XMSS-based Chain of Trust

Soundes Marzougui[1] and Jean-Pierre Seifert[2]

[1] Technische Universität Berlin, Berlin, Germany
soundes.marzougui@tu-berlin.de
[2] Technische Universität Berlin, Berlin, Germany
Jean-Pierre.Seifert@external.telekom.de

Abstract

Given that large-scale quantum computers can eventually compute discrete logarithm and integer factorization in polynomial time [44], all asymmetric cryptographic schemes will break down. Hence, replacing them becomes mandatory. For this purpose, the National Institute of Standards and Technology (NIST) initiated a standardization process for post-quantum schemes. These schemes are supposed to substitute classical cryptography in different use-cases, such as client-server authentication during the TLS handshake. However, their signatures, public key sizes, and signature verification time impose difficulty, especially for resource-constrained devices. In this paper, we improve the TLS handshake performance relying on post-quantum signatures by combining the XMSS and the Dilithium signature schemes along the chain of certificates. We provide proof-of-concept implementation of our solution by integrating the two signature schemes in the WolfSSL library. Moreover, we evaluate the performance of our solution and establish that it reduces the signature verification time considerably and minimizes the size of the chain of trust. We provide a security proof of the proposed chain of trust which relies on the security of the XMSS scheme.

Keywords: Chain of Trust, Dilithium, Handshake Protocol, Post-quantum Cryptography, XMSS

1 Introduction

TLS security is based on public-key cryptography and provides connection integrity, confidentiality, and authenticity using X.509 certificates. X.509 are IETF-standard certificates used in digital authentication [18]. In a TLS connection, endpoints acting as servers (or clients) verify the communicating peer's identity and public key (PK) held by its certificate by leveraging a chain of certificates rooted to a pre-trusted root CA. We use the term chain of trust to refer to these chain of certificates throughout the paper. The signing algorithm used along with the chain of certificates is usually the same, such as Elliptic Curve Digital Signature (ECDSA) and RivestShamirAdleman (RSA).

With the advent of quantum computers, a soon-to-become physical reality, breaking the discrete logarithm and integer factorization cryptography has become increasingly possible as compared to traditional binary computers [45]. This calls for an urgent need to migrate toward post-quantum cryptography to avoid the insecurities owing to quantum computer-based attacks.

There is a large body of literature on post-quantum signature schemes and their families. Each of these families is based on its underlying cryptographic hardness assumptions. Among these, the most widely known schemes are based on error-correcting codes, lattices, multivariate systems, and cryptographic hash functions. The downside to these schemes is that they bear large signatures and public keys sizes (kilobytes to megabytes) compared to their classical counterparts e.g., RSA and ECDSA. Moreover, the higher the security level of these schemes,

the worst their performance regarding signing, verification time, signatures, and public keys size [42].

Using a pure post-quantum chain of certificates in the TLS handshake protocol leads to large chain's size. The signature and public key sizes of these signature schemes are in the order of kilobytes to megabytes, which is considered as the bottleneck of post-quantum authentication in TLS [46].

Our contribution: We summarize the key contributions of our work as follows:

- We propose a mixed post-quantum chain of trust that will substitute the pure classical one. Our new design combines two post-quantum signature schemes: XMSS and Dilithium.
- We provide a rigid security proof of the proposed chain of trust, relying on the security of the XMSS scheme.
- We prove that the proposed XMSS-based chain of trust ensures a better performance compared to the pure classical chain of certificates and the previous work [46], resulting in faster signature verification time and smaller public key and signature sizes, hence, a smaller chain of certificates.
- We enhance our solution by a proof-of-concept implementation integrated in the WolfSSL library.

Related Work In [36], Marzougui and Krämer evaluated the submissions to the NIST competition [2] with regard to their applicability to the most fundamental security use-cases of embedded systems. They identified XMSS [32] and qTESLA [10] signature schemes as the most fitting, and implemented them on ARM Cortex-R5 microprocessor-based development board. They also evaluated the performance of their implementation [36]. As a result, they suggested the use of hardware accelerator and the optimization of both algorithms to reduce their memory consumption.

In [22], Bürstinghaus-Steinbach et al. showed the integration of the post-quantum key encapsulation mechanism (KEM) scheme Kyber for key establishment and the post-quantum signature scheme SPHINCS+ into the embedded TLS library mbedTLS [1]. They measured the performance of these post-quantum primitives on four different embedded platforms and presented the challenges regarding large certificates chain and slow signing process during the TLS handshake protocol. They concluded that these challenges affect the use of embedded systems as TLS server but do not necessarily prevent them from acting as TLS client. While the writers of [22] recommended that the use of hardware accelerators may be considered to speed up the SPHINCS+ computations [22], no countermeasures were suggested for the huge size of the chain of trust.

Another possible improvement is the use of different post-quantum signature schemes in the same chain of certificates to improve the performance of post-quantum authentication in TLS as in [46]. A highly secure signing scheme is used in some certificates at the expense of low performance, while other certificate encapsulates a public key corresponding to an efficient signing scheme (at the cost of a low security level). By doing so, we reduce the duration of the endpoint's signing operation. In addition, the overall certificates' chain size is smaller than a pure certificate [46].

The idea of combining multiple post-quantum signature schemes came first in [17] to ensure the transition to post-quantum cryptography. Bindel et al. in [17] introduced *hybrid certificate* which includes two public keys for the subject, one classical and another post-quantum, and two

different Certificate authorities. They investigated the use of hybrid digital signature schemes, employed multiple methods for combining them, and framed unforgeable conditions for the resulting hybrid scheme [17]. This approach is, however, inefficient to tackle the problem of large chains of certificates.

A different way of combining multiple post-quantum signature schemes is called *mixed certificate* and is applied along a chain of certificates [46]. Certificates in this approach are signed with different signing algorithm. Sikeridis et al. were the first to introduce *mixed certificate* [46]. First, they investigated the impact of larger post-quantum chain of certificates and slower signing and verification on the throughput of the authenticating server. Then, they established that large certificates and signed messages are penalized by the Transmission Control Protocol (TCP) congestion window leading to extra Round-Trip Time (RTT)and higher TLS handshake latency. With slow signing, as they noticed, the server reaches the saturation point much earlier [46] deeming the process crucial for the server's performance. Sikeridis et al. concluded that the certificates chain size together with heavy signing and verification affects the TLS handshake time. To resolve the issue, they proposed a combination of the Dilithium [25] and the FALCON [26] signature schemes along the same chain of certificates.

