for Post-Quantum Cryptography

As explained before in section 4, traditional public key cryptosystems such as RSA and Elliptic Curve Cryptography (ECC) rely on mathematical problems (e.g., integer factorization and discrete logarithms) that can be efficiently solved by quantum computers using Shor's algorithm. As a result, once sufficiently powerful quantum computers become available, these schemes will be rendered insecure. Integrating PQC into protocols like TLS is crucial to achieving *quantum resistance*, especially for the authentication step, which protects against impersonation and man-in-the-middle attacks. Furthermore, to mitigate the threat of "harvest-now, decrypt later."

10.7 Set Up the Environment

Since post-quantum cryptographic algorithms are not included in standard OpenSSL distributions, we compiled a custom version from source. More information regarding the testing scripts made can be found in Appendix E.

10.8 TLS Results

Table 15 presents a performance comparison of various classical, post-quantum, and hybrid digital signature algorithms used for server authentication during the TLS 1.3 handshake. Each row in the table reports the average handshake time and the corresponding certificate size for a given configuration, which was conducted over 100 iterations. The results reveal notable differences in efficiency across these cryptographic approaches, highlighting both the potential and the trade-offs of post-quantum adoption in real-time secure communication.

Among the post-quantum candidates, ML-DSA and FALCON show strong performance, achieving average handshake times of approximately 5.8 ms and 5.7 ms, respectively—both outperforming the classical RSA-3072, which averages 9.2 ms. FALCON also offers a favorable certificate size of just 2.4 KB, making it particularly suitable for deployment in latency-sensitive applications. In contrast, SPHINCS+ incurs a significant overhead with a handshake time of 38.9 ms and a large 22.9 KB certificate, reflecting its higher computational cost.

Hybrid signature schemes provide a bridge for the post-quantum transition, combining classical and quantum-safe methods. Configurations like RSA & ML-DSA and RSA & FALCON remain relatively efficient, with handshake times around 11 ms, only modestly higher than their individual components. However, the hybrid RSA & SPHINCS+ combination inherits the full cost of SPHINCS+, reaching 46.0 ms, suggesting limited benefit in combining it with RSA solely for the purpose of a transition phase to pure-PQC.

Table 15: TLS Handshake Results in ms

Algorithm	**Avg. Time**	**Certificate Size**
RSA-3072	9.2187	1.4 KB
ML-DSA-44	5.829	5.3 KB
FALCON-512	5.753	2.4 KB
SPHINCS+ 128f	38.892	22.9 KB
RSA-3072 & SPHINCS+ 128f	46.019	24.0 KB
RSA-3072 & ML-DSA-44	10.856	6.0 KB
RSA-3072 & FALCON	11.857	3.4 KB

10.9 TLS Extended

After completing TLS integration and verifying the secure handshake, we sought to further validate the correctness of the underlying connection when transmitting data after the handshake. To accomplish this, we implemented a lightweight HTTP test. The goal was to simulate typical post-TLS communication patterns, such as HTTP request/response exchanges, and ensure that data could be reliably transmitted and correctly interpreted. The results, summarized in Table 16, were compared with a second set of benchmarks that included both the handshake and the HTTP message exchange.

As expected, the inclusion of a minimal HTTP exchange introduced no significant performance degradation. For each cryptographic configuration, the total average connection time remained closely aligned with the original handshake-only benchmarks. For instance, the RSA-3072 handshake averaged 9.22 ms, while the TLS+HTTP exchange averaged 9.17 ms. Even in the most computationally intensive case, SPHINCS+ sha2-128f-simple, the full TLS+HTTP flow remained consistent at 36.20 ms compared to the 38.89 ms measured during the handshake alone.

These results validate that once a secure channel is negotiated, application data can be transmitted with negligible additional delay, confirming the reliability of our end-to-end TLS implementation when extended to support HTTP communication. This also reinforces the viability of PQC-based authentication schemes in realistic post-handshake scenarios.

Table 16: TLS with HTTP Request Results in ms

Algorithm	**Avg. Time**	**Certificate Size**
RSA-3072	9.166	1.4 KB
ML-DSA-44	5.551	5.3 KB
FALCON-512	6.152	2.4 KB
SPHINCS+ 128f	36.205	22.9 KB
RSA-3072 & SPHINCS+ 128f	47.892	24.0 KB
RSA-3072 & ML-DSA-44	10.296	6.0 KB
RSA-3072 & FALCON	11.714	3.4 KB

10.10 Feasibility Conclusion

Across both vehicular communication networks and secure internet protocols, the feasibility of SPHINCS+ is consistently challenged by its performance limitations. In C-V2X systems, which impose stricter timing and bandwidth requirements than traditional DSRC-based V2V, SPHINCS+'s large signature size and high computational cost result in excessive transmission delays and the potential for missed communication windows. This makes SPHINCS+ an unacceptable candidate for the even stricter real-time vehicular authentication. This again highlights the urgent need for more efficient, non-lattice-based backup post-quantum signature schemes.

By contrast, the TLS 1.3 protocol does not impose explicit size or timing constraints, making it a more accommodating environment for exploring the use of post-quantum algorithms such as SPHINCS+. However, even without formal limitations, performance still plays a critical role in user experience and system scalability. The integration of PQC into TLS 1.3 is a necessary evolution to maintain the security of internet communications in the post-quantum era. SPHINCS+ introduces significantly higher handshake latency and certificate overhead compared to other post-quantum options. Based on the results presented, ML-DSA and FALCON offer practical options for deployment in modern TLS stacks.

