# SoK: The Engineer's Guide to Post-Quantum Cryptography for Embedded Devices

Maximilian Pursche*†, Nikolai Puch*†, Sebastian N. Peters*†, Michael P. Heinl*†‡
* *Technical University of Munich; TUM School of Computation, Information and Technology; Department of Computer Engineering; Chair of IT Security*
† *Fraunhofer AISEC; Department Product Protection and Industrial Security*
‡ *Munich University of Applied Sciences HM; Department of Computer Science and Mathematics*

***Abstract*—Embedded systems are flexible and cost-effective and thus have found a use case in almost every part of our daily lives. Due to their widespread use, they have also become valuable targets for cyber attacks. However, translating cutting-edge cyber security from servers and desktops to the embedded realm can be challenging due to the limited computational power and memory of embedded devices. Although quantum computing is still in early research and development, it threatens to break conventional asymmetric cryptography which is a key component of most secure applications currently in use. Given the long lifespan of embedded devices, which can last for decades, research must find solutions for post-quantum (PQ) security rather sooner than later. The field of post-quantum cryptography (PQC) received significant attention in 2019 when the National Institute for Standards and Technology (NIST) launched a competition to find suitable PQC algorithms. During the PQC competition, the applicability of novel PQC algorithms to embedded devices was an important topic that garnered significant research interest. We provide a survey of the latest research regarding PQC for embedded systems. However, rather than focusing on PQC algorithms, our study revolves around practical use cases intending to help embedded developers understand the current state of research from an integration perspective.**

## 1. Introduction

Embedded platforms are extensively used throughout today's technology driven society and can be found in a variety of environments ranging from personal smart home devices to industrial Operational Technology. According to estimates, there are already 16 billion Internet of Things (IoT) devices deployed worldwide, and the number is growing [1]. Despite their functional suitability for many use cases and environments, their security is often inadequate. Due to their long lifespan, limitation to few core tasks, and price pressure, embedded platforms often do not receive the same level of attention regarding security from both users and developers. As a result, they are often vulnerable [2], as demonstrated by multiple large-scale attacks on IoT devices in recent years, such as the Mirai botnet [3]. Thus, although these devices have limited computational capabilities and memory constraints, security must become a priority. One of the industry's major security challenges is the threat of a high-performance quantum computer becoming available during the long lifespan of embedded devices. Due to Shor's algorithm [4], conventional asymmetric cryptography is broken once a powerful enough quantum computer exists. Motivated by recent advances by IBM [5] and Google [6], security research focuses on finding corresponding countermeasures. Post-Quantum Cryptography (PQC) is the field of research that tries to find mathematical problems that can withstand quantum cryptanalysis while remaining usable on conventional computers. One of the cornerstones of this research is the PQC competition [7] held by the National Institute of Standards and Technology (NIST) which ultimately aims to standardize different PQC algorithms.

### 1.1. Problem Statement and Research Questions

This competition raised fundamental questions about the new PQC algorithms' applicability to embedded devices. As a result, a significant amount of embedded PQC research has been conducted in the last few years. However, obtaining an overview of the current state of research is challenging due to the fast-moving competition and changes in the PQC algorithms. This leads to the following research questions:

**RQ1** What are the strengths and weaknesses of existing PQC algorithms?

**RQ2** Which algorithms are suitable for which use case in the context of embedded devices?

### 1.2. Contributions

To answer these questions, we survey the research field of PQC for embedded devices, focusing on the ARM Cortex M4 as a reference architecture and real-world use cases including secure communication, secure boot, and secure software updates. Our objective is to provide a consolidated overview with the intention to assist developers and integrators in better understanding the challenges they may encounter when implementing PQC for their specific applications. To ensure a reliable representation despite the fast-moving PQC research, this paper focuses on already standardized algorithms and software implementations as they are more flexibly applicable than specialized hardware.

## 2. Related Work

Collecting and categorizing PQC research has been a regular aspect of the NIST competition, as NIST itself publishes a report [8] at the end of each round summarizing the most significant advances. However, due to the fast-paced nature of PQC research, new insights are constantly gained, and algorithms that were deemed secure get broken. Standardization of algorithms did not begin until the end of the third round of the NIST competition in 2022. At the time of writing, these algorithms are still in draft form. This work will focus on these standardization candidates since they are the most mature algorithms and most likely to be adopted into practice.

Notable other recent surveys in the field of PQC, include those by Dam et al. [9] and Koziel et al. [10], which survey the current literature and the trends in publication. They focus on the NIST round 3 algorithms, their improvements, and their implementation in software and hardware. In a similar fashion, Alnahawi et al. [11] categorize papers by keywords and application fields and outline the migration process and standardization efforts. Recently, their team also collected research on the PQC adoption regarding quantum-safe electronic identity documents and machine-readable travel documents [12]. The chip cards used in such applications are often highly constrained with only a fraction of the RAM and computing capabilities a typical embedded device has. Furthermore, protocols and use cases for electronic identity documents are highly specialized and translate only partly to other embedded devices. For this reason, our paper does not consider embedded devices with lower capacity than the ARM Cortex M4 and focuses on general-purpose use cases. A survey by Kyung-Ah Shim [13] analyzes Post-Quantum (PQ) signature algorithms in the context of vehicular communication. Similar to other surveys, she focuses on the PQC algorithms' performance in software and hardware. The embedded devices found in vehicles are often more powerful than the ARM Cortex M4 and have very specialized applications that are unique to vehicular communication. In 2020, Fernández-Caramés [14] analyzed the current research on IoT security, collecting performance data on different hardware that is used in IoT environments. The paper refers to NIST round two and in contrast to this work, the author does not examine specific protocols and use cases but rather focuses on the standalone performance of the algorithms themselves.

Recently, in 2024, Alnahawi [15] published a survey on PQ TLS. While their findings on the overall good performance of PQC in TLS match our results, they focus largely on ideal network conditions, which are rare for real world embedded devices.

## 3. State of Art

Since the formulation of Shor's algorithm [4], which threatens to break conventional asymmetric cryptography, academic research has tried to find new algorithms capable of securing information in a post-quantum future. The field of PQC got a special focus in 2016 when NIST started a competition to standardize post-quantum signatures and key exchange algorithms [16]. The competition is currently in preparation for Round 4 after the first three standardization candidates have been nominated [8]. As indicted by Table 1, algorithms in the standardization process must meet specific security levels which correspond to the effort of breaking conventional symmetric cryptography by a PQ adversary. Standardization candidates should have multiple parameter sets to fulfill different security levels. The two leading algorithms to attack conventional symmetric cryptography and hash functions are based on the Grover algorithm [17]. It allows PQ adversaries to search the key space of AES in $\mathcal{O}(\sqrt[2]{n})$. A variation of the Grover algorithm by Brassard et al. [18] allows finding a hash collision of SHA-2 in $\mathcal{O}(\sqrt[3]{n})$. In addition to the NIST competition, the Internet Engineering Task Force (IETF) also worked on standardizing PQC algorithms and their integration into protocols.

Table 1. NIST SECURITY LEVEL REQUIREMENTS

| Security Level | Security Analogous to Breaking |
|---|---|
| 1 | AES-128 = $\mathcal{O}(2^{64})$ |
| 2 | SHA-256 = $\mathcal{O}(2^{85})$ |
| 3 | AES-192 = $\mathcal{O}(2^{96})$ |
| 4 | SHA-384 = $\mathcal{O}(2^{128})$ |
| 5 | AES-256 = $\mathcal{O}(2^{128})$ |

Attack on AES by Grover [17]
Attack on SHA hash-family by Brassard et al. [18]

PQC has two main primitives: Key Encapsulation Mechanisms (KEMs) and Digital Signature Algorithms (DSAs). KEMs aim to secure symmetric key material for a shared secret, while DSAs provide authentication by signing and verifying digital messages. PQ KEMs are intended to replace conventional key-sharing protocols, such as Diffie-Hellman Key Exchange (DHKE), but they slightly differ in application. KEMs use an ephemeral asymmetric key pair to encapsulate the secret material itself, instead of both parties exchanging information to generate a shared secret. The general application flow involves generating a key pair, sending the public key to the other party which then encapsulates the secret and sends it back. The initial party can then decapsulate the secret using the generated private key upon receipt. PQ DSAs are more similar to their conventional predecessors. They use a long-term established key pair to provide authenticity by signing and verifying messages.

Since standardization is ongoing, a full transition to PQC in long-term applications seems not recommended. Due to the fast-changing PQC research field, early adopters are advised to follow a hybrid strategy, especially for KEMs [19], [20], to protect against a 'store now, decrypt later' attack. Hybridization aims to combine conventional cryptography and PQC for specific use cases. Although this approach doubles the cost, it protects against PQ adversaries while guaranteeing conventional security should the less mature PQC algorithm be broken in the future. For PQ DSAs, this approach seems to be less popular since they are a greater

burden on the embedded devices and authentication cannot be broken retroactively.