This combination, however, has its own limitations. These certificates are bulky in size, especially for a higher security level. In addition, FALCON's signing is also CPU-intensive and slow, resulting in a handshake delay. The limitations become more apparent because the writers excluded certain certificate extensions which limits the applicability of their combination.

Organization We structure the paper the following sections. In Sec. 2, we introduce the TLS architecture. We give an overview on the integration of the two post-quantum signature schemes XMSS and Dilithium in the WolfSSL library in Sec. 3. In Sec. 4, we detail on the proposed XMSS-based chain of trust. In Sec. 5, we provide the security proof of the XMSS-based chain of trust. Sec. 6 assesses the performance of the designed chain of trust. Finally, we conclude this work by citing the advantages and the limitations of the proposed chain of trust.

2 Background

2.1 X.509 Certificate

Among the various public key certificates, X.509 is the most common Public Key Infrastructure (PKI) standard adopted by the IETF protocols (RFC5280 [18] and RFC6818 [49]) and is excessively employed in digital authentication for various protocols (e.g., TLS, SSH, IKEv2).

A *Certificate Authority* (CA) issues an entity's certificate. A digital certificate assures the entity's identity and ties the public key to it. In turn, the entity's identity is appended to the certificate in its subject field, while the public key is stored in the Subject Public Key Information (SPKI) along with the issuer's algorithm. A certificate comes with a specific validity period and extensions to enable additional functionality. Using the specified signature algorithm, the public key of an entity is signed by the CA's private key. The signature is then appended to the certificate's signature field.

During the session setup, the entities exchange certificates in order to verify the peer's identity. At the top of an X.509, are trusted CAs that self-sign their public keys known as root CA certificates. A root CA issues intermediate CAs (ICA). Subsequently, these intermediate CAs are used by the ICA to further issue and sign leaf certificates. This process creates a chain of certificates called a chain of trust. A chain of trust consists of two to four certificates but can be arbitrarily longer. In the final step of the chain verification, a leaf certificate is validated

by an endpoint if the endpoint trusts the chain's root CA, and the issuer's public keys verify all the issued signatures from the leaf to the root certificate.

2.2 Handshake Protocol

TLS consists of a record layer and a handshake layer. As shown in Fig. 1, the record layer is located above the transport layer and uses the Transmission Control Protocol (TCP). It uses symmetric cryptographic algorithms such as AES for bulk data encryption. To ensure a post-quantum security, AES-256 or higher security bits can be used [28]. The record layer is responsible for fragmentation, compression, and encryption of the sent messages. Above the record layer is the handshake layer that contains change cipher spec and alert. An application data protocol is located above the handshake layer. While the alert protocol is responsible for handling errors, the change cipher spec protocol indicates whether the cipher suites are used, in addition to changing the encryption keys during the handshake protocol.

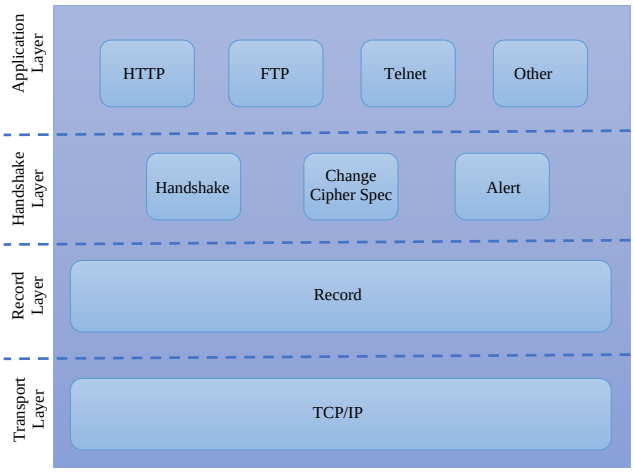

Figure 1: TLS Protocol Layers

The handshake protocol ensures confidentiality, integrity, and authenticity of the exchanged messages and takes place in a series of steps. First, the cryptographic primitives are negotiated between the client and the server. Then, the session keys are generated and then validated. Session keys are used later for the encryption of exchanged messages. After the client and the server are ready, the connection achieves a symmetric encryption.

Fig. 2 presents a high overview of the TLS handshake protocol steps. The handshake begins when the client starts a TLS connection by sending a *ClientHello* message. The message embeds the supported TLS version, cipher suites, and compression algorithms along with client random and optional data such as session ID or ticket for session resumption. This entire process is mediated by a cipher suite which describes these attributes: the key exchange cryptographic algorithm, signatures algorithm, encryption algorithm, message authentication, and the Pseudo Random Function (PRF) used to calculate keying material. For example, TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA256 is a cipher suite that uses ECDHE for the key exchange algorithm; ECDSA as the authentication algorithm; AES256 for the bulk data encryption algorithm, and SHA256 for the Message Authentication Code (MAC) algorithm). In return, the server answers with a *ServerHello* message which contains the chosen

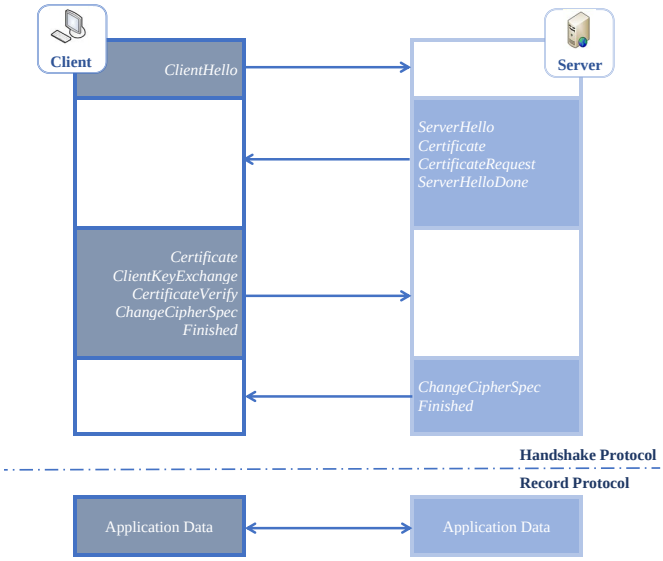

Figure 2: Handshake Protocol

TLS version, cipher suite, compression algorithm, and a server random. Along with it, it also sends a *Certificate* message containing the certificate with the server's public key and the entire chain of certificates up to a root certification authority (CA). At the moment of writing this paper, the adopted chain of certificates is pure, i.e., one signature algorithm is used along the chain of certificates. To ensure resistance against quantum computer attacks, classical signature schemes need to be replaced with post-quantum signature schemes.