Though SPHINCS+ offers cryptographic conservatism and quantum resilience, but these strengths come at a high performance cost. While it's technically still a strong candidate for TLS 1.3, it lags behind ML-DSA and FALCON significantly and would have an impact on users. However, in this scenario, it does do its job as a backup alternative.

11 Future Work

Given that SPHINCS+ variants have been ruled out for V2V authentication due to their significant signature sizes and high verification latency, future research must focus on identifying and evaluating feasible quantum-safe alternatives. This study demonstrates that FALCON-512 currently offers the most viable path forward for integration into bandwidth- and delay-sensitive V2X systems. However, as the PQC landscape continues to evolve, this conclusion must be treated as provisional. If a new digital signature scheme is standardized or adopted by the community, similar benchmarking and performance validation, particularly in the context of constrained V2V environments, should be repeated. The rigorous testing framework presented in this work provides a reusable methodology for such evaluations. The IEEE 802.11 standard and associated work presented focused on DSRC-based communication. However, C-V2X introduces stricter constraints on message size and timing due to its LTE-based physical layer. Future research should evaluate how and if PQC signatures can be compressed, split, or otherwise optimized to function within these bounds.

Another area of important work includes extending the Haskell-based crypton library [17] to support PQC primitives, in line with emerging NIST standards. Implementing stateless signature schemes such as SLH-DSA and alternatives like ML-DSA and FALCON would ensure long-term resilience against quantum threats. As crypton is written in Haskell, a memory-safe, statically typed, and purely functional language, these additions would benefit from strong compile-time guarantees and safety properties. While Rust ecosystems have already begun incorporating these standards, enabling similar support in Haskell would ensure equality across memory-safe languages.

Looking ahead, an essential direction for future research is the integration of PQC signature schemes into Bitcoin Core. While NIST standardized schemes like SPHINCS+ offer strong security under quantum threat models, their large signature sizes introduce substantial overhead. Future work would involve adapting the Bitcoin transaction to accommodate these larger signatures without breaking consensus rules. Ideally, explore the PQC scheme FALCON as it is the most promising candidate with the least amount of disruption.

In parallel, an urgent area of investigation involves legacy transactions and UTXOs protected only by ECDSA. These vulnerable outputs pose a significant long-term security risk, since public keys become visible upon spending, exposing them to quantum-enabled front-running attacks. To address this, we would like to explore automated migration protocols that can identify and flag high-risk UTXOs, encouraging users to proactively transfer their funds to quantum-safe addresses. Additionally, we want to investigate protocol-level solutions that enforce quantum-safe key rotation after a given block height. This would effectively sunset ECDSA over time, nudging the network toward PQC adoption without requiring a disruptive hard fork. Addressing this legacy vulnerability is critical to achieving full-spectrum quantum resistance for Bitcoin, ensuring not only that future transactions are secure but also that past funds remain protected.

Lastly, an area we would like to investigate more is the deployment of SPHINCS+ in high-throughput, non-latency-sensitive environments using hardware acceleration. Recent research has shown that SPHINCS+ can be significantly optimized using GPU-based parallelism, with throughput improvements exceeding 1,000× over CPU implementations [23]. Future work should explore the real-world deployment of these GPU-accelerated variants as well as continue testing different optimization routes.

12 Conclusion

This thesis set out to determine the readiness of SPHINCS+ for standardization by evaluating its theoretical strengths alongside its practical limitations across a range of use cases. SPHINCS+, now formally recognized under the FIPS 205 SLH-DSA standards, is well recognized for its conservative cryptographic design, relying solely on well-understood hash-based primitives rather than newer algebraic assumptions. Its stateless nature and strong resistance to quantum attacks made it a compelling candidate for long-term digital signature schemes.

However, our analysis demonstrates that this security assurance comes at a significant cost. SPHINCS+'s large signature sizes and slower performance metrics create practical limitations, especially in environments with strict bandwidth, latency, or storage constraints. In particular, its use in V2V is impeded by frame size caps that make even standalone SPHINCS+ signatures challenging to deploy, let alone in hybrid schemes where the inclusion of a second signature further compounds this issue.

When evaluated in blockchain systems, SPHINCS+ proved similarly impractical. While its statelessness and long-term reliability are appealing for decentralized applications, the inflated transaction size drastically impacts performance and storage, which are critical dimensions in blockchain deployment.

Hybrid schemes that combine SPHINCS+ with classical signatures like ECDSA were also evaluated. Though they enhance transitional security by offering quantum resilience even if one component is broken, they inherit the size and performance drawbacks of SPHINCS+. Thus, while hybridization may offer a short-term strategy during migration, it cannot fully mitigate the long-term inefficiencies tied to SPHINCS+.

The findings suggest that, while SPHINCS+ offers strong security against quantum computers and has been standardized by NIST, its substantial performance drawbacks raise critical concerns about its practicality. These limitations indicate that SPHINCS+, despite its conservative design, may not be suitable for many real-world applications and, arguably, should not have advanced to standardization in its current form. Given the potential for breakthroughs in lattice cryptanalysis, the cryptographic community cannot afford to rely solely on lattice-based schemes such as FALCON or ML-DSA.

Therefore, we urge NIST to proactively identify and support the development of an additional post-quantum signature scheme that is not based on lattice assumptions and does not suffer from the size and performance constraints that characterize SPHINCS+. Such a backup is essential to ensure long-term cryptographic resilience across diverse applications and infrastructures.