There are multiple promising mathematical problems suitable for PQC algorithms. Although there are algorithms based on other problems, this paper focuses solely on the already standardized KEMs and DSAs from IETF and NIST. The following subsections group the different algorithms by their underlying cryptographic primitives and explain key performance indicators.

### 3.1. Hash-Based Algorithms

The class of hash-based algorithms includes all algorithms whose security is based on cryptographic hash functions. Conventional hash functions are deemed to be secure against a PQ adversary, however, their security strength is weakened. This makes them a prime candidate for a conservative choice as a PQC algorithm since the underlying security assumptions are well understood. The best currently known attack on cryptographic hash functions is the attack by Brassard et al. [18], reducing the pre-image search by $\sqrt[3]{O}$. Hash-based algorithms are exclusively used for signature algorithms which can be divided into two categories: stateful and stateless.

Stateful algorithms are based on One-Time Signature (OTS) keys that utilize a given hash function in a chain-like manner. The secret key, signature, and public key are part of this chain which can only be used once. To utilize this technique more than once, these OTS keys are combined in a Merkle hash tree which allows a single public key to be derived from all the OTS public keys. The final signature for a message consists of the OTS signature, the OTS public key, and an authentication path that contains all the necessary nodes of the Merkle tree to derive the root node, which corresponds to the main public key. Since signers cannot use the OTS keys more than once, they need to keep track of which OTS keys have already been used. This state needs to be updated after each signature and kept for the keys' lifetime. The IETF standardized two stateful hash-based algorithms, namely LMS and XMSS, in RFC 8554 [21] and RFC 8391 [22], respectively. Their practical difference lies in the variety of parameters that can be adjusted to the use case. The most important parameter is the height of the Merkle tree which influences the number of signatures that can be generated. A height of n will result in $2^n$ available signatures. Since large trees are difficult to generate and manage, both XMSS and LMS have a hyper tree variant, where the Merkle tree root node is used as a leaf of the above tree. This is called multi-tree XMSS [22] or Hierarchical Signature System (HSS) for LMS [21]. It allows generating new Merkle trees on demand if the signatures of one tree are exhausted. Additionally, it reduces the number of nodes needed in the authentication path for a single signature. Furthermore, LMS allows for adjustable Winternitz parameters used in the construction of the OTS, which impacts signature size as well as verification performance [21]. In 2020, NIST published a recommendation [23] for stateful hash-based PQC algorithms based on XMSS and LMS.

Stateless algorithms are based on a similar construction, but instead of an OTS they use a so-called Few-Time Signature (FTS) (FTS) scheme as leave nodes in the Merkle tree. The only algorithm employing this technique is the signature algorithm SPHINCS+ [24]. It is currently in the standardization process under NIST FIPS 205 [25]. In detail, SPHINCS+ uses the forest of random subsets (FORS) scheme to sign messages. In FORS, the secret key is split into pieces whose hashes build the leave nodes of multiple binary hash trees whose root nodes are compressed into a single FORS public key. Similar to stateful algorithms, these FTS are then used as leaves of a Merkle hypertree whose root node serves as the single public key. Authentication paths for both the FORS and the Merkle tree are included in the signature to verify the root node from the FORS leaves.

Both stateful and stateless hash-based algorithms have similar performance metrics and characteristics that make them a fitting choice for specific use cases. First and foremost, the security assumptions are well understood and based on years of research. From all of the PQC algorithms, hash-based ones are the most mature, while achieving a high NIST security level, due to the inherent security of the used hash functions. Another distinguishing factor are their small public and private keys, ranging from 32 to 256 bytes. The signature size on the other hand is highly dependent on the Merkle tree's height and the utilized hash function. Especially SPHINCS+ signatures can grow very large ($> 10$kB) due to the FORS authentication path. While their signing and key generation performance is among the worst of the presented PQ DSAs, their verification performance is decent. All of the mentioned DSAs can operate with the SHA-2 and Keccak hash schemes. Since these are widely used in non-PQ applications, hardware accelerators for these hashes are available for a variety of platforms. Therefore, performance can be increased on platforms with the corresponding hash accelerator [26], [27]. Furthermore, hash-based algorithms are the most customizable out of all PQC algorithms. SPHINCS+ has a wide range of underlying hash functions and different construction methods that can be employed. LMS and XMSS both have definable Merkle tree height, with LMS additionally having its Winternitz parameter adjustable. In overall performance, the stateful algorithms beat SPHINCS+ in almost every category. However, managing the state is a very harsh requirement that limits the application possibilities of LMS and XMSS.

### 3.2. Lattice-Based Algorithms

Lattice-based cryptography is probably the most promising approach for asymmetric PQC algorithms. It gained the most traction over the NIST competition and spawned the most algorithms for digital signatures and key encapsulation [8]. Using lattices for cryptography came up in the early 2000s after the basic underlying problem was proven to be NP-hard in 1996 by Ajitai [28]. The basis for most algorithms is the Shortest Vector Problem (SVP) in high-dimensional lattices. The idea is to build a lattice from a well-formed, short base and then derive a scrambled base

Table 2. SUMMARY OF KEM AND PQ DSA USAGE

| | KEM | | | Signature Algorithm | | | |
|---|---|---|---|---|---|---|---|
| **Use Case** | **Key Generation** | **Decapsulate** | **Encapsulate** | **Key Generation** | **Verify** | **Sign** | **Long-term Key** |
| TLS | ✓ | ✓ | ✓/x[1] | x | ✓ | ✓/x[1,2] | x |
| Secure Boot | x | x | x | x | ✓ | x | ✓ |
| SUIT | x | x | x | x | ✓ | x | ✓ |

[1] Used when the device acts as a server, [2] Used when TLS client authentication is enabled

from the same lattice. The simple, well-formed base is used as a private key, while the scrambled base can be publicized. A message is interpreted as a specific point in the lattice and a random error is added to deviate from the exact point while still being the closest point within the lattice. Computing the original point without the added error is relatively simple when using the well-formed base, but computationally very expensive when using the public, scrambled base. This problem is also called the closest vector problem and can be reduced to the SVP which is the problem of recomputing the well-formed base from the scrambled one. There are multiple other problems involved in lattice cryptography, most notably the Learning With Errors (LWE) problem, which is a more general problem of introducing random errors to a secret value. LWE was introduced by Regev [29] in 2005 and can be reduced to the SVP. The three lattice-based algorithms that are currently being standardized by NIST after round 3 are all based on the SVP or variants of the LWE.

CRYSTALS-Kyber is the only KEM that is currently being standardized after round three under FIPS 203 [30]. It is based on the module LWE problem in cyclotomic rings and uses the SHAKE-256 hash for key generation. Each transaction generates new keys that encapsulate the symmetric shared secret. Its performance is significantly better than RSA encryption for key transfer and also outperforms the standard Finite Field DHKE and even Elliptic Curve Diffie-Hellman (ECDH) in terms of speed [8]. However, in comparison to DH key exchange the memory footprint increases from a few bytes to multiple kilobytes for public key and ciphertext, depending on the security level.

CRYSTALS-Dilithium originated from the same research collective as Kyber and is also based on the module LWE on lattices. It is one of two lattice-based PQ DSAs that are being standardized under FIPS 204 [31]. However, it uses the Fiat-Shamir with Aborts [32] technique to generate digital signatures. Overall, it is regarded as the most well-rounded PQ DSA since its sign and verification performance is closely competitive to ECDSA on the lower security level [8]. The only disadvantage of Dilithium is the relatively large signature and public key size, which increases substantially with a higher security level.

Falcon is the last of the lattice-based algorithms that is being standardized by NIST. In contrast to the previous algorithms it is not based on a derivation of the LWE problem, but rather on the short integer solution (SIS) problem over lattices [33]. The SIS problem can be reduced to the SVP, which makes the underlying hardness assumption similar to LWE. Falcon uses a trapdoor sampler based on the Fast Fourier Transformation [34] to generate its signature in a hash-and-sign scheme. Performance-wise, Falcon provides the smallest signature and public key of all DSAs as well as a fast verification performance. However, it is magnitudes slower than Dilithium or conventional ECDSA in signing messages and requires 53-bit floating point arithmetic precision. This is usually not found in embedded system hardware which usually only provides 32-bit floats. As a result, Falcon needs to emulate higher precision operations in software, which slows it down, especially during signing.

The data is primarily sourced from the pqm4 [35] library, which is a part of the mupq project that develops optimized PQC implementations for embedded platforms and FPGAs. The pqm4 implementation uses assembly code optimized for the Cortex-M4 processor and is regularly updated with changes coming from standardization and NIST round advances. It is available open-source on GitHub [1].

## 4. Use Cases for Embedded Devices

The following section is based on the ARM Cortex M4 as a reference platform chosen by NIST as the representative for embedded CPUs [36]. The ARM Cortex M4 is a 32-bit CPU in the mid-range segment of embedded hardware. It is typically equipped with 256 kB of RAM, 1 MB of flash memory, and operates at a maximum clock speed of 220 MHz.