In case Diffie-Hellman (DH) is used for key exchange, a *ServerKeyExchange* message is sent containing the server's ephemeral DH public key, a signature over the public DH key, client random, and server random. To ensure post-quantum security, in the same light, DH should be replaced by a post-quantum KEM (e.g., Kyber as in [22]).

Likewise, if mutual authentication is required, the server is expected to send a *CertificateRequest* message to which the client responds with a similar *Certificate* message. With the *ServerHelloDone* message, the server gives the client the signal to continue the handshake. The client sends a *ClientKeyExchange* message. As we are interested only in building a secure chain of trust used for client and server authentication, we will not investigate the choice of KEM.

In case of mutual authentication, the client sends first a client *Certificate* message and a *CertificateVerify* message. This message holds a digital signature over all handshake messages sent or received, starting at *ClientHello* and up to, but excluding this message. If the client wants to switch the negotiated algorithms, he sends a *ChangeCipherSpec* message to the server (as shown in Fig. 2). Second, the *Finished* message completes the handshake. Then, the server answers also with a *ChangeCipherSpec* protocol message and a *Finished* message. Finally, application data can be securely exchanged.

2.3 Post-quantum Families

TLS makes heavy use of classical asymmetric algorithms in the handshake protocol for signing and key exchange (e.g., RSA, ECDSA). To ensure a quantum-resistant connection, these

algorithms have to be exchanged by post-quantum schemes.

Currently, there are four promising families of mathematical objects and schemes being discussed in the field of post-quantum cryptography: lattice, code, hash, and multivariate families. Each family's security relies on its underlying cryptographic hardness assumptions.

Lattice-based Cryptography The National Institute of Standards and Technology (NIST), initiated a standardization process of post-quantum cryptographic schemes [2]. Since five out of seven finalists of the NIST post-quantum cryptography standardization process are lattice-based schemes, lattice-based cryptography can be regarded as currently the most relevant family of post-quantum cryptography.

Lattice-based cryptography was introduced by Ajtai [8]. A lattice is defined as the set of all integer linear combinations of linearly independent vectors in real n-space $\mathbf{R}^n$. There exist many lattice-related NP-hard problems used for cryptographic purposes, namely the Shortest Vector Problem (SVP), the Closest Problem (CVP), and Learning with Errors (LWE). While the shortest vector problem (SVP) pertains to finding a shortest non-zero vector in the Euclidean norm, the closest vector problem (CVP) is solved by finding a lattice vector that minimizes the distance from another target lattice. The security of lattice-based signatures submitted to NIST relies on the Learning with Errors problem (LWE). The LWE instance contains the secret vectors blinded with a noise vector (error) which usually are taken from a Gaussian distribution. This makes them expensive in terms of execution time [41].

Nevertheless, lattice-based cryptography offers highly efficient schemes compared to other post-quantum families. Signature schemes based on lattice problem hardness are divided into two categories: hash-and-sign and Schnorr-based schemes. Hash-and-sign based schemes, such as GPV [27], qTESLA [10], and Dilithium [25], are based on pre-image trapdoor functions [39]. These schemes have been continuously improved to reduce the execution time of the key generation, signing, and verification algorithms [39] and to shrink the signature size [14]. The second category of lattice-based signature schemes, for example Falcon [26] and pqNTRUSign [29], are based on the same ideas as Schnorr signature schemes [34, 35].

The first generation of both classes of lattice-based schemes is based on standard lattices, i.e., matrix and vector operations, which are costly in terms of execution time and memory consumption (in the order of megabytes). More recent lattice-based schemes employ ring analogies of standard lattices that use polynomial representation rather than matrices and vectors.This representation significantly reduces the execution time and the memory consumption [9].

Hash-based Cryptography Hash-based schemes are considered to be the most mature schemes for post-quantum digital signatures [21]. They were first introduced by Lamport [33], and developed further [37, 24] to result in schemes such XSMSS and SPHINCS+ [21]. The security of these schemes, improved over the last decades, relies on cryptographic hash functions considered secure against quantum computer-based attacks for appropriately chosen security parameters. There are two categories of hash-based signatures, namely the stateful and stateless schemes.

– Stateful hash-based signature schemes: These schemes are built from one-time signature (OTS) schemes such as [33, 37, 24] with multiple OTS signing keys. The signer has to keep track of already-used OTS keys to avoid using them more than once. This state management requirement is considered a significant disadvantage. The public keys corresponding to the OTS private keys are typically represented as leaves of a Merkle hash tree. A signature consists of an OTS, the OTS public key, and the values (i.e., hash

digests) of the intermediate nodes from the OTS public key to the root of the tree. This approach allows verifying every OTS using the root of the hash tree as public key. XMSS [30] and XMSSMT [32] are the examples of stateful hash-based schemes.

– Stateless hash-based signature schemes: These schemes do not require the signer to maintain any state (e.g., WOTS [19], HORST [12], and SPHINCS [16]). They are typically built upon Few-Time Signatures (FTS) that allow signing multiple messages with the same signing key. However, these schemes gradually leak information on the signing key with each signature generated using this key [16].

  Stateless hash-based signature schemes have the highest signature verification performance among all post-quantum signature schemes [36]. Their public keys size is tiny (around 60 bytes), but the size of their signatures is in the range of 40 kilobytes [46].

3 Post-quantum Integration in the wolfSSL Library

To perform a post-quantum TLS handshake, we integrated the reference implementations of the post-quantum signature algorithms XMSS and Dilithium into the wolfSSL library [4]. The wolfSSL is open source under the GPLv2 license for non-commercial use. There are three divisions in the wolfSSL library: cryptographic primitives, TLS protocol, and tools described in the following subsections. The code for our XMSS-based chain of trust can be found at `https://github.com/Soundes-M/Soundes-M-XMSS_based_chain_of_trust`.