References

[1] Gorjan Alagic, Maxime Bros, Pierre Ciadoux, David Cooper, et al. *Status Report on the First Round of the Additional Digital Signature Schemes for the NIST Post-Quantum Cryptography Standardization Process.* Tech. rep. NIST IR 8528. Gaithersburg, MD: National Institute of Standards and Technology, 2024. DOI: 10.6028/NIST.IR.8528. URL: https://doi.org/10.6028/NIST.IR.8528.

[2] Andreas M. Antonopoulos. *Mastering Bitcoin: Unlocking Digital Cryptocurrencies.* O'Reilly Media, 2015. ISBN: 978-1449374044.

[3] Jean-Philippe Aumasson, Daniel J. Bernstein, Christoph Dobraunig, Maria Eichlseder, et al. *SPHINCS+.* Submission to NIST's post-quantum crypto standardization project. 2022. URL: https://sphincs.org/data/sphincs+-r3.1-specification.pdf.

[4] Itan Barmes, Bram Bosch, and Olaf Haalstra. *Quantum Computers and the Bitcoin Blockchain.* https://www.deloitte.com/nl/en/services/risk-advisory/perspectives/quantum-computers-and-the-bitcoin-blockchain.html. 2020.

[5] Charles H. Bennett, Ethan Bernstein, Gilles Brassard, and Umesh Vazirani. "Strengths and Weaknesses of Quantum Computing". In: *SIAM Journal on Computing* 26.5 (Oct. 1997), pp. 1510–1523. ISSN: 1095-7111. DOI: 10.1137/s0097539796300933. URL: http://dx.doi.org/10.1137/S0097539796300933.

[6] D.J. Bernstein, D. Hopwood, A. Hülsing, T. Lange, et al. "SPHINCS: Practical Stateless Hash-Based Signatures". In: *Advances in Cryptology – EUROCRYPT 2015.* Ed. by E. Oswald and M. Fischlin. Vol. 9056. Lecture Notes in Computer Science. Springer, Berlin, Heidelberg, 2015, pp. 368–397. DOI: 10.1007/978-3-662-46800-5_15.

[7] Daniel J. Bernstein, Andreas Hülsing, Stefan Kölbl, Ruben Niederhagen, et al. "The SPHINCS+ Signature Framework". In: *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security (CCS '19).* ACM, 2019. DOI: 10.1145/3319535.3363229.

[8] Nina Bindel, Udyani Herath, Matthew McKague, and Douglas Stebila. *Transitioning to a Quantum-Resistant Public Key Infrastructure.* Tech. rep. 2017/460. Cryptology ePrint Archive, 2017. URL: https://eprint.iacr.org/2017/460.

[9] T. J. Borrelli and Stanisław P. Radziszowski. *CSCI.764.01 – Quantum-Resistant Cryptography.* Lecture slides, Rochester Institute of Technology. 2025.

[10] Wouter Castryck and Thomas Decru. *An efficient key recovery attack on SIDH.* Cryptology ePrint Archive, Paper 2022/975. 2022. URL: https://eprint.iacr.org/2022/975.

[11] Ming-shing Chen, Jintai Ding, Matthias Kannwischer, et al. *Rainbow: A Multivariate Signature Scheme.* https://www.pqcrainbow.org. Accessed: March 23, 2025. 2025.

[12] Coinbase. *What is Segregated Witness (SegWit).* Accessed: 2025-05-15. n.d. URL: https://www.coinbase.com/learn/crypto-glossary/what-is-segregated-witness-segwit.

[13] David A. Cooper, Daniel C. Apon, Quynh H. Dang, Michael S. Davidson, et al. *Recommendation for Stateful Hash-Based Signature Schemes.* NIST Special Publication 800-208. National Institute of Standards and Technology, Oct. 2020. DOI: 10.6028/NIST.SP.800-208. URL: https://doi.org/10.6028/NIST.SP.800-208.

[14] ISARA Corporation. *Math paths to quantum-safe security: Hash-based cryptography.* 2020. URL: https://www.isara.com/blog-posts/hash-based-cryptography.html.

[15] Q. Dang. "Recommendation for Applications Using Approved Hash Algorithms". In: *National Institute of Standards and Technology* (2012).

[16] Pierre-Alain Fouque, Jeffrey Hoffstein, Paul Kirchner, et al. *Falcon: Fast-Fourier Lattice-based Compact Signatures over NTRU.* 2024. URL: https://falcon-sign.info.

[17] Vincent Hanquez. *crypton.* https://hackage.haskell.org/package/crypton. n.d.

[18] Andreas Huelsing. "RFC 8391: XMSS: Extended Merkle Signature Scheme". In: *IETF Datatracker* (2018). URL: https://datatracker.ietf.org/doc/html/rfc8391.

[19] Andreas Huelsing. *SPHINCS+ – The smaller SPHINCS.* 2017. URL: https://huelsing.net/wordpress/?p=558.

[20] Andreas Hülsing, Stefan-Lukas Gazdag, Denis Butin, and Johannes Buchmann. "Hash-based Signatures: An Outline for a New Standard". In: *NIST Workshop on Cybersecurity in a Post-Quantum World.* Gaithersburg, MD, Apr. 2015. URL: https://csrc.nist.gov/csrc/media/events/workshop-on-cybersecurity-in-a-post-quantum-world/documents/papers/session5-hulsing-paper.pdf.

[21] IBM Quantum. *Grover's Algorithm — Fundamentals of Quantum Algorithms.* https://learning.quantum.ibm.com/course/fundamentals-of-quantum-algorithms/grovers-algorithm. 2025.