To decide which of the aforementioned algorithms is suitable for deployment on this hardware, different use cases are described. The most prevalent use cases for asymmetric cryptography in IoT can be divided into three main groups: secure network communication, trusted execution in the form of secure boot, and secure updates. Specifically analyzed are TLS as representative of secure network communication, secure boot, and SUIT as a workflow for secure updates. Table 2 shows the different asymmetric cryptography applications within these use cases.

### 4.1. Transport Layer Security

TLS is the de-facto standard for securing network traffic over the web. It provides an end-to-end-secure channel over a TCP connection, including bidirectional authentication, data confidentiality, and message integrity. Because of its flexibility in terms of cryptographic algorithms, it is also a popular choice in embedded environments [37]. Traditionally, embedded platforms act as clients that connect to

1. pqm4 library source code available at https://github.com/mupq/pqm4

management instances such as Supervisory Control and Data Acquisition (SCADA) systems or servers on the internet. However, due to the increasing use of embedded devices in IoT hardware applications, they are also increasingly being used as servers, for example in smart home bridges, or they come with a web interface for configuration. The latest version, TLS 1.3 [38], was standardized in 2018 and offers many improvements over the previous version. However, TLS version 1.2 remains prevalent in embedded systems today due to their long lifespan and inability to be easily upgraded.

#### 4.1.1. TLS Phase Analysis

A TLS session consists of two phases. The first phase is the *handshake protocol*, which authenticates the parties and agrees on cryptographic algorithms. During this phase, a master secret is negotiated, conventionally using DHKE. This master secret is used to derive the subsequent transport keys, which are in turn used to symmetrically encrypt the application data in the following *record protocol*. X.509 Public Key Infrastructure (PKI) certificates are used to authenticate both the server and, optionally, the client. To better illustrate the difference between conventional TLS and PQ TLS, we first describe the classic TLS handshake from which the changes for PQ are made. A visualization of a full PQ TLS handshake can be found in Figure 3.

When initiating a TLS 1.3 handshake, the client starts with a *ClientHello* message containing the supported cipher suites and the client parameter for the DH key exchange. The server responds with a *ServerHello* message containing the server-side DH parameters and the chosen cipher suite from the list offered by the client. Once the server has chosen the cipher, it can generate the necessary server-side parameter and calculate the master secret using the client's information. From here on, all messages are encrypted using secrets derived from the master secret agreed upon via DH key exchange. Moreover, the server sends the *Certificate* message containing its own X.509 certificate chain up to the root authority. If mutual authentication is required, the server requests a client certificate via a *CertificateRequest* message. The server's final message at this stage is a finishing message containing a signature over all the previously exchanged messages. On the client side, the master secret can now be calculated to decrypt the received handshake messages. In the next step, the server certificate chain is validated up to the root CA. After the signature is validated, server-side authentication is complete. If requested by the server, the client sends its certificate chain and a signature over the handshake to prove possession of the private key. The client completes the *handshake protocol* with an HMAC-authenticated hash over all previous messages.

Asymmetric cryptography vulnerable to PQ adversaries is only used in the *handshake protocol*. However, it is present in multiple steps of the handshake, most notably in the DH key exchange and the X.509 certificates. X.509 certificates contain identity information about the end entity, its public key, and a signature of the issuing CA. The size of the individual certificates and the length of the certificate chain are the primary factors that determine the amount of data transmitted during the handshake. In an embedded environment, network bandwidth may be limited, causing the entire handshake to bottleneck while waiting for the receipt of the certificate chain. In conventional TLS, the most expensive computational operation regarding certificates is usually the verification of the certificate chains, due to the multiple signatures that need to be verified. The signing operation is only required by the client if mutual authentication is performed.

The key exchange is the fundamental part of TLS which allows both parties to partake in the generation of a shared key. For this purpose, an ephemeral key pair is generated to ensure forward secrecy, even if the certificate's key pair is compromised. Conventional Diffie-Hellman (DH) key exchange can be used in conjunction with RSA or elliptic curves, differing in the exchanged key sizes and calculation performance. For embedded systems, Rivest–Shamir–Adleman cryptosystem (RSA) can be very expensive to the point where it becomes the most expensive operation during the handshake. In comparison, ECDH outperforms RSA by at least a factor of 10 in terms of computation time [39].

#### 4.1.2. Embedded PQ TLS Literature Analysis

TLS is used in a wide variety of applications and is one of the central protocols in securing network traffic. The following papers focus on its employment in embedded environments with regard to the ARM Cortex M4. They test the PQC algorithms in various network constellations and with different PKI hierarchies. The benchmarks primarily measure handshake latency and RAM usage. Table 7 gives an overview and short summary of their findings.

One of the earliest research investigating PQ TLS on embedded devices was Bürstinghaus-Steinbach et al. [40]. The authors were one of the first to integrate PQC algorithms into the popular mbed TLS library. Their work focuses on proving the feasibility of full PQ TLS 1.2 on three different embedded devices, ranging from a rather powerful Raspberry Pi 3 to a limited field bus option card. They tested Kyber 512 and SPHINCS with SHA-256 and SHAKE-256 in the fast variant, respectively. The KEM public key is transmitted during the very first *ClientHello* message. All three embedded devices are tested as server and client. Authentication is server-sided only, with a single self-signed server certificate. The results show that PQ handshake times are very similar to ones with conventional crypto, even outperforming the Elliptic Curve Diffie-Hellman Ephemeral (ECDHE)-Elliptic Curve Digital Signature Algorithm (ECDH) baseline in some cases. This is due to the good performance of Kyber in comparison to ECDHE and the fast verification speed of SPHINCS when compared to ECDH. As servers, the embedded devices are slow in comparison, taking over 10-50 times as much as the conventional handshake. This was expected due to the slow SPHINCS signing process. While this work does not include the ARM Cortex M4 specifically, it shows that similar powerful devices can successfully perform PQ TLS handshakes.

Since the focus was on verifying the general feasibility of PQ TLS, only a single combination of DSA and KEM was tested. The authors were the first to show that deploying PQ TLS on embedded devices is possible, without the need for specific hardware accelerators. However, SPHINCS might not be a reasonable choice as a PQ DSA when the embedded device needs to sign messages.

One of the most notable works on the Cortex M4 is the paper from Tasopoulos et al. [39]. They benchmark all NIST round 3 KEM and DSA candidates on the M4 and integrate them into the WolfSSL TLS 1.3 library. If available, they use the assembler-optimized implementation of the pqm4 library. They evaluate different KEM and PQ DSA combinations that they introduce as supported groups in TLS 1.3. Similar to previous implementations, the KEM public key is exchanged in the *ClientHello* message. The embedded device acts as a client against a regular high-powered server, that is directly connected via Ethernet. Their tests utilize mutual authentication, with a two-layered Public Key Infrastructure (PKI) certificate hierarchy. In a single test case, all certificates use the same PQ DSA meaning that both server and client will sign the handshake message transcript with the same algorithm. In addition to the PQ performance tests they also benchmark conventional TLS 1.3 handshakes using EC and RSA algorithms. Their evaluation shows that in comparison to RSA, all PQC combinations perform rather well, except SPHINCS. The sign operation of RSA is very expensive, which slows down mutual authentication considerably since the client needs to sign the handshake transcript with its private key. When comparing with ECDH, combinations with Dilithium and Kyber are very competitive, even outperforming it on security level 1. Combinations with Falcon and Kyber take double the handshake time but only need to transmit half of the communication size. This is due to the small signatures and public keys of Falcon. Tests with SPHINCS and Kyber yield results similar to the server tests by Bürstinghaus-Steinbach et al [40]. The mutual authentication and slow signing performance of SPHINCS make it slower by a large margin compared to other PQC combinations. In 2023 they extended their work with an analysis of the energy consumption of PQ TLS [41]. Their findings largely concur with their previous results, as again PQC, except SPHINCS, is competitive to even ECDH in terms of power consumption.

Schöffel et al. [37] also focus on the Cortex M4 as an embedded platform, but instead of TLS 1.3, they implement the NIST round 3 candidates into Transport Layer Security (TLS) 1.2. They use the popular Mbed TLS library and integrate the pqm4 assembler-optimized algorithms. Similar to Tsaopoulos et al. [39], the authors test a variety of PQ KEM and DSA combinations by introducing new supported groups and exchanging the KEM public key in the *ClientHello*. What makes their work stand out from the others, is their test setup. Instead of directly connecting the embedded device to a communication partner, their battery-powered Cortex M4 platform is connected to a gateway via Bluetooth Low Energy which is then connected to the server via ethernet. This corresponds to a distributed sensor network that can be found in many IoT use cases. Due to this setup, high bandwidth requirements are punished and algorithms with large signatures, public keys, or ciphertexts perform worse. Server and client are mutually authenticated by a two-layered PKI. The authors test heterogenous PKIs with different algorithms for client, server, and CA. In this test setup, most KEMs, again, perform well in comparison to conventional ECDHE. Kyber is still the fastest of all tested KEMs. However, it is unable to outperform ECDHE due to its communication overhead. Notable are the measurements with PQ DSAs. While Dilithium is faster than Falcon in signing, handshakes with Dilithium server and CA certificates were the slowest in comparison to Falcon. Due to the constricted network bandwidth, handshakes with Dilithium certificates spend most of the time transmitting and receiving data. When using Falcon for all certificates, the overall handshake completes in half the time compared to pure Dilithium and overall presents the best-performing full PQC combination. However, instead of transmission delay, Falcon signature computation takes a significant share of the handshake time. In addition to handshake latency, battery consumption of the embedded platform is measured and lifetimes estimated. Computation on the embedded device impacts battery consumption more than transmitting data, making Falcon use more battery even though it finishes the handshake faster. When using Falcon for the CA and server certificate, but Dilithium for the client certificate, the overall handshake time is only marginally slower than using Falcon for the client certificate, but signature computation time is reduced drastically. This results in an almost halved battery consumption.