3.1 Cryptographic Primitives

The wolfSSL library is written in C and is backed by a C-language-based embedded cryptography engine WolfCrypt [3]. The WolfCrypt library offers not just low memory usage and optimized speed, but also high portability. It capsules cryptographic algorithms into modules grouped into four categories: symmetric encryption algorithms, hash functions, random number generators, and public key algorithms. At the time of framing this paper, the wolfSSL library contains only the post-quantum cryptographic scheme NTRU. We encapsulated XMSS and Dilithium each in its own module using the algorithm's reference code as a base for the implementation. The reference implementation of XMSS offers to choose parameters like the hash function during compilation. We support the variant `XMSS-SHA2_10_256`. We changed the SHA2-256 function calls in the XMSS reference implementation from OpenSSL to the wolfSSL.

Since Dilithium requires the hash function SHAKE-256 which is not part of the WolfSSL library, we kept the reference implementation of Dilithium.

3.2 TLS Protocol

In this paper, we focus on TLS connections with server authentication only. In a TLS connection, we identify three messages of particular interest: *Certificate*, *ServerKeyExchange*, and *ClientKeyExchange*. The *Certificate* message contains the chain of certificates. These certificates allow the client to verify the legitimacy of the server's public key by validating all the certificates up to the root. The verification process is more detailed in Sec.4. We outlined an ASN1-based structure to define the new CA and ICA X.509 certificates. A CA is a self-signed certificate using a Dilithium secret key. Additionally, two ICAs are used. One contains a Dilithium public key but is signed by an XMSS secret key, and the other contains the XMSS root node signed with a Dilithium secret key.

Once the chain of certificates is verified, the post-quantum KEM public key is generated and pasted into the *ServerKeyExchange*. Subsequently, the key exchange data is signed with the server's Dilithium public key. The signature is, therefore, added to the *ServerKeyExchange* message and sent to the client. The client, after receiving the *ServerKeyExchange* message, verifies the Dilithium signature and generates its own post-quantum KEM public key response. In our test scenario, the client key exchange is unauthenticated.

3.3 Tools

We added some extensions to existing tools accompanying the wolfSSL library and to its configuration. All changes we made to the library can be controlled through the config.h file during compilation. Each module of the cryptographic primitives can be activated or deactivated separately. The XMSS and Dilithium must be activated by setting `--enable-XMSS` and `--enable-DILITHIUM` respectively. But given that the XMSS algorithm employs the hash function defined in the wolfSSL library, `--enable-opensslextra` must be set when activating it.

4 Description of the XMSS-based Chain of Trust

4.1 XMSS Signature Scheme

Like the Merkle signature scheme [38], XMSS uses a one-time signature scheme (OTS) that can only sign one message with one key. To accommodate this limitation, a hash tree is used as shown in Fig. 3. The use of a hash tree reduces the authenticity of many OTS verification keys to one public XMSS key. To minimize storage requirements, pseudo-random generators (PRG) are used. Since the XMSS signature scheme is based on a hash tree, it can be used to sign a limited number of messages with one public key pk. Note that the number of possible messages to sign N must be a power of two.

Key Generation: Initially, N key pairs of a Winternitz OTS [24]) are generated (X_i, Y_i). For each $1 \leq i \leq 2^h$, a hash value of the public key $H(Y_i)$ is computed.

The hash values $H(Y_i)$ constitute the leaves of the hash tree. By placing these leaves and recursively hashing them, we form a binary tree. Let $a_{i,j}$ denote the node in the tree with height i and left-right position j. Then, the hash values $H(Y_i) = a_{0,i}$ form the leaves of the tree. For each node, we choose two bitmasks α and β uniformly at random, then calculate each tree node value by hashing the concatenation of its two children, xor-ed with random bitmasks (e.g., $a_{1,2} = H((a_{0,4} \oplus \alpha) \| (a_{0,5} \oplus \beta))$.

The private key of the XMSS signature scheme scales linearly with the number of messages to be sent. However, in our scenario, the secret key size does not impose a difficulty as it is not transferred during the TLS connection.

The public key pk is the root of the tree (A_3 in Fig. 3), which can be built by concatenating the hashes of the individual public keys Y_i. These individual public keys can also be made public without breaking security.

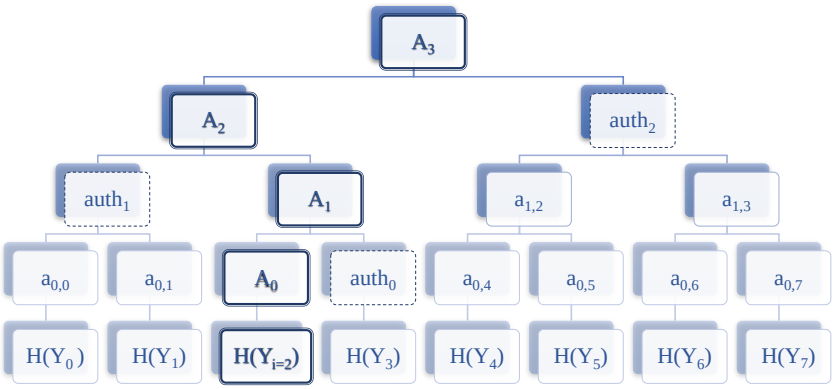

Figure 3: XMSS Hash Tree

Signing: The signer determines whether the key pair has been previously used to sign a message and if so, chooses an unused pair (X_i, Y_i). It then uses the Winternitz OTS scheme to sign a message M, resulting in a signature *sig'* and corresponding public key Y_i. To prove the legitimacy of the (X_i, Y_i) pair, the signer includes intermediate nodes of the XMSS hash tree (e.g., in Fig. 3 A_0, A_1, A_2) so that the verifier verifies if $H(Y_i)$ was used to compute the public key $a_{n,0} = A_n$ at the root of the tree.

To be able to calculate the node A_{i+1}, a verifier needs to know the other child of A_{i+1}, the sibling node of A_i. We call this node $auth_i$, so that $A_{i+1} = H((A_i \oplus \alpha) \| (auth_i \oplus \beta))$ (with α and β are maskbits chosen randomly at uniform). Hence, n nodes $auth_0, \ldots, auth_{n-1}$ are needed, to reconstruct $A_n = a_{n,0} = pk$ from $A_0 = a_{0,i}$. We present an example of an authentication path in Fig. 3.