[22] "IEEE Standard for Wireless Access in Vehicular Environments–Security Services for Application and Management Messages". In: *IEEE Std 1609.2-2022 (Revision of IEEE Std 1609.2-2016)* (2023), pp. 1–349. DOI: 10.1109/IEEESTD.2023.10075082.

[23] DongCheon Kim, HoJin Choi, and Seog Chung Seo. "Parallel Implementation of SPHINCS+ With GPUs". In: *IEEE Transactions on Circuits and Systems I: Regular Papers* 71.6 (2024), pp. 2810–2823. DOI: 10.1109/TCSI.2024.3370802.

[24] Stefan Kölbl, Morten M. Lauridsen, Florian Mendel, and Christian Rechberger. "Haraka v2 – Efficient Short-Input Hashing for Post-Quantum Applications". In: *IACR Transactions on Symmetric Cryptology* 2016.2 (2017), pp. 1–29. DOI: 10.13154/tosc.v2016.i2.1-29. URL: https://doi.org/10.13154/tosc.v2016.i2.1-29.

[25] Hugo Krawczyk, Kenneth G. Paterson, and Hoeteck Wee. "On the Security of the TLS Protocol: A Systematic Analysis". In: *Advances in Cryptology – CRYPTO 2013.* Ed. by Ran Canetti and Juan A. Garay. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 429–448.

[26] Mikhail Kudinov, Andreas Hülsing, Eyal Ronen, and Eylon Yogev. *SPHINCS+C: Compressing SPHINCS+ With (Almost) No Cost.* Cryptology ePrint Archive, Paper 2022/778. 2022. URL: https://eprint.iacr.org/2022/778.

[27] Claire Lee. *Floyd's Cycle Detection Algorithm.* 2022. URL: https://yuminlee2.medium.com/floyds-cycle-detection-algorithm-b27ed50c607f.

[28] Xiuhan Lin, Mehdi Tibouchi, Yang Yu, and Shiduo Zhang. "Do Not Disturb a Sleeping Falcon: Floating-Point Error Sensitivity of the Falcon Sampler and Its Consequences". In: (2024). URL: https://eprint.iacr.org/2024/1709.

[29] Dustin Moody, Gorjan Alagic, Daniel Apon, David Cooper, et al. *Status Report on the Second Round of the NIST Post-Quantum Cryptography Standardization Process.* NIST Interagency/Internal Report (NISTIR) 8309. Gaithersburg, MD: National Institute of Standards and Technology, 2020. DOI: 10.6028/NIST.IR.8309. URL: https://doi.org/10.6028/NIST.IR.8309.

[30] Dustin Moody, Ray Perlner, A. Regenscheid, Angela Robinson, et al. *Transition to Post-Quantum Cryptography Standards.* NIST Internal Report (IR) 8547 ipd. Gaithersburg, MD: National Institute of Standards and Technology, 2024. DOI: 10.6028/NIST.IR.8547.ipd. URL: https://doi.org/10.6028/NIST.IR.8547.ipd.

[31] Arvind Narayanan, Joseph Bonneau, Edward Felten, Andrew Miller, et al. *Bitcoin and Cryptocurrency Technologies: A Comprehensive Introduction.* USA: Princeton University Press, 2016. ISBN: 0691171696.

[32] National Institute of Standards and Technology. *Advanced Encryption Standard (AES).* Tech. rep. FIPS PUB 197. Washington, DC: U.S. Department of Commerce, 2001. DOI: 10.6028/NIST.FIPS.197. URL: https://doi.org/10.6028/NIST.FIPS.197.

[33] National Institute of Standards and Technology. *Digital Signature Standard (DSS).* Tech. rep. FIPS PUB 186-4. Washington, D.C.: U.S. Department of Commerce, 2013. DOI: 10.6028/NIST.FIPS.186-4. URL: https://doi.org/10.6028/NIST.FIPS.186-4.

[34] National Institute of Standards and Technology. *FIPS PUB 180-4: Secure Hash Standard (SHS).* Tech. rep. FIPS PUB 180-4. U.S. Department of Commerce, Aug. 2015. DOI: 10.6028/NIST.FIPS.180-4. URL: https://doi.org/10.6028/NIST.FIPS.180-4.

[35] National Institute of Standards and Technology. *FIPS PUB 202: SHA-3 Standard: Permutation-Based Hash and Extendable-Output Functions.* Tech. rep. FIPS PUB 202. U.S. Department of Commerce, Aug. 2015. DOI: 10.6028/NIST.FIPS.202. URL: https://doi.org/10.6028/NIST.FIPS.202.

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

[37] Dana Nickerson. *Collision Detection and Pollard's Rho Algorithm for the Discrete Logarithm Problem.* Undergraduate thesis, Carleton University. Supervisor: Daniel Panario. May 2020.

[38] Christof Paar and Jan Pelzl. *Understanding Cryptography: A Textbook for Students and Practitioners.* Springer, 2010.

[39] Christof Paar, Jan Pelzl, and Tim Güneysu. *Understanding Cryptography. From Established Symmetric and Asymmetric Ciphers to Post-Quantum Algorithms.* 2nd ed. eBook ISBN: 978-3-662-69007-9; Published: May 15, 2024; Hardcover ISBN: 978-3-662-69006-2; Published: May 16, 2024; Pages: XXI, 543. Springer Berlin, Heidelberg, 2024. ISBN: 978-3-662-69006-2. DOI: 10.1007/978-3-662-69007-9. URL: https://doi.org/10.1007/978-3-662-69007-9.