Gonzales et al. [42] focus on KEMTLS. KEMTLS was first proposed by Wiggers et al. [43] in 2020 and is a different implementation of PQC into TLS 1.3. Instead of PQ DSAs, KEMTLS uses KEMs for authentication during the handshake protocol. This is achieved by establishing long-term KEM public keys and distributing them via a PKI. CAs still utilize PQ DSAs within the PKI itself to sign messages, but the end-entity certificates contain a KEM public key. A party can prove its authenticity by decapsulating a secret that the other party encapsulated with the respective public key. This secret is incorporated into the master secret. Thus, the communication can only continue if the parties are indeed in possession of their respective private keys. The main motivation behind KEMTLS is a performance increase and simplification due to the minimization of the needed number of PQC algorithms during the TLS handshake. Since KEMs are needed to establish the shared master secret for symmetric encryption anyway, using the same KEM in authentication simplifies the code base and reduces complexity. Furthermore, most KEMs perform encapsulate and decapsulate operations faster than the PQ DSAs sign and verify. The authors evaluated different KEMTLS combinations on the Cortex M4 and compared them to PQ TLS using PQ DSAs for authentication. Their PQ TLS builds on the WolfSSL library, similar to the work of Tasopoulos et al. [39]. Their test setup includes a two-layered PKI hierarchy with a PQ DSA CA certificate and a KEM server certificate. In the

Table 3. CONNECTION CHARACTERISTICS FROM GONZALEZ ET AL. [42]

| Connection Type | Bandwidth | RTT |
|---|---|---|
| Broadband (BB) | 1 Mbit | 26 ms |
| LTE Machine Communication (LTE-M) | 1 Mbit | 120 ms |
| Narrowband-IoT (NB-IoT) | 46 Kbit | 3 s |

conducted KEMTLS and PQ TLS experiments, only the high-powered server is authenticated. The authors simulate three different network conditions: regular broadband, LTE-Machine Type Communication (LTE-M), and Narrowband-IoT (NB-IoT). On broadband, the regular PQ TLS implementation benefits most from deploying Falcon as PQ DSA in the CA and Server certificate. Since the embedded platform does not need to sign the handshake, the fast verify operation from Falcon and its small public keys and signatures speed up the handshake significantly compared to Dilithium. As KEM, SABER achieves the best performance, being less than one percent faster, than combinations with Kyber. For PQ TLS, this trend continues through all the different network setups. Pure Dilithium PKIs take around double the handshake time in comparison to pure Falcon ones, while Kyber and SABER perform best as KEMs. In the KEMTLS experiments, combinations with Kyber and SABER perform well, with SABER again being faster than Kyber by a small margin. As PQ DSA for the CA certificate, Rainbow seems to be performing best. However, the used parameter set of Rainbow is now broken [44]. Ignoring Rainbow combinations, Falcon is again the best-performing PQ DSA. Comparing KEMTLS and PQ TLS, KEMTLS handshake traffic is roughly equal in size and performs similarly in the given setups. The only case where KEMTLS outperforms PQ TLS by a large margin is the NB-IoT network when Rainbow is used for the CA certificate.

#### 4.1.3. Findings

There are some general results that can be observed from all experimental setups. (i) The choice of KEMs has only a slight impact compared to the selected DSA and the network bandwidth. Kyber outperforms ECDH in terms of computation speed on the ARM Cortex M4, while only marginally adding to the handshake bandwidth. As Kyber is currently the only standardized KEM, early adopters may have less incentive to implement other options, regardless of network conditions or topology.

The choice of PQ DSA is more complicated and depends on multiple factors of topology and network condition. The PQ DSA is responsible for the majority of the required handshake bandwidth. The size of the PQ public key and signature affects the overall handshake size multiple times, as they are present in each of the transmitted certificates. Among all the standardized PQ DSAs, research focuses on the two lattice-based options Falcon and Dilithium. The primary tradeoff between these two is bandwidth versus signing performance. Dilithium provides fast signature computation and verification, but it significantly increases the handshake size compared to Falcon. Yet, Kyber and Dilithium combinations can outperform even conventional, mutually authenticated ECDHE-ECDH TLS in fast networks. Falcon, on the other hand, performs well in restricted network environments and if there is no client authentication required. It suffers from the missing hardware support for 64-bit floating point precision, which makes its signing operation perform poorly. Falcon performs well in every network condition if the client does not need to sign the handshake, as shown in Gonzales et al. [42]. Furthermore, Falcon can be faster than Dilithium in heavily restricted networks, even if mutual authentication is required. In the experiment of Schöffel et al. [37], the two-layered PKI with Falcon performs the fastest handshake. However, it takes more computation time than Dilithium. This presents a trade-off between power consumption and handshake speed. Furthermore, both Schöffel et al. [37] and Tasopoulos et al. [39] suggest that heterogeneous PKIs might perform best in the embedded use case. Using Falcon for server authentication and CA certificates, while deploying Dilithium certificates to the embedded clients, appears to yield the best of both algorithms.

All papers also measure RAM usage during the handshake. Depending on the algorithm and security level, PQ TLS requires 40-80 KBs of RAM, which is 40 times more than the average EC TLS implementation. However, the ARM Cortex M4 is usually equipped with 256 KB of RAM which still leaves room for the customer application to run. The other hash-based DSAs that are being standardized find almost no mention in TLS research. In TLS, they are considered impractical due to slow signing performance and large signatures.

KEMTLS is an interesting field of research for embedded platforms since it reduces memory requirements and handshake bandwidth. In Gonzalez et al.'s experiments [42], it does not outperform the regular PQ TLS with Falcon and Kyber. However, they test it without mutual authentication which would significantly impact PQ DSA performance, due to the expensive sign operation. Since KEMTLS uses the KEM encapsulation operation to authenticate, which performs well on embedded devices, another comparison to PQ TLS with mutual authentication would be interesting.

### 4.2. Secure Boot

On an embedded device, secure boot aims to restrict code execution to legitimate software only. This is achieved via digital signatures that authenticate each software image that is supposed to be executed. Naturally, this includes software from multiple entities and stakeholders ranging from the producer of the chip to the vendor of the embedded solution.

#### 4.2.1. Secure Boot Process

The full authentication process of the secure boot chain is visualized in Figure 1. It starts with the execution of a trusted zero-stage boot loader (ZSBL) located in read-only memory (ROM). In embedded platforms, this boot loader initiates a chain of trust by loading the next firmware image from flash memory into the internal RAM, authenticating it,

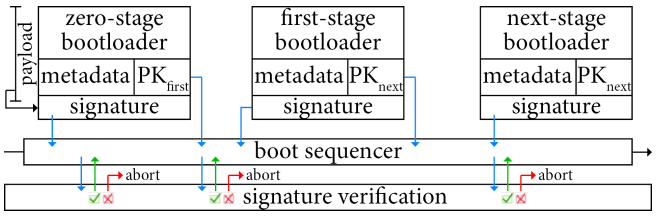


Figure 1. Secure boot validation chain adapted from Kumar et al. [45]

and passing the control flow to it. Authentication is based on the certificate attached to each of the software images, a signature over the image itself, and corresponding metadata. The initial stage of this chain, which resembles the root of trust, is particularly critical. This trust is derived from a boot loader public key whose hash is burnt into the chip during manufacturing. The subsequent boot loader images are signed with the corresponding private key and are verified by the previous boot loader. Once the control flow passed all the boot loader images, the firmware is running and can initialize additional hardware or start customer software. Since this software is usually developed by entities other than the firmware, different key pairs are used to sign it. The public keys are part of the firmware and thus implicitly authenticated in the chain.

Secure boot imposes certain restrictions and requirements on the signature algorithm in use. The longevity of public keys is here the primary consideration. The chip manufacturer generates and manages the key pair for the ZSBL, which is typically used for an entire chip revision. As the public key is burnt into each chip, it cannot be replaced during its lifetime. Therefore, the security level of the signature algorithm should be chosen conservatively to accommodate advances in computing power and cryptanalysis. Due to the rapid advances in PQC algorithms and their cryptanalysis, decisions for such a long product lifespan are difficult. Under these circumstances, hybridization could be highly beneficial. However, the strict hardware and timing requirements of secure boot make it unfeasible.