These nodes $auth_0, \ldots, , auth_{n-1}$, the Y_i, and the one-time signature sig', together constitute a signature of M using the XMSS signature scheme:$sig = (sig' \| Y_i \| auth_0 \| auth_1 \| \ldots \| auth_{n-1})$.

Signature Verification: With the knowledge of the public key pk, the message M, and the signature $sig = (sig' \| Y_i \| auth_0 \| auth_1 \| \ldots \| auth_{n-1})$, the verifier first verifies the Winternitz OTS sig' of the message M using the Winternitz OTS public key Y_i. If sig' is a valid signature of M, the verifier computes $A_0 = H(Y_i)$ by hashing the OTS public key. The nodes of A_j of the path are computed with $j = 1, \ldots, n-1$. If the obtained result equals the public key pk of the XMSS signature scheme, the signature is considered valid.

4.2 Dilithium Signature Scheme

The Dilithium signature scheme is based on the Fiat-Shamir paradigm [34]. It can also be seen as a variant of the Bai-Galbraith scheme (BG) [13].

Dilithium is known as one of the most promising post-quantum signatures submitted to the NIST competition [23]. At the moment of writing this paper, it has reached the third round of the NIST competition. The Dilithium scheme is known for its good performance because of its key size and signature verification time (i.e., Dilithium-II has a public key and signature size of 1312 and 2420 bytes respectively; its signing process takes 194.892 cycles and its signature verification takes 72.663 cycles on a Skylake CPU (AVX implemented) [5].

Dilithium's better performance can be owed to different reasons. First, Dilithium is instantiated with Module-LWE which deals with matrix of 'small' polynomials A instead of a unique polynomial A (as in Ring-LWE). Module-LWE addresses the limitation of R-LWE: the size of polynomials increases with security. For Module, only the number of rows and columns (noted k and l respectively in the official submission of Dilithium [22]) impacts security, not the polynomial size (256 coefficients), which is set the same for all.

Another reason is Dilithium's compression mechanisms. The compression is done in two ways. First, the sampling of A is produced with XOF function (Extendable Output Function), which generates a (deterministic) pseudo-random string from a small seed. Therefore, the public key contains the seed instead of the polynomial A. Another compression is a per-coefficient truncation (or rounding), associated with a correcting code mechanism to guess truncated bits.

4.3 XMSS-based Chain of Trust

In the existing literature, a chain of trust refers to a hierarchy of certificates used to verify the validity of a certificate's issuer. Each certificate in the chain of trust is issued and signed by a certificate that lives higher up in the hierarchy. The certificate at the top of the hierarchy (called a trust anchor) is a self-signed certificate.

In this section, we draw our attention to building a new chain of trust by combining two post-quantum algorithms: XMSS and Dilithium. A high overview of the XMSS-based chain of trust is given in Fig. 4. The validation of the chain of trust from the bottom to the top is outlined in the steps below.

1. During the handshake protocol, the client receives the *Certificate* message from the server. The *Certificate* message contains an XMSS-based chain of trust (as in Fig. 4). The ICA of the chain (in blue) are XMSS certificates. These certificates contain Dilithium public keys (i.e., a Dilithium public key signed with XMSS private key and stored in an XMSS certificate). The signed Dilithium keys are verified in theses 2 steps:
   (a) The client verifies the Winternitz One-Time signatures using the XMSS public keys. These signatures are independent, and their verification can be parallelized to reduce the time to $\frac{1}{m}$ of the total sequential verification time, with m being the number of certificates (e.g., four certificates in Fig. 4).
   (b) Then, the client computes the path through the XMSS hash tree and finally compares it with the XMSS root node (see Fig. 4). If the values are equal, the verification of the chain of trust proceeds to step 2. Otherwise, the verification fails, and the chain of trust is broken.
2. To verify the signed root node of the XMSS hash tree (II), the client uses the Dilithium public key of the root certificate (I) as shown in Fig. 4, which is a Dilithium self-signed CA certificate. This step ensures the integrity of the XMSS root node.

In our XMSS-based chain of trust, we do not keep a state on the used XMSS keys because the Dilithium keys are signed once and do not change often. In case of certificate revocation, the whole XMSS hash tree should be regenerated.

Usually, a bigger tree height leads to an exponential growth in the run time of key generation [48]. As the number of leaves (the number of chain certificates) is small, regenerating the XMSS hash tree is not time-consuming. In case of revocation, the root CA generates the hash tree and signs the root node of the XMSS hash tree and the Dilithium public keys of the trusted endpoint. Therefore, we believe that when using the XMSS chain of trust, one should consider the overload on the root CA.

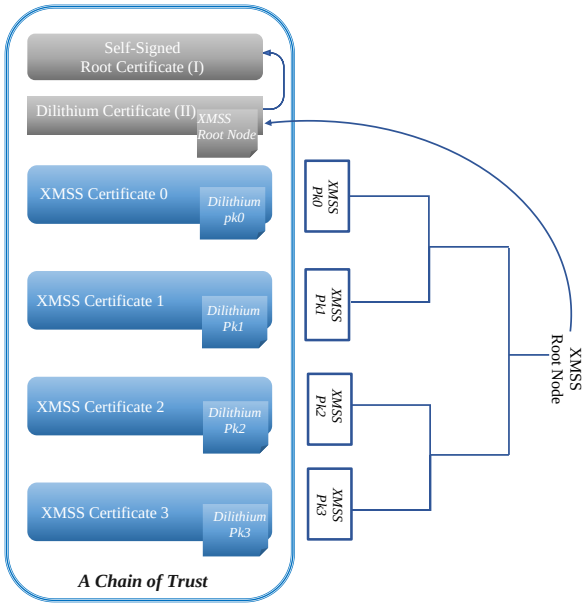

Figure 4: A Chain of Trust based on XMSS and Dilithium

5 Security Proof

The XMSS-based chain of trust inherits its security from the XMSS scheme. In essence, forging the XMSS signatures breaks the chain of trust described in Sec. 4.3. The security of XMSS is based on two functions: the *Hash function* H (H is second preimage resistant) and the *pseudorandom function* F. To capture this formally, we follow Theorem.1 from [20].