[40] Tim Polk, Sean Turner, Lily Chen, and Matthew Campagna. *Guidelines for the Selection, Configuration, and Use of Transport Layer Security (TLS) Implementations.* Tech. rep. NIST SP 800-52 Rev. 2. National Institute of Standards and Technology, 2020. URL: https://doi.org/10.6028/NIST.SP.800-52r2.

[41] Eric Rescorla. *The Transport Layer Security (TLS) Protocol Version 1.3.* RFC 8446. Aug. 2018. DOI: 10.17487/RFC8446. URL: https://www.rfc-editor.org/info/rfc8446.

[42] National Institute of Standards and Technology. *Call for Proposals: Post-Quantum Cryptography Standardization.* https://csrc.nist.gov/projects/post-quantum-cryptography/post-quantum-cryptography-standardization/call-for-proposals. 2017.

[43] National Institute of Standards and Technology. *Digital Signature Standard (DSS).* Tech. rep. Federal Information Processing Standards Publication (FIPS) NIST FIPS 186-5. Washington, D.C.: Department of Commerce, 2023. DOI: 10.6028/NIST.FIPS.186-5. URL: https://doi.org/10.6028/NIST.FIPS.186-5.

[44] National Institute of Standards and Technology. *Module-Lattice-Based Digital Signature Standard.* Tech. rep. Federal Information Processing Standards Publication (FIPS) NIST FIPS 204. Washington, D.C.: Department of Commerce, 2024. DOI: 10.6028/NIST.FIPS.204. URL: https://doi.org/10.6028/NIST.FIPS.204.

[45] National Institute of Standards and Technology. *Module-Lattice-Based Key-Encapsulation Mechanism Standard.* Tech. rep. Federal Information Processing Standards Publication (FIPS) NIST FIPS 203. Washington, D.C.: Department of Commerce, 2024. DOI: 10.6028/NIST.FIPS.203. URL: https://doi.org/10.6028/NIST.FIPS.203.

[46] 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. Accessed: March 23, 2025. 2024.

[47] National Institute of Standards and Technology. "Stateless Hash-Based Digital Signature Standard". In: *Federal Information Processing Standards Publication (FIPS) NIST FIPS 205 ipd* (2023). DOI: 10.6028/NIST.FIPS.205.ipd.

[48] National Institute of Standards and Technology. *Stateless Hash-Based Digital Signature Standard.* Tech. rep. Federal Information Processing Standards Publication (FIPS) NIST FIPS 205. Washington, D.C.: Department of Commerce, 2024. DOI: 10.6028/NIST.FIPS.205. URL: https://doi.org/10.6028/NIST.FIPS.205.

[49] D. Stinson and M. Paterson. *Cryptography: Theory and Practice.* CRC Press, 2018. URL: https://books.google.com/books?id=nHxqDwAAQBAJ.

[50] The NIST PQC Team. *PQC standardization process: Announcing four candidates to be standardized, plus fourth round candidates.* 2022. URL: https://www.nist.gov/news-events/news/2022/07/pqc-standardization-process-announcing-four-candidates-be-standardized-plus.

[51] Edlyn Teske. *NIST PQC Finalists Update: It's Over for the Rainbow.* https://www.cryptomathic.com/blog/nist-pqc-finalists-update-its-over-for-the-rainbow. 2022.

[52] Jay Thakkar. *TLS 1.3 Handshake: Taking a Closer Look.* Accessed: 2025-05-14. Mar. 2018. URL: https://www.thesslstore.com/blog/tls-1-3-handshake-tls-1-2/.

[53] The Block. *Bitcoin on-chain data and charts for transactions, addresses, and miners.* https://www.theblock.co/data/on-chain-metrics/bitcoin. 2025.

[54] Geoff Twardokus, Nina Bindel, Hanif Rahbari, and Sarah McCarthy. "When Cryptography Needs a Hand: Practical Post-Quantum Authentication for V2V Communications". In: *Network and Distributed System Security (NDSS) Symposium 2024.* San Diego, CA, USA, 2024. DOI: 10.14722/ndss.2024.24267.

[55] Robert S. Winternitz. "A Secure One-Way Hash Function Built from DES". In: *1984 IEEE Symposium on Security and Privacy.* 1984, pp. 88–88. DOI: 10.1109/SP.1984.10027.

[56] YCharts. *Bitcoin Average Transaction Fee.* https://ycharts.com/indicators/bitcoin_average_transaction_fee. Accessed: 2025-05-13. 2024.

[57] M. Yehia, R. AlTawy, and T. A. Gulliver. "Verifiable Obtained Random Subsets for Improving SPHINCS+". In: *Information Security and Privacy. ACISP 2021.* Ed. by J. Baek and S. Ruj. Vol. 13083. Lecture Notes in Computer Science. Springer, Cham, 2021. DOI: 10.1007/978-3-030-90567-5_35. URL: https://doi.org/10.1007/978-3-030-90567-5_35.

A Appendix: PQC Definitions

Post-Quantum Cryptography/Quantum-Resistant Cryptography is the study of classical (non-quantum) system that are intended to be resistant to QC [9]. QRC is the goal, PQC is the path to that goal, involving NIST's standardization of PQC schemes that will replace quantum vulnerable schemes.