The second restriction is performance. Signing and key generation performance are less critical in this case since they are done during the manufacturing of the chip and not during the boot itself. On the other hand, signature verification directly affects boot times and is its most important performance factor. Especially for embedded systems with safety functionality, boot time can be crucial. Slow verification repeatedly impacts boot time since each boot loader image has its own signature that needs to be verified.

Finally, the most significant limitation is memory. On the embedded platform, the primary constraint is the internal on-chip RAM size, which is approximately 256Kb for the ARM Cortex M4. The boot loader images, their corresponding signatures, the boot loader public key, and the verify code itself are loaded into this internal RAM. Since the CPU can only use internal RAM during the first stages of the boot loader, there is a hard limit on memory consumption. To ensure success in this use case, a PQC algorithm must have minimal signature and public key sizes, as well as minimal stack usage during verification. Once the firmware is fully operational, it can initialize external RAM if available, which relaxes the memory requirements for customer software.

#### 4.2.2. PQ Secure Boot Literature Analysis

This section focuses not only on the ARM Cortex M4 but also on multiple other embedded platforms to provide a more comprehensive overview of the current state of the art. Whenever possible, a direct comparison to the ARM Cortex M4 is provided to enable comparability. An overview of the mentioned papers can be found in Table 5.

One of the first papers targeting secure boot with the now standardized algorithms was authored by Marzougui et al. [46] in 2019. They focus on software authentication in embedded devices and identify secure boot and protection of intermediate keys as the most important applications for PQ DSAs. The paper analyzes the requirements that PQC must fulfill for these two applications and concludes that verification time and RAM usage are critical. Next, the authors evaluate these requirements for the PQC algorithms of the time to identify promising candidates for implementation in secure boot. Since in 2019 NIST round two has not yet concluded, this research still incorporates algorithms that have later been eliminated from the competition. They identify hash-based and lattice-based algorithms as the most promising candidates. From this conclusion, they benchmark XMSS and qTESLA, a lattice-based algorithm that was broken in round 2 but has very similar performance characteristics to Dilithium. As their embedded platform they choose the ARM Cortex R5 processor, which is more powerful than the ARM Cortex M4 and is equipped with twice as much RAM. Their software benchmark shows that XMSS with a tree height of 20, produces a smaller RAM footprint of less than 5Kb while qTesla takes around 50Kb. However, qTesla verification performance is almost five times as fast in comparison to XMSS. This presents a trade-off between RAM usage and verification speed. The authors conclude in favor of verification speed since it repeatedly impacts boot times during secure boot.

Wagner et al. [26] target hash-based PQ DSAs for embedded secure boot. The authors identify similarities in computation between SPHINCS, LMS, and XMSS and develop a hardware/software co-design for secure boot utilizing a newly designed hardware accelerator. Furthermore, they investigate different parameter sets for stateful and stateless schemes based on SHA-256, with a special focus on the Winternitz parameter of LMS. The LMS algorithm is standardized with a wide range of parameters for its OTS, that affect signature size and verification performance. The Winternitzer parameter influences the signature size and verification performance. Increasing the parameter decreases the signature size but increases the amount of hash computation within the OTS chain. The authors find that Winternitz parameters of 2 and 4 result in the same number of hash function calls, even without an accelerator. Therefore, a Winternitz parameter of 2 should be avoided since

it has a larger signature than 4 but the same performance. They also identify that the calculation of the hash chain constitutes the majority of the operations when using hash-based algorithms, with Merkle tree authentication taking less than 15 percent. Thus, hardware that accelerates the hash chain computation is most valuable to the performance of hash-based algorithms. However, using a standard SHA-256 hardware hash accelerator, shifts the bottleneck to the communication with such an accelerator since the hash algorithm must be called repeatedly in the hash chain. The authors solve that problem by designing a new hardware accelerator for the hash chain computation that can be fitted to the different stateful or stateless algorithms during synthesis time. In detail, they improve the OpenTitan SHA256 hardware accelerator by adding a chain register and a digest feedback path. This allows the accelerator to compute an entire hash chain without additional communication overhead.

Additionally, the authors benchmark different parameter configurations on the RISC-V Ibex processor. They test the parameters not only with their newly designed accelerator but also in software implementations and with a general-purpose SHA-256 core. Table 4 shows the averages of the benchmark in context to the signature size and NIST level. The results show that software implementations of LMS struggle with large Winternitz parameters, resulting in a 20-fold increase in verification run time between the lowest (w = 4) and highest (w = 256) parameters. Despite this, LMS still outperforms XMSS even with higher Winternitz parameters. Another finding is that general-purpose hash accelerators are beneficial, reducing the time compared to pure software implementations by about half for stateful algorithms and by a factor of five for SPHINCS. The newly designed accelerator outperforms software and general-purpose SHA-256 core implementations by a significant margin, particularly for LMS where it provides nearly the same performance for all tested Winternitz parameters. As such LMS with Winternitz parameter 256 allows for a fast verification while having the smallest signature when using the newly proposed accelerator.

Furthermore, the authors test the rapidly verifiable signature approach [47] for stateful algorithms where the signer appends a random counter to the message, which yields a shorter OTS hash chain during verification. The approach's results demonstrate that verification speed increases by 30 percent, even with a low number of counters tried, regardless of the Winternitz parameter of LMS on software and general-purpose accelerators. Finally, the paper evaluates the hash-based algorithms directly in the context of secure boot, by simulating a full firmware verification. The different implementations are compared to RSA and ECDSA. In software, the stateful algorithms outperform the conventional algorithms, while SPHINCS performs worse. When accelerated by dedicated hardware, only LMS can beat ECDSA, whose performance is improved by the OpenTitan BigNumber accelerator.

Kampanakis et al. [48] investigate SPHINCS and LMS for the UEFI secure boot on high-end devices and the performance of FPGA LMS implementations. Although their primary focus is not on embedded platforms, their research provides insights that are also relevant for constraint devices. They explore various parameter sets of SPHINCS and LMS that can be used for the UEFI use case. To increase the number of signatures per public key with LMS without significantly reducing signing performance, they opt for the HSS hyper-tree approach. This approach adds another Merkle tree above the first one instead of increasing the tree's height. It enables the combination of the various keys required for secure boot in a single tree hierarchy and allows for the revocation of keys in a granular manner. The authors briefly consider using trees with more than two children per node but reject it since the memory and signature size benefits are negligible compared to the performance impact. Additionally, they implement LMS with a Winternitz parameter of eight on an FPGA and compare it to FPGA RSA implementations. The results show that LMS verification is outperforming RSA with 4096 bits, while also having a similar footprint on the FPGA. The authors suggest a tradeoff in FPGA LMS design between logic area and performance and identify SHA-256 hash logic as the most significant optimization in terms of speed or size.

Table 4. VERIFICATION PERFORMANCE OF HASH-BASED ALGORITHMS MEASURED BY WAGNER ET. AL. ON IBEX PROCESSOR [26]

| Algorithm | SL | Sign. | Software | SHA-256 | SHA-2$^+$ |
|---|---|---|---|---|---|
| LMS w = 4 | 5 | 4.7 KiB | 20 ms | 8 ms | 5.5 ms |
| LMS w = 16 | 5 | 2.7 KiB | 40 ms | 16 ms | 4 ms |
| LMS w = 256 | 5 | 1.6 KiB | 300 ms | 110 ms | 5 ms |
| XMSS | 5 | 2.6 KiB | 260 ms | 80 ms | 20 ms |
| SPHINCS | 3 | 29 KiB | 700 ms | 200 ms | 50 ms |

SHA-256 denominates general-purpose OpenTitan hash accelerator
SHA-2$^+$ denominates the accelerator designed by the authors

Kumar et al. [45] present another FPGA implementation targeting secure boot focusing on XMSS. The target platform is a RISC-V-based Kintex7 FPGA where the hardware XMSS implementation assists during the secure boot signature validation. As a parameter set, XMSS with SHA-256 and a tree height of ten is selected. The evaluation indicates that the XMSS implementation is comparable in size to ECDSA FPGA implementations, but verification is significantly slower, ranging from a factor of 10 to 30, depending on parallelization. However, the verification process still takes less than two milliseconds in wall-clock time, which is acceptable for most embedded systems. The authors also compare their work to preliminary FPGA implementations of Dilithium, which shows a smaller footprint and faster performance for XMSS.