Theorem 1. *If H_n is a second preimage resistant hash function family and F_n a pseudorandom function family, then XMSS is existentially unforgeable under chosen message attacks.*

In this paper, we employ XMSS from RFC 8391 [31], that defines different parameter sets with $n = 32$ (n being the length of the hash in bytes) to provide a classical security level of 256 bits. A setting with $n = 64$ provides a classical security level of 512 bits. When considering quantum-computer-aided attacks, these output sizes yield post-quantum security of 128 and 256 bits respectively [31]. SHA2 is used to instantiate the H and F functions. For $n = 32$ setting, XMSS employs SHA2-256, and for $n = 64$, SHA2-512 is used.

An XMSS-based chain of trust is valid only if all the signatures are valid. Therefore, breaking the chain of trust requires breaking all the signatures of the chain.

This issue was introduced by Anderson in [11] under the name of *forward security* for signature schemes (FSSIG), later formalized in [15]. It states that even if a key is compromised, all signatures created before remain valid.

Theorem. 2 [20] proves that XMSS is FSSIG.

Theorem 2. *If H_n is a second preimage resistant hash function family and F_n a pseudorandom function family, then XMSS is a forward-secure digital signature scheme.*

If an attacker learns the actual secret key X_i, she is still not be able to forge a signature under a secret key X_j, $j < i$. This is a desirable property, especially in the context of chain of

trust. For example, the attacker who forges the signature of the XMSS Certificate 3 (Fig. 4) and gets the corresponding secret key will not be able to reveal the other XMSS secret keys.

6 Performance Evaluation

6.1 Platform Description

Our evaluations so far have only covered the cryptographic primitives. We therefore created a test setup to assess the verification time of the chain of trust and its size. The test setup was composed of a 64-bit Ubuntu desktop PC and an Intel Whiskey Lake running on 1.8 GHz, with 16 GB RAM. We adopted the WolfSSL [4] version 4.5.0, which supports TLS1.3. The desktop PC acted as a client. We further investigated the server-authenticated TLS connections only. Hence, the evaluation of the chain of trust takes place on the client side.

6.2 Signature Verification Time

When using an XMSS-based chain of trust during handshake protocol, the verification of all WOT XMSS signatures can be executed in parallel as described in Sec.4.3. Then, step (b) was done by calculating the paths to the XMSS root node and comparing them to the signed root node of the XMSS hash tree. As the run time of the XMSS signature verification is only linearly impacted by h (h is the height of the XMSS hash tree) [48] and h is considerably small, the verification is faster. In Tab. 1, we show a comparison of signatures verification time when using different chain of trust (i.e., pure classical certificates, mixed post-quantum certificates combining FALCON1024 and SPHINCS+ SHA256-128f-simple as in [46], and XMSS-based chain of trust).

We compared our results to two pure classical chains of trust. For RSA, we used 3072-bit version that provides 128 bits of security. While for ECDSA, we utilized the `secp384r1` curve which offers 192 bits.

We opted for a higher security level for ECC signatures (i.e., 192 instead of 128) because these primitives will be broken by a quantum computers before their equivalent security level big number RSA signatures. In other words, both RSA3072 and ECDSA256 would offer 0 bits of security in a post-quantum setting. However, the amount of time needed to break ECDSA256 is smaller [7]. By choosing ECDSA348, we guarantee that both schemes need similar period of time to be broken by a quantum computer.

We present the results of the previous work [46] in Tab. 1, where Sikeridis et al. combined the two post-quantum signature schemes SPHINCS+ and FALCON along a chain of trust. They picked SPHINCS-SHA256-128f-simple for integration and evaluation as it is the most efficient among the SPHINCS+ variants [46]. SPHINCS-SHA256-128f-simple provides 64-bit security in a post-quantum setting. Sikeridis et al. chose the variant FALCON1024 provided by the Falcon inventors in the liboqs library [6, 47]. FALCON1024 provides 230-bit security in a post-quantum setting. The adopted version does not include the floating point hardware optimizations that could have improved FALCONS's signing performance by 20 times [40].

6.3 Chain size

To quantify the differences between the sizes of the chains of trust, we use Tab.2. The results presented summarize similar Distinguished Encoding Rules (DER)-encoded certificates' chain sizes excluding the root CA certificate. When a certificates' chain exceeds the maximum length,

Chain of Turst	Signature Verification (ms)
Our work	1.9
Pure RSA 3072	0.12
Pure ECDSA 384	2.10
FALCON and SPHINCS+ [46]	4.3

Table 1: Comparison of Signature Verification Time (two ICA)

TLS utilizes record fragmentation, and in turn, the record layer sends the certificate in chunks of 2^{14} bytes [43].

An XMSS signature size is calculated through $4 + n(p + h + 1)$, where p is the number of Winternitz chains used in a single OTS operation, h the height of the XMSS hash tree, and n the length of the hash (in bytes). The values p, h, and n influence the size of the signature. Larger values imply larger signature sizes [48].

Given that the number of certificates constituting the chain of trust is small, and n being fixed to 32 to ensure 128-bit security against computer-aided attacks, the signature size is relatively small (in the order of 2.5 MB). In addition, our chain of trust does not necessitate a state management; hence, we do not need additional fields in the certificate.

Chain of Trust	Server's Certificate size (KB)
Our work	4.096
Pure RSA 3072	2.44
Pure ECDSA 384	2.15
FALCON and SPHINCS+ [46]	9.89

Table 2: Comparison of Signatures and chains' sizes (two ICA)

7 Conclusion

In this paper, we introduced a cutting-edge solution for the integration of the post-quantum signature schemes XMSS and Dilithium along a chain of certificates used during the TLS handshake protocol. Moreover, we enhanced our solution by a proof-of-concept implementation and integrated the XMSS and the Dilithium schemes in the wolfSSL library [4]. We evaluated the performance of a TLS variant using our chain of trust by comparing the signatures verification and the chain size to the results of previous work [46].

Our solution offers a better performance owing to the comparable performance of XMSS to that of hash functions.