A.1 Quantum Computer

A quantum computer capable of breaking modern cryptographic schemes must be general-purpose, large-scale, and fault-tolerant. A general-purpose quantum computer could run a broad range of quantum algorithms. Many quantum systems today have tens to hundreds of qubits, but these are not enough to break real-world cryptography. A true large-scale quantum computer would need 1,000 to several million logical qubits. Lastly, to break cryptography reliably, you need fault-tolerant systems where quantum error correction compensates for hardware noise [9].

Quantum computing leverages principles of quantum mechanics to perform certain computations exponentially faster than classical methods. Central to this model is the qubit, or quantum bit, which is the quantum analog of a classical bit. Unlike a classical bit, which exists in a state of either 0 or 1, a qubit exists in a superposition of both states [9]. Mathematically, the state of a single qubit is represented as:

$$|\psi\rangle = \alpha|0\rangle + \beta|1\rangle \tag{9}$$

Here, α and β are complex-valued probability amplitudes such that:

$$|\alpha|^2 + |\beta|^2 = 1 \tag{10}$$

A single qubit can also be visualized geometrically on the block sphere, with the general form:

$$|\psi\rangle = \cos\left(\frac{\theta}{2}\right)|0\rangle + e^{i\phi}\sin\left(\frac{\theta}{2}\right)|1\rangle \tag{11}$$

Upon measurement, the qubit collapses to the classical state $|0\rangle$ with probability $|\alpha|^2$ and to $|1\rangle$ with probability $|\beta|^2$. The act of measurement forces the qubit to choose a definite value, and the superposition is lost. This is why quantum algorithms are designed to delay measurement until the final step. Until then, qubits remain in superposition and entanglement, allowing quantum computers to explore many possibilities in parallel.

A.2 Quantum Time Complexity and BQP

In computational complexity theory, the class BQP encompasses all decision problems that can be solved by a quantum computer. BQP is often considered the quantum analogue of the classical class bounded-error probabilistic polynomial time, BPP, which includes problems solvable by randomized algorithms within polynomial time and bounded error [9]. Notably, BQP includes all problems in P, and it is known to be contained within PSPACE, but its exact relation to NP and NP-complete problems remains unresolved. For instance, factoring, efficiently solved by Shor's algorithm, is in $\mathsf{BQP} \cap \mathsf{NP} \cap \mathsf{co\text{-}NP}$, but is not known to be NP-complete.

The existence of BQP challenges the Strong Church-Turing Thesis, which asserts that all efficient computation can be simulated by a classical Turing machine [9]. Quantum algorithms such as Shor's and Grover's demonstrate that, for certain problems, quantum computers provide exponential or quadratic speedups, respectively. These results suggest that quantum computers may solve specific classes of problems more efficiently than classical computers, though they are unlikely to solve all hard problems, such as those that are NP-complete, without further breakthroughs in quantum algorithm design.

B Appendix: Benchmarking Overview

The benchmarking suite was implemented in `C++` to evaluate the performance of several digital signature algorithms, including ECDSA, FALCON, SPHINCS+, and ML-DSA. Each scheme is benchmarked using a consistent structure that captures timing and CPU usage metrics across multiple iterations. The full benchmarking implementation used in this objective is available **here https://github.com/jancillotti/MS_Thesis**, under the `objective-one/` directory.

B.1 Setting up the Environment

The project uses `CMake` as the primary build system and requires a C++ compiler with support for the C++20 standard. The following tools must be installed on the system:

- **CMake (version $\geq$ 3.30)**: Used to generate platform-specific build files.
- **GCC or Clang**: A C++20-compliant compiler such as `gcc` (version 10 or higher) or `clang`.
- **pkg-config**: Utility used to locate installed libraries and their compilation flags.

The project depends on three external libraries: `Botan`, `liboqs`, and `OpenSSL`. These libraries must be installed and discoverable via `pkg-config`.

`Botan` is a cryptographic library used for implementing traditional digital signatures and most of the selected PQC schemes, the creators are waiting for the official FIPS document before releasing FALCON.

`liboqs` is the Open Quantum Safe library, providing implementations of post-quantum cryptographic algorithms such as FALCON.

`OpenSSL` is a widely-used library that provides cryptographic primitives and is used in this project for ECDSA-related functionality.

Once all dependencies are installed and configured, the project can be built using the following commands:

```
mkdir build
cd build
cmake ..
make -j$(nproc)
```

B.2 Code Architecture

- Each signature algorithm is implemented in its own file (e.g., `ecdsa.cpp`, `falcon.cpp`) and provides a function `run_<scheme>_benchmark(std::ostream&)`.
- Shared data structures and statistics utilities are declared in `util.h` and implemented in `util.cpp`.
- The `main.cpp` file executes all benchmarks and directs output to a file.

B.3 Core Benchmarking Workflow

Each benchmarked algorithm follows a three-phase process:

1. **Key Generation:** Measures the time required to generate the public/private key pair.
2. **Signing:** Measures both wall-clock time and CPU usage required to sign a fixed-size message.
3. **Verification:** Measures the wall-clock and CPU time needed to verify the signature.

B.4 Result Collection

The results of Table 8 were found using the following system.