Román et al. [49] focus on XMSS. The authors propose a new two-stage boot process. The first authentication check performed in this process uses a Physical Unclonable Function (PUF) that relies on SRAM cells instead of a signature and an unalterable public key. The ZSBL checks the first image by computing an HMAC, which relies on a symmetric key generated from a PUF based on the startup values of the device's SRAM cells. SRAM cells have either random or static values after boot, which deviate from device to device, but are fixed after manufacturing. This can be used

Table 5. PQ SECURE BOOT PAPERS

| Reference | Approach | Algorithms | Implementation | Main Findings |
|---|---|---|---|---|
| Marzougui et al. [46] | Identify requirements for PQ secure boot in embedded environments | XMSS, qTesla | Software | - Main limitations are memory requirements and verification performance<br>- Hash-based and lattice-based algorithms are promising |
| Wagner et al. [26] | Hardware/Software co-design for hash-based PQ secure boot | LMS, XMSS, SPHINCS | Software, FPGA, ASIC | - LMS better performance than XMSS and SPHINCS in hardware and software<br>- LMS outperforms ECDSA in software<br>- General-purpose SHA-2 hash accelerators increase verification performance significantly |
| Kampanakis et al. [48] | Test SPHINCS and LMS for UEFI secure boot on different platforms | LMS | FPGA | - Hypertree LMS approach can combine different keys needed for UEFI secure boot<br>- LMS FPGA implementation faster verification than RSA FPGA implementation |
| Kumar et al. [45] | XMSS FPGA hardware optimizations for secure boot | XMSS | FPGA | - XMSS comparable to ECDSA in footprint size but slower verification on FPGA<br>- XMSS smaller and faster than Dilithium in FPGA implementations |
| Román et al. [49] | Novel two-stage secure boot with PUF in combination with PQ DSAs | XMSS | Software | - XMSS signature can be reduced to OTS for speed and size benefits<br>- Compared to ECDSA, OTS validation is significantly faster, and smaller on storage |

to generate a device-specific key that is then utilized for a symmetric HMAC authentication. Once the ZSBL successfully verifies the first image, the control flow is passed on, allowing the next boot loader to load and verify the firmware and application software. For this step, the authors propose using XMSS. However, they suggest using only the Winternitz OTS to verify the firmware. This approach makes the XMSS signature smaller and verification faster since only the hash chain needs to be calculated to obtain the OTS key. The authors argue that the user can validate the authentication path to the manufacturer's public key once at the time of purchase to initially validate the authenticity.

Implementation of this proposal is benchmarked on an ESP32 microcontroller, that is equipped with 512Kb of RAM, an SHA-2 accelerator, and runs on a CPU frequency of 160MHz which makes it slightly more powerful than the ARM Cortex M4. Results indicate that the first symmetric HMAC authentication performs significantly faster than asymmetric verification in software, taking around 6ms. The XMSS OTS validation is compared to ECDSA. Signature verification of the proposed solution is faster than ECDSA by a factor of 10, excluding the firmware hashing. The flash memory requirements are also lower than typical ECDSA implementations due to the small code size of the OTS validation, which outweighs the larger signature size. When comparing the OTS to regular XMSS, the authors observe a signature size reduction of 10-20%, depending on the XMSS parameters.

#### 4.2.3. Findings

The first finding is that current research primarily focuses on hash-based algorithms for the secure boot use case. Algorithms should be chosen conservatively since the RoT in the ZSBL cannot be changed once deployed. This is especially important considering the long lifetime of embedded devices. Together this favors hash-based algorithms as their security is well-understood. Furthermore, hash-based algorithms offer good verification performance and relatively small signatures and public keys which is crucial for constrained embedded devices. In particular, the two stateful algorithms LSM and XMSS gained attention, probably also due to NIST recommending them for firmware authentication [23]. The main disadvantage of having to manage a state is considered less problematic since key generation and signing are performed by the manufacturer and firmware developer, who are not limited by the constraints of the embedded platform. Further information on state management can be found in the NIST recommendation [23] and the work of McGrew et al. [50]. With secure boot, only a limited amount of signatures need to be generated under very strictly defined circumstances, such as the release of new firmware. This makes it an ideal target for stateful algorithms, as state management can be easily centralized.

When comparing LMS and XMSS, LMS offers a wider range of parameters to choose from, particularly the flexible Winternitz parameter that affects the size and verification performance of the OTS. According to Wagner et al. [26], software implementations of LMS generally outperform XMSS on embedded platforms. Campos et al. [51] confirm this observation. An interesting finding of Wagner et al. [26] is that general-purpose hash accelerators can significantly improve the performance of hash-based DSAs, even without specialized additions. Since SHA-2 hashes are used extensively in embedded environments, these general-purpose accelerators are often found in embedded devices and might help adopt the new PQC algorithms. Rapidly verifiable signatures indicate a promising performance improvement. According to Wagner et al. [26], software implementations benefit from a verification speed increase of up to 30 percent. Since the additional effort only affects the firmware provider but releases all devices on every boot process, the approach should be pursued for stateful signatures.

Tree size is an important parameter that is handled differently by the presented papers. With the single tree variant

of LMS and XMSS, the tree height must be sufficiently large to generate enough signatures throughout the lifetime of the key. This depends on the device's update rate, as well as the number of firmware images in the secure boot chain. The smallest tree height considered in the presented papers is ten which performs best but only allows $2^{10} = 1024$ signatures, which might not suffice for the entire lifespan of the device. The next standardized height is 15 for LMS and 16 for XMSS which can generate a more flexible amount of signatures with $2^{15} = 32768$ or $2^{16} = 65536$. It is important to choose the tree height with foresight, particularly for keys utilized during the initial stages of secure boot, to ensure that there are enough signatures available. This is because the public key present in the ZSBL cannot be changed, which limits the total number of signatures that can be used throughout the device's lifetime.

Since larger heights lengthen the authentication path and negatively affect verification performance, the proposal of Román et al. [49] to include the OTS only for firmware signatures might prove valuable for very constrained devices. The authentication path from the OTS key to the Merkle tree root can be verified once at purchase time and after each update, proving authenticity while lessening the burden on the device. Another option for tree height is the multitree approach. Kampanakis et al. [48] propose consolidating the keys involved in the UEFI secure boot use case into a single tree hierarchy. This approach can simplify key revocation and save resources. However, it may be challenging to implement due to the various entities involved in embedded platform development. Further research is needed to explore this approach and its potential use in update mechanisms.

Stateful algorithms appear to be competitive with ECDSA, even in software implementation, with LMS even outperforming ECDSA [26]. SPHINCS is compared to the stateful algorithms in some of the papers, however, it performs worse and comes with a larger signature. Wagner et al. [26] demonstrate that even with specialized hardware accelerators, the verification performance still does not outperform LMS software implementations. Nevertheless, SPHINCS can be used as an alternative to LMS if secure state management cannot be guaranteed. Full hardware FPGA implementations of XMSS and LMS are promising compared to FPGA implementations of conventional algorithms. While dedicated FPGA hardware accelerators for single signature algorithms are not common in IoT devices, they can be found in more specialized settings like industrial environments. However, according to the papers, they do not seem necessary to adopt PQ secure boot in embedded platforms.

## 4.3. Secure Software Updates

Secure updates are another crucial security aspect of embedded devices. Patches to the application software and firmware are vital for their security state, given the long lifetime of up to twenty years. Securing these update processes closes the attack vector of infecting the embedded platforms via inauthentic, malicious firmware. The IETF recognized

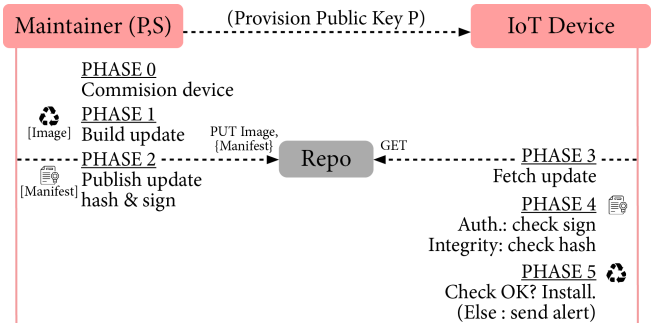


Figure 2. SUIT workflow adapted from Benegas et al. [54]

this threat and founded the *Software Updates for Internet of Things* (SUIT) working group. The working group has published two RFCs [52] [53] to date, with multiple drafts currently in progress. The Software Updates for Internet of Things (SUIT) workflow facilitates updates for IoT devices integrated into the secure boot Root of Trust (RoT) and allows crypto-agility with PQC.

SUIT's primary security objective is authenticity, which is achieved through code signatures and certificates. It protects against various attacks such as malicious updates, replay, and mismatch attacks. SUIT requires the device to be deployed with a trusted boot loader and a long-term public key. New software updates and the corresponding metadata are signed and transferred to the applicable devices. On the device, the signature is verified and the update is installed. Figure 2 visualizes the described workflow. The security and performance requirements for the SUIT workflow are similar to those of the secure boot use case, with the main cryptographic operation being signature verification. However, instead of focusing on runtime performance, the SUIT workflow benefits most from small signatures and public keys, since the update duration is more influenced by network transfer costs than signature validation. Furthermore, the implementation size plays a significant role in the overall size of the firmware and, consequently, in updates containing the cryptographic library. The network bandwidth required for each update containing the library can therefore be directly affected by its size.

### 4.3.1. PQ Secure Updates Literature Analysis

Although secure updates and firmware signing are mentioned as one of the core applications of PQC [8], [23], there is surprisingly little research targeting this use case. Trusted firmware updates are mentioned in some of the secure boot papers [48], [49], but only from the perspective of integrating new key material. Only one study directly targets embedded secure firmware updates with PQC algorithms.