We can easily tune the security of our solution to reach higher security levels by leveraging the security parameters of the hash function used to hash the leaves of the XMSS tree. For example, `XMSS-SHA2-10-256` ($w = 16$, w is the Winternitz parameter) provides 196 bits of security [20]. Another advantage of our solution is that hardware accelerators can be used to speed up the signing and verification processes. Hence, running the chain's validation on a source-constrained device may be possible e.g., EFM32 Gecko 32-bit micro-controller that comes with a cryptography accelerator supporting SHA-2. As stated in Sec. 4.3, the verification process can also be parallelized.

By combining XMSS and Dilithium, we enable high-security enhanced performance for the signature along the chain of certificate. We witness this boost in performance by employing

the Dilithium key, one of the finalists in the Post Quantum Cryptography Standardization and known to perform better than other candidates [2], in the endpoints to sign the KEM public key. By using Dilithium, we minimize the signing time and the chain size as compared to the previous work [46].

However, a limitation of our work is the overload applied to the root CA. The CA should manage the chain of trust for each endpoint separately; thus it may be potentially unreachable. This will result in exhausting the root CA, incapacitated to handle incoming requests.

Future work will address the root CA overload challenge and the evaluation of a complete handshake protocol that uses an optimized version of our XMSS-based chain of trust.

Acknowledgment

The work described in this paper has been supported by the German Federal Ministry of Education and Research (BMBF) under the project Full Lifecycle Post-Quantum PKI - FLOQI (ID 16KIS1074).

References

[1] Mbedtls library website. https://tls.mbed.org/. [Online; accessed 9-September-2021].

[2] NIST Post-Quantum Cryptography Standardization website. https://csrc.nist.gov/Projects/post-quantum-cryptography/. [Online; accessed 9-September-2021].

[3] Wolfcrypt library website. https://www.wolfssl.com/products/wolfcrypt/. [Online; accessed 9-September-2021].

[4] Wolfssl library website. https://www.wolfssl.com/. [Online; accessed 9-September-2021].

[5] Dilithium website. https://pq-crystals.org/dilithium/index.shtml, 2020. [Online; accessed 9-September-2021].

[6] Liboqs website. https://github.com/open-quantum-safe/liboqs, 2020. [Online; accessed 9-September-2021].

[7] The Search for Quantum Resistant Cryptography. https://sectigo.com/uploads/resources/Quantum-Resistance-Whitepaper.pdf, September 2019. [Online; accessed 9-September-2021].

[8] M. Ajtai. Generating hard instances of lattice problems (extended abstract). pages 99–108, 1996.

[9] Sedat Akleylek, Nina Bindel, Johannes Buchmann, Juliane Krämer, and Giorgia Azzurra Marson. An efficient lattice-based signature scheme with provably secure instantiation. In David Pointcheval, Abderrahmane Nitaj, and Tajjeeddine Rachidi, editors, *Progress in Cryptology – AFRICACRYPT 2016*, pages 44–60, Cham, 2016. Springer International Publishing.

[10] Erdem Alkim, Paulo S. L. M. Barreto, Nina Bindel, Juliane Kramer, Patrick Longa, and Jefferson E. Ricardini. The lattice-based digital signature scheme qtesla. Cryptology ePrint Archive, Report 2019/085, 2019. https://eprint.iacr.org/2019/085.

[11] Ross Anderson. Two remarks on public key cryptology. 10 2000.

[12] Jean-Philippe Aumasson and Guillaume Endignoux. Improving stateless hash-based signatures. In *Cryptographers' Track at the RSA Conference*, pages 219–242. Springer International Publishing, 2018.

[13] Shi Bai and Steven D. Galbraith. An improved compression technique for signatures based on learning with errors. Cryptology ePrint Archive, Report 2013/838, 2013. https://eprint.iacr.org/2013/838.

[14] Rachid El Bansarkhani and Johannes Buchmann. Lcpr: High performance compression algorithm for lattice-based signatures. Cryptology ePrint Archive, Report 2014/334, 2014. https://eprint.iacr.org/2014/334.

[15] Mihir Bellare and Sara K. Miner. A forward-secure digital signature scheme. In Michael Wiener, editor, *Advances in Cryptology — CRYPTO' 99*, pages 431–448, Berlin, Heidelberg, 1999. Springer Berlin Heidelberg.

[16] Daniel J. Bernstein, Daira Hopwood, Andreas Hülsing, Tanja Lange, Ruben Niederhagen, Louiza Papachristodoulou, Michael Schneider, Peter Schwabe, and Zooko Wilcox-O'Hearn. Sphincs: Practical stateless hash-based signatures. In Elisabeth Oswald and Marc Fischlin, editors, *Advances in Cryptology – EUROCRYPT 2015*, pages 368–397, Berlin, Heidelberg, 2015. Springer Berlin Heidelberg.

[17] Nina Bindel, Udyani Herath, Matthew McKague, and Douglas Stebila. Transitioning to a quantum-resistant public key infrastructure. Cryptology ePrint Archive, Report 2017/460, 2017. https://eprint.iacr.org/2017/460.

[18] Sharon Boeyen, Stefan Santesson, Tim Polk, Russ Housley, Stephen Farrell, and Dave Cooper. Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile. RFC 5280, May 2008.

[19] Johannes Buchmann, Erik Dahmen, Sarah Ereth, Andreas Hülsing, and Markus Rückert. On the security of the winternitz one-time signature scheme. In Abderrahmane Nitaj and David Pointcheval, editors, *Progress in Cryptology – AFRICACRYPT 2011*, pages 363–378, Berlin, Heidelberg, 2011. Springer Berlin Heidelberg.

[20] Johannes Buchmann, Erik Dahmen, and Andreas Hülsing. Xmss - a practical forward secure signature scheme based on minimal security assumptions. In Bo-Yin Yang, editor, *Post-Quantum Cryptography*, pages 117–129, Berlin, Heidelberg, 2011. Springer Berlin Heidelberg.

[21] Johannes A. Buchmann, Denis Butin, Florian Göpfert, and Albrecht Petzoldt. *Post-Quantum Cryptography: State of the Art*, pages 88–108. Springer Berlin Heidelberg, Berlin, Heidelberg, 2016.

[22] Kevin Bürstinghaus-Steinbach, Christoph Krauß, Ruben Niederhagen, and Michael Schneider. Post-quantum tls on embedded systems: Integrating and evaluating kyber and sphincs+ with mbed tls. ASIA CCS '20, page 841–852, New York, NY, USA, 2020. Association for Computing Machinery.