- **Model:** MacBook Pro (14-inch, 2021)
- **Chip:** Apple M1 Pro
  - 8-core CPU
- **Memory:** 16 GB
- **Architecture:** ARM64 (AArch64)

Each test is repeated for `ITERATIONS = 1000`. For each phase, the following statistics are computed:

- Average key generation time
- Average and standard deviation of signing and verification times
- Average CPU time for signing and verification

C Appendix: V2V Code Overview

To support reproducibility and transparency, this thesis utilizes the PQ-Benchmarking artifact introduced in Twardokus [54], submitted to NDSS 2024. The benchmarking suite provides a standardized environment for measuring cryptographic performance, specifically targeting post-quantum digital signature schemes on commodity hardware. The implementation used in this objective can be found under the `objective-two/` directory located **here https://github.com/jancillotti/MS_Thesis**.

The artifact includes all necessary C++ source files, a functional test script, and detailed environment setup instructions, including a VirtualBox-based Ubuntu 22.04 image. It supports benchmarking across both x86 and cross-compiled ARM platforms, with integration of the Botan cryptographic library and the liboqs library from the Open Quantum Safe project.

In addition to the configurations evaluated in the original publication, we extended the benchmarking framework to include the following SPHINCS+ variants: SHAKE-based SPHINCS+ and SPHINCS+ Simple SHA2 variant.

These extensions involved updating the `functional_test.bash` script and modifying per-algorithm result handlers to include additional output files for the added variants. Benchmark results were recorded and analyzed for average key generation, signing, and verification times, and were incorporated into Section 8.6 of this thesis.

The `plot.py` script provides a visual comparison of the total byte size required for various digital signature schemes when used in DSRC, highlighting their impact on frame size constraints. This script is particularly useful for illustrating the feasibility of integrating PQC signatures into DSRC-based vehicular networks.

The script generates a stacked bar chart comparing the total size (in bytes) of:

- Public key
- Signature
- Additional overhead

across different schemes: ECDSA, ML-DSA, FALCON, SPHINCS+ Robust, and SPHINCS+ Simple.

The code architecture for the hybrid schemes benchmarking is a similar setup to Appendix B.

D Appendix: Blockchain Code Overview

This appendix provides an overview of the C++ codebase used to benchmark and evaluate digital signature schemes for blockchain-style transaction signing and verification. The code simulates simplified Bitcoin-like transactions and measures the performance of classical and post-quantum cryptographic schemes. The implementation used in this objective is available **here https://github.com/jancillotti/MS_Thesis**, under the `objective-three/` directory.

D.1 Benchmark Runner (`benchmark_runner.h`, `benchmark_runner.cpp`)

This module implements the main benchmarking logic for each cryptographic signature scheme:

- **ECDSA** (`secp256k1`, via Botan)
- **ML-DSA** (`ML_DSA_4x4`, via Botan)
- **FALCON** (`liboqs`, FALCON-512)
- **SPHINCS+** (Simple and Robust, via Botan)
- **SPHINCS+ Hybrid** (ECDSA + SPHINCS+ Robust)
- **FALCON+ Hybrid** (ECDSA + FALCON)
- **ML-DSA+ Hybrid** (ECDSA + ML-DSA)

Each function in the format `run_<scheme>()` performs the following steps:

1. Key pair generation and size measurement
2. Address derivation using `RIPEMD160(SHA256(pubkey))`
3. Transaction construction and serialization
4. Signing and verification of the transaction hash
5. Timing and size collection

All results are stored in a `BenchmarkResult` structure, which tracks operation timings, data sizes, and validity of signature verification.

D.2 Entry Point (`main.cpp`)

The main function orchestrates benchmarking for each scheme:

- Runs each benchmark 100 times via `average_results()`
- Computes average timings and sizes
- Prints results in a formatted table
- Confirms signature verification success

D.3 Transaction Serialization

Simulates a Bitcoin-like transaction format in `transaction_serializer.h`, `transaction_serializer.cpp`:

- Uses versioning, inputs, outputs, and locktime
- Includes `scriptSig` with dummy public key and signature data based on the scheme
- Uses correct signature and key sizes for each scheme (e.g., 7856 bytes for SPHINCS+ Robust, 64 bytes for ECDSA)
- Computes transaction size for benchmarking

D.4 Utilities

This section describes the modules used to support hashing and serialization in the benchmarking framework. These utilities help model realistic blockchain behavior,, such as Bitcoin-style address hashing and compact integer encoding.

Hashing (`hash_utils.h`, `hash_utils.cpp`) implements a Bitcoin-style address derivation hash by performing a double SHA-256 followed by a RIPEMD-160 hash, emulating how Bitcoin generates addresses from public keys.

Variable-Length Integer Encoding (`varint.h`, `varint.cpp`) implements a variable-length integer encoding scheme. This encoding follows Bitcoin's compact integer format, where small values use fewer bytes, thereby conserving space and closely modeling the serialization used in real blockchain transactions.

D.5 Experimental Setup

All results were collected on the following machine:

- **Model:** MacBook Pro (14-inch, 2021)
- **Chip:** Apple M1 Pro
  - 8-core CPU
- **Memory:** 16 GB
- **Architecture:** ARM64 (AArch64)

E Appendix: TLS Benchmarking Code Overview

This appendix details the Python-based benchmarking framework used to evaluate the performance of TLS 1.3 handshakes under various cryptographic certificate configurations. The framework comprises five core scripts: `generate_certs.py`, `tls_server.py`, `tls_client.py`, `tsl_test.py`, and `tls_run_all.py`. The implementation used in this objective is available **here https://github.com/jancillotti/MS_Thesis**, under the `objective-four/` directory.