Benegas et al. [54] focus on PQC integration into SUIT for constrained embedded devices. Their target firmware is RIOT [2], an open-source IoT firmware with SUIT integration. The authors differentiate between updates of different sizes ranging from 5kB to 250kB, including and excluding the

2. RIOT operating system, online at https://github.com/RIOT-OS/RIOT

| SUIT - DSA | Firmware on Flash | Stack Usage | Tranfer excl. Library | Transfer incl. Library |
|---|---|---|---|---|
| ECDSA | 52.4kB | 16.3kB | 47kB | 53kB |
| Falcon | +120% | +18% | +1.1% | +120% |
| LMS | +34% | +1.2% | +9% | +43% |

Table 6. RELATIVE COST INCREASE OF FALCON AND LMS FOR SUIT ON ARM CORTEX M4 ADOPTED FROM BANEGAS ET AL. [54]

PQ DSA library. They benchmark SUIT on an ARM Cortex M4, an ESP32, and a RISC-V development board. First, the authors investigate different hash functions for integrity and PQ signature checks. They identify no direct requirement for PQ SUIT but advise utilizing SHA3-256 since it is used in most PQ DSAs, saving flash space and minimizing complexity. The choice of PQ DSA is more complicated and depends on multiple different factors. The authors compare ECDSA to LMS, Dilithium, and Falcon. They find that in small updates, the larger PQC signatures create the most overhead, with Falcon performing best, due to its small signature and public key. LMS provides a good trade-off for larger updates containing the crypto library itself since its implementation is smaller compared to the other PQ DSAs. Comparison between Falcon and LMS against ECDSA for a medium-sized update ($\sim$50kB) can be found in Table 6. LMS is considered to be the most secure option for a long-term deployment such as the RoT. The authors also measure RAM requirements during verification and find that LMS is similar to ECDSA with Falcon being second. Verification performance is also benchmarked and found that on all the embedded devices the PQ DSAs perform faster than ECDSA. However, verification typically does not significantly impact the overall update performance, especially with larger updates. For the largest update of 250kB, the choice of PQ DSA hardly changes the overall performance since network transfer is the bottleneck and the choice of signature affects the total image size only by a few percent. Dilithium ranks second on both small updates in terms of signature size overhead and large updates in terms of implementation size. However, it does require more stack memory than other PQC algorithms.

#### 4.3.2. Findings

Secure firmware updates present a multi-dimensional challenge due to network transfer costs and the long-term key that forms the RoT. The size of the update during transfer can be affected not only by the PQ signature but also by the PQC implementation if the update includes the crypto library itself. Banegas et al. [54] show a trade-off among the tested PQC algorithms. Falcon performs well in terms of signature size and verification speed but has the largest implementation size. For larger updates that include the crypto libraries themselves, LMS seems to be the best option since its bigger signature has less effect on the required network bandwidth, while its implementation is relatively small. Additionally, LMS is less demanding on RAM-constrained devices, with moderate RAM usage similar to ECDSA.

As with secure boot, the long lifetime of the key is a concern because the RoT cannot be easily updated. Since Falcon and Dilithium are both lattice-based algorithms with less cryptanalysis done, early adopters may want to deploy hash-based algorithms first, due to their conservative security assumptions. According to the secure boot analysis in Section 4.2.3 and the work by Campos et al. [51], LMS appears to perform better than XMSS on embedded devices and should be the first choice when employing stateful hash-based algorithms.

## 5. Conclusion

This survey provides an overview of research on PQC for embedded devices, with the goal of enabling developers to implement quantum-resistant solutions. In the last year, NIST has selected the first standardization candidates, which provides an initial outlook on what the future of PQC will look like. All the analyzed papers agree that in the future, conventional algorithms will eventually be replaced by PQC. They also demonstrate that for most use cases PQC is already competitive with conventional cryptography and implementation can be started within certain limits. However, most papers indicate that transitioning to PQC is not a straightforward drop-in replacement of algorithms and will come with a cost of higher memory consumption and latency, especially with PQ DSAs. Furthermore, they demonstrate that there is no one-fits-all solution. Instead, most applications involve a trade-off that needs to be evaluated before implementation. Another point of agreement is the careful selection of the PQ DSA based on the lifetime of the public key. Lattice-based algorithms are currently performing well and are at the forefront of standardization. However, they are relatively new and may be affected by advances in cryptanalysis. On the other hand, hash-based algorithms are preferred for applications that rely on a long-term public key due to their well-understood security assumptions and high security level. Hardware limitations can be a significant obstacle in embedded environments, but the analyzed research suggests that for devices similar to the ARM Cortex M4, software implementations are sufficient for all of the investigated use cases. It is important to note that the ARM Cortex M4 is considered a mid-range product in the realm of embedded devices. Therefore, it is unclear if the presented results apply to even more restricted devices in terms of computation power and RAM. Future research will also need to investigate more specific use cases, like VPN connections, remote access, and applications specific to industrial environments. Although protocols like SSH and SSL-VPN are commonly used, they find almost no mention in the current embedded PQC research. Furthermore, the NIST PQC competition is still ongoing and in the upcoming fourth round, at least one more KEM will be standardized. Additionally, NIST issued a new call for proposals to find PQ DSAs based on mathematical problems other than hashes and lattices. Once more candidates are ready for standardization, their implementation into embedded use cases needs to be investigated.

## Acknowledgments

This work was supported by the German Federal Ministry for Economic Affairs and Climate Action (BMWK) under grant no. 13I40V010A, project 'PoQsiKom'.

## References

[1] L. S. Vailshery, "Number of Internet of Things (IoT) connections worldwide from 2022 to 2023, with forecasts from 2024 to 2033," Statista, 2024. [Online]. Available: https://www.statista.com/statistics/1183457/iot-connected-devices-worldwide/

[2] D. Papp, Z. Ma, and L. Buttyan, "Embedded systems security: Threats, vulnerabilities, and attack taxonomy," in *2015 13th Annual Conference on Privacy, Security and Trust (PST)*, 2015, pp. 145–152.

[3] M. Antonakakis, T. April, M. Bailey, M. Bernhard, E. Bursztein, J. Cochran, Z. Durumeric, J. A. Halderman, L. Invernizzi, M. Kallitsis, D. Kumar, C. Lever, Z. Ma, J. Mason, D. Menscher, C. Seaman, N. Sullivan, K. Thomas, and Y. Zhou, "Understanding the mirai botnet," in *26th USENIX Security Symposium (USENIX Security 17)*. Vancouver, BC: USENIX Association, Aug. 2017, pp. 1093–1110. [Online]. Available: https://www.usenix.org/conference/usenixsecurity17/technical-sessions/presentation/antonakakis

[4] P. W. Shor, "Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer," *SIAM Journal on Computing*, vol. 26, no. 5, p. 1484–1509, Oct. 1997. [Online]. Available: http://dx.doi.org/10.1137/S0097539795293172

[5] D. Castelveccihi. (2023) Ibm releases first-ever 1,000-qubit quantum chip. [Online]. Available: https://doi.org/10.1038/d41586-023-03854-1

[6] ——. (2023) Google's quantum computer hits key milestone by reducing errors. [Online]. Available: https://doi.org/10.1038/d41586-023-00536-w

[7] NIST, "Information Technology Laboratory - Computer Security Resource Center: Post-Quantum Cryptography," 2024. [Online]. Available: https://csrc.nist.gov/projects/post-quantum-cryptography

[8] G. Alagic, D. Cooper, Q. Dang, T. Dang, J. M. Kelsey, J. Lichtinger, Y.-K. Liu, C. A. Miller, D. Moody, R. Peralta, R. Perlner, A. Robinson, D. Smith-Tone, and D. Apon, "Status report on the third round of the nist post-quantum cryptography standardization process," 2022-07-05 04:07:00 2022. [Online]. Available: https://tsapps.nist.gov/publication/get_pdf.cfm?pub_id=934458

[9] D.-T. Dam, T.-H. Tran, V.-P. Hoang, C.-K. Pham, and T.-T. Hoang, "A survey of post-quantum cryptography: Start of a new race," *Cryptography*, vol. 7, no. 3, 2023. [Online]. Available: https://www.mdpi.com/2410-387X/7/3/40

[10] B. Koziel, M. M. Kermani, and R. Azarderakhsh, *Post-Quantum Cryptographic Hardware and Embedded Systems*. Cham: Springer International Publishing, 2021, pp. 229–255. [Online]. Available: https://doi.org/10.1007/978-3-030-64448-2_9

[11] N. Alnahawi, A. Wiesmaier, T. Grasmeyer, J. Geißler, A. Zeier, P. Bauspieß, and A. Heinemann, "On the state of post-quantum cryptography migration," INFORMATIK 2021, pp. 907–941, 2021.