[23] NIST Information Technology Laboratory Computer Security Resource Center. NIST Standardization round. https://csrc.nist.gov/projects/post-quantum-cryptography, 2020. [Online; accessed 9-September-2021].

[24] Chris Dods, Nigel Smart, and Martijn Stam. Hash based digital signature schemes. In *Cryptography and Coding - IMACC 2005*, volume 3796, pages 96 – 115, Germany, November 2005. Springer Berlin Heidelberg. Conference Proceedings/Title of Journal: Cryptography and Coding, Springer LNCS 3796.

[25] L. Ducas, Eike Kiltz, Tancrède Lepoint, Vadim Lyubashevsky, P. Schwabe, Gregor Seiler, and D. Stehlé. Crystals-dilithium: A lattice-based digital signature scheme. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2018:238–268, 2018.

[26] Pierre-Alain Fouque, Jeffrey Hoffstein, Paul Kirchner, Vadim Lyubashevsky, Thomas Pornin, Thomas Prest, Thomas Ricosset, Gregor Seiler, William Whyte, and Zhenfei Zhang. Falcon: Fast-fourier lattice-based compact signatures over ntru. *Submission to the NIST's post-quantum cryptography standardization process*, 36, 2018.

[27] Craig Gentry, Chris Peikert, and Vinod Vaikuntanathan. Trapdoors for hard lattices and new cryptographic constructions. Cryptology ePrint Archive, Report 2007/432, 2007. https://eprint.iacr.org/2007/432.

[28] Markus Grassl, Brandon Langenberg, Martin Roetteler, and Rainer Steinwandt. Applying grover's algorithm to aes: Quantum resource estimates. pages 29–43, 02 2016.

[29] Jeffrey Hoffstein, Nick Howgrave-Graham, Jill Pipher, Joseph H. Silverman, and William Whyte. Ntrusign: Digital signatures using the ntru lattice. In Marc Joye, editor, *Topics in Cryptology — CT-RSA 2003*, pages 122–140, Berlin, Heidelberg, 2003. Springer Berlin Heidelberg.

[30] Andreas Huelsing, Denis Butin, Stefan-Lukas Gazdag, Joost Rijneveld, and Aziz Mohaisen. Xmss: extended merkle signature scheme. [Online; accessed 9-September-2021].

[31] Andreas Huelsing, Denis Butin, Stefan-Lukas Gazdag, Joost Rijneveld, and Aziz Mohaisen. XMSS: eXtended Merkle Signature Scheme. RFC 8391, May 2018.

[32] Andreas Hülsing, Lea Rausch, and Johannes Buchmann. Optimal Parameters for XMSSMT. In Alfredo Cuzzocrea, Christian Kittl, Dimitris E. Simos, Edgar Weippl, and Lida Xu, editors, *Security Engineering and Intelligence Informatics*, pages 194–208. Springer Berlin Heidelberg, 2013.

[33] Leslie Lamport. Constructing digital signatures from a one way function. Technical Report CSL-98, October 1979. This paper was published by IEEE in the Proceedings of HICSS-43 in January, 2010.

[34] Vadim Lyubashevsky. Fiat-shamir with aborts: Applications to lattice and factoring-based signatures. In Mitsuru Matsui, editor, *Advances in Cryptology – ASIACRYPT 2009*, pages 598–616, Berlin, Heidelberg, 2009. Springer Berlin Heidelberg.

[35] Vadim Lyubashevsky. Lattice signatures without trapdoors. pages 738–755, 2012.

[36] Soundes Marzougui and Juliane Krämer. Post-quantum cryptography in embedded systems. ARES '19, New York, NY, USA, 2019. Association for Computing Machinery.

[37] Ralph C. Merkle. A certified digital signature. In Gilles Brassard, editor, *Advances in Cryptology — CRYPTO' 89 Proceedings*, pages 218–238, New York, NY, 1990. Springer New York.

[38] Ralph C. Merkle. A certified digital signature. In Gilles Brassard, editor, *Advances in Cryptology — CRYPTO' 89 Proceedings*, pages 218–238, New York, NY, 1990. Springer New York.

[39] Daniele Micciancio and Chris Peikert. Trapdoors for lattices: Simpler, tighter, faster, smaller. Cryptology ePrint Archive, Report 2011/501, 2011. https://eprint.iacr.org/2011/501.

[40] Thomas Pornin. New efficient, constant-time implementations of falcon. Cryptology ePrint Archive, Report 2019/893, 2019. https://ia.cr/2019/893.

[41] Thomas Prest. *Gaussian sampling in lattice-based cryptography.* PhD thesis, Ecole normale supérieure-ENS PARIS, 2015.

[42] Manohar Raavi, Simeon Wuthier, Pranav Chandramouli, Yaroslav Balytskyi, Xiaobo Zhou, and Sang-Yoon Chang. Security comparisons and performance analyses of post-quantum signature algorithms. In Kazue Sako and Nils Ole Tippenhauer, editors, *Applied Cryptography and Network Security*, pages 424–447, Cham, 2021. Springer International Publishing.

[43] Eric Rescorla. The Transport Layer Security (TLS) Protocol Version 1.3. RFC 8446, August 2018.

[44] Peter W. Shor. Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. *SIAM J. Comput.*, 26(5):1484–1509, October 1997.

[45] Peter W. Shor. Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. *SIAM J. Comput.*, 26(5):1484–1509, October 1997.

[46] Dimitrios Sikeridis, Panos Kampanakis, and M. Devetsikiotis. Post-quantum authentication in tls 1.3: A performance study. *IACR Cryptol. ePrint Arch.*, 2020:71, 2020.

[47] D. Stebila and M. Mosca. Post-quantum key exchange for the internet and the open quantum safe project. In *SAC*, 2016.

[48] W. Wang, Bernhard Jungk, Julian Wälde, S. Deng, Naina Gupta, Jakub Szefer, and Ruben Niederhagen. Xmss and embedded systems - xmss hardware accelerators for risc-v. *IACR Cryptol. ePrint Arch.*, 2018:1225, 2018.

[49] Peter E. Yee. Updates to the Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile. RFC 6818, January 2013.