E.1 Setting up the Environment

Since most post-quantum cryptographic algorithms are not included in standard OpenSSL distributions, we compiled a custom version from source. If you are on OpenSSL version 3.5, you will need to follow the steps in Appendix F. If you are on ¡3.5 you can proceed as normal here. The procedure was as follows:

1. Clone the `oqs-provider` repository from GitHub:

   ```
   git clone https://github.com/open-quantum-safe/oqs-provider
   ```

2. Install the required dependencies: `cmake` and `ninja` for building, using the commands:

   ```
   brew install cmake ninja
   ```

3. Run the full build script provided by the project:

   ```
   ./scripts/fullbuild.sh
   ```

This script compiles the OQS library (`liboqs`), a local OpenSSL instance, and installs the `oqsprovider` module into the local OpenSSL modules directory (typically `./local/lib/ossl-modules/`).OpenSSL providers must be explicitly activated. To enable both the default and `oqsprovider` providers globally, we modified the OpenSSL configuration file. On macOS with Homebrew, this file is typically located at:

`/opt/homebrew/etc/openssl@3/openssl.cnf`

The following block was appended to the configuration file:

```
[provider_sect]
default = default_sect
oqsprovider = oqsprovider_sect

[default_sect]
activate = 1

[oqsprovider_sect]
activate = 1
module = /absolute/path/to/local/lib/ossl-modules/oqsprovider.dylib
```

This configuration ensures that both providers are loaded, enabling support for quantum-safe algorithms and access to essential cryptographic primitives (e.g., secure randomness and hashing). Correct activation can be verified using:

```
openssl list -providers -verbose
```

E.2 generate_certs.py: Certificate Generation Automation

This script automates the creation of X.509 certificates using `openssl` for classical, post-quantum, and hybrid cryptographic algorithms. The supported configurations include:

- Classical: RSA 3072
- Post-Quantum: SPHINCS+ (sha2-128f-simple), FALCON-512, ML-DSA 44
- Hybrid: RSA + SPHINCS+, RSA + FALCON, RSA + ML-DSA

Certificates are saved under the `certs/` directory and named to reflect their algorithm. All certificate generation is handled via `subprocess.run()` calls to `openssl req`, using non-interactive inputs for automation and reproducibility.

E.3 tls_server.py: TLS 1.3 Server Implementation

This script launches a TLS 1.3-only server using Python's built-in `ssl` and `socket` modules. It loads a provided certificate and key, listens on a specified port, and completes a handshake upon client connection. After sending a simple greeting message, it closes the connection. The server enforces TLS 1.3 using `ssl.PROTOCOL_TLS_SERVER` with version bounds set explicitly.

E.4 tls_client.py: TLS 1.3 Client and Timer

The client connects to the server using `ssl.create_default_context()`, configured to use only TLS 1.3. It records the full handshake duration using Python's `time` module and verifies the server's certificate. The resulting handshake time (in seconds) is returned for performance comparison.

E.5 tsl_test.py: Test Harness and Packet Capture

This module runs end-to-end handshake tests:

- Starts a `tcpdump` process for optional packet capture
- Spawns the server as a background process
- Executes the client and records handshake time
- Shuts down the server and captures certificate size

It returns a tuple containing the algorithm name, average handshake time, and certificate size in kilobytes.

E.6 http_server.py and http_client.py: Post-TLS HTTP Testing

To verify the integrity of post-handshake communication, a lightweight HTTP test was conducted using TCP sockets. The `http_server.py` script launches a simple HTTP server that listens on a port, accepts a single client connection, and responds to a basic `GET` request with an HTTP/1.1-compliant response. This includes a status line, a minimal header, and a short payload.

The `http_client.py` script connects to the server using a TCP socket, manually sends a properly formatted HTTP request string (`GET/HTTP/1.1\r\nHost:localhost\r\n\r\n`), and reads the response. These scripts confirm that application-layer data transmission functions correctly once a TLS session has been negotiated.

E.7 http_test.py: Automated HTTP Communication Validator

To orchestrate the client-server interaction and validate the timing and consistency of HTTP exchanges, the `http_test.py` script acts as a test harness. It performs the following steps:

- Launches the HTTP server as a subprocess
- Waits briefly to ensure server availability
- Runs the HTTP client to initiate and complete the request
- Terminates the server once the exchange is complete

E.8 tls_run_all.py: Test Runner and Aggregator

This is the main entry point for benchmarking:

- Calls `generate_certs()` to create or load certificates
- Iteratively runs `run_test()` 100 times per algorithm
- Averages the connection duration using `statistics.mean`
- Writes results to terminal

E.9 Experimental Setup

All results were collected on the following machine:

- **Model:** MacBook Pro (14-inch, 2021)
- **Chip:** Apple M1 Pro
  - 8-core CPU
- **Memory:** 16 GB
- **Architecture:** ARM64 (AArch64)

F Append: OpenSSL 3.5 Update

Due to the OpenSSL 3.5 update and disabling most of the PQC algorithms to run Appendix E there is additional steps needed.

- **OpenSSL version 3.4.1**: Install this version into a directory of your choosing.
- **Rebuild OQS**: Assuming you have OQS from previous steps in this Appendix, you will need to rebuild it targeting the custom OpenSSL 3.4.1. Then follow the steps from the Appendix E.1 appending to the configuration file. This file must match the OpenSSL 3.4.1 install you did.
- **Python3**: Build a new python3 with the custom OpenSSL
  `../configure --with-openssl=<your OpenSSL 3.4.1 path>`

To run the Botan environments, no additional steps are needed but there is a bug in the OQS version that Botan is using that will print this error `error registering mldsa44 with no hash` but does not exit so the programs will continue to run.