[12] N. Alnahawi, N. Schmitt, A. Wiesmaier, and C.-M. Zok, "Towards next generation quantum-safe eids and emrtds – a survey," *ACM Trans. Embed. Comput. Syst.*, mar 2023, just Accepted. [Online]. Available: https://doi.org/10.1145/3585517

[13] K.-A. Shim, "A survey on post-quantum public-key signature schemes for secure vehicular communications," *IEEE Transactions on Intelligent Transportation Systems*, vol. 23, no. 9, pp. 14 025–14 042, 2022.

[14] T. M. Fernández-Caramés, "From pre-quantum to post-quantum iot security: A survey on quantum-resistant cryptosystems for the internet of things," *IEEE Internet of Things Journal*, vol. 7, no. 7, pp. 6457–6480, 2020.

[15] N. Alnahawi, J. Müller, J. Oupický, and A. Wiesmaier, "A comprehensive survey on post-quantum tls," *IACR Communications in Cryptology (IACR CiC)*, vol. 1, 2024.

[16] National Institute of Standards and Technology, "Submission requirements and evaluation criteria for the post-quantum cryptography standardization process," 2016-06-24 2016. [Online]. Available: https://csrc.nist.gov/CSRC/media/Projects/Post-Quantum-Cryptography/documents/call-for-proposals-final-dec-2016.pdf

[17] L. K. Grover, "A fast quantum mechanical algorithm for database search," 1996.

[18] G. Brassard, P. HØyer, and A. Tapp, *Quantum cryptanalysis of hash and claw-free functions: Invited paper*. Springer Berlin Heidelberg, 1998, p. 163–169. [Online]. Available: http://dx.doi.org/10.1007/BFb0054319

[19] E. Barker, L. Chen, and R. Davis, "Recommendation for key-derivation methods in key-establishment schemes," 2020-08-18 2020.

[20] F. D, "Terminology for Post-Quantum Traditional Hybrid Schemes," Internet-Draft draft-ietf-pquip-pqt-hybrid-terminology-02, Feb. 2024, work in Progress. [Online]. Available: https://datatracker.ietf.org/doc/draft-ietf-pquip-pqt-hybrid-terminology/02/

[21] D. McGrew, M. Curcio, and S. Fluhrer, "Leighton-Micali Hash-Based Signatures," RFC 8554, Apr. 2019.

[22] A. Huelsing, D. Butin, S.-L. Gazdag, J. Rijneveld, and A. Mohaisen, "XMSS: eXtended Merkle Signature Scheme," RFC 8391, May 2018. [Online]. Available: https://www.rfc-editor.org/info/rfc8391

[23] D. Cooper, D. Apon, Q. Dang, M. Davidson, M. Dworkin, and C. Miller, "Recommendation for stateful hash-based signature schemes," 2020-10-29 00:10:00 2020.

[24] D. J. Bernstein, A. Hülsing, S. Kölbl, R. Niederhagen, J. Rijneveld, and P. Schwabe, "The sphincs+ signature framework," Cryptology ePrint Archive, Paper 2019/1086, 2019, https://eprint.iacr.org/2019/1086. [Online]. Available: https://eprint.iacr.org/2019/1086

[25] National Institute of Standards and Technology, "Stateless hash-based digital signature standard," 2023-08-24 2023.

[26] A. Wagner, F. Oberhansl, and M. Schink, "To be, or not to be stateful: Post-quantum secure boot using hash-based signatures," in *Proceedings of the 2022 Workshop on Attacks and Solutions in Hardware Security*, ser. ASHES'22. New York, NY, USA: Association for Computing Machinery, 2022, p. 85–94. [Online]. Available: https://doi.org/10.1145/3560834.3563831

[27] P. Karl, J. Schupp, and G. Sigl, "The impact of hash primitives and communication overhead for hardware-accelerated sphincs+," Cryptology ePrint Archive, Paper 2023/1767, 2023, https://eprint.iacr.org/2023/1767. [Online]. Available: https://eprint.iacr.org/2023/1767

[28] M. Ajtai, "Generating hard instances of lattice problems (extended abstract)," in *Proceedings of the Twenty-Eighth Annual ACM Symposium on Theory of Computing*, ser. STOC '96. New York, NY, USA: Association for Computing Machinery, 1996, p. 99–108. [Online]. Available: https://doi.org/10.1145/237814.237838

[29] O. Regev, "On lattices, learning with errors, random linear codes, and cryptography," in *Proceedings of the Thirty-Seventh Annual ACM Symposium on Theory of Computing*, ser. STOC '05. New York, NY, USA: Association for Computing Machinery, 2005, p. 84–93. [Online]. Available: https://doi.org/10.1145/1060590.1060603

[30] National Institute of Standards and Technology, "Module-lattice-based key-encapsulation mechanism standard," 2023-08-24 2023.

[31] ——, "Module-lattice-based digital signature standard," 2023-08-24 2023.

[32] V. Lyubashevsky, "Fiat-shamir with aborts: Applications to lattice and factoring-based signatures," in *Advances in Cryptology – ASIACRYPT 2009*, M. Matsui, Ed. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009, pp. 598–616.

[33] P.-A. Fouque, J. Hoffstein, P. Kirchner, V. Lyubashevsky, T. Pornin, T. Prest, T. Ricosset, G. Seiler, W. Whyte, and Z. Zhang, "Falcon: Fast-fourier lattice-based compact signatures over ntru," 2019. [Online]. Available: https://api.semanticscholar.org/CorpusID:231637439

[34] L. Ducas and T. Prest, "Fast fourier orthogonalization," Cryptology ePrint Archive, Paper 2015/1014, 2015, https://eprint.iacr.org/2015/1014. [Online]. Available: https://eprint.iacr.org/2015/1014

[35] M. J. Kannwischer, R. Petri, J. Rijneveld, P. Schwabe, and K. Stoffelen, "PQM4: Post-quantum crypto library for the ARM Cortex-M4," https://github.com/mupq/pqm4.

[36] D. Moody. (2019) Nist focus on the cortex m4, slide 22. [Online]. Available: https://csrc.nist.gov/CSRC/media/Presentations/the-2nd-round-of-the-nist-pqc-standardization-proc/images-media/moody-opening-remarks.pdf

[37] M. Schöffel, F. Lauer, C. C. Rheinländer, and N. Wehn, "Secure iot in the era of quantum computers-where are the bottlenecks?" *Sensors*, vol. 22, no. 7, 2022. [Online]. Available: https://www.mdpi.com/1424-8220/22/7/2484

[38] E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.3," RFC 8446, Aug. 2018.

[39] G. Tasopoulos, J. Li, A. P. Fournaris, R. K. Zhao, A. Sakzad, and R. Steinfeld, "Performance evaluation of post-quantum tls 1.3 on resource-constrained embedded systems," in *Information Security Practice and Experience*, C. Su, D. Gritzalis, and V. Piuri, Eds. Cham: Springer International Publishing, 2022, pp. 432–451.

[40] K. Bürstinghaus-Steinbach, C. Krauß, R. Niederhagen, and M. Schneider, "Post-quantum tls on embedded systems," Cryptology ePrint Archive, Paper 2020/308, 2020, https://eprint.iacr.org/2020/308. [Online]. Available: https://eprint.iacr.org/2020/308

[41] G. Tasopoulos, C. Dimopoulos, A. P. Fournaris, R. K. Zhao, A. Sakzad, and R. Steinfeld, "Energy consumption evaluation of post-quantum tls 1.3 for resource-constrained embedded devices," *International Conference on Computing Frontiers (CF)*, vol. 20th, 2023.

[42] R. Gonzalez and T. Wiggers, "Kemtls vs. post-quantum tls: Performance on embedded systems," in *Security, Privacy, and Applied Cryptography Engineering*, L. Batina, S. Picek, and M. Mondal, Eds. Cham: Springer Nature Switzerland, 2022, pp. 99–117.

[43] P. Schwabe, D. Stebila, and T. Wiggers, "Post-quantum TLS without handshake signatures," in *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '20. New York, NY, USA: Association for Computing Machinery, 2020, p. 1461–1480. [Online]. Available: https://kemtls.org/publication/kemtls/

[44] W. Beullens, "Breaking rainbow takes a weekend on a laptop," in *Advances in Cryptology – CRYPTO 2022*, Y. Dodis and T. Shrimpton, Eds. Cham: Springer Nature Switzerland, 2022, pp. 464–479.

[45] V. B. Y. Kumar, N. Gupta, A. Chattopadhyay, M. Kasper, C. Krauß, and R. Niederhagen, "Post-quantum secure boot," in *2020 Design, Automation & Test in Europe Conference & Exhibition*, 2020, pp. 1582–1585.

[46] S. Marzougui and J. Krämer, "Post-quantum cryptography in embedded systems," in *Proceedings of the 14th International Conference on Availability, Reliability and Security*, ser. ARES '19. New York, NY, USA: Association for Computing Machinery, 2019. [Online]. Available: https://doi.org/10.1145/3339252.3341475

[47] J. W. Bos, A. Hülsing, J. Renes, and C. van Vredendaal, "Rapidly verifiable xmss signatures," Cryptology ePrint Archive, Paper 2020/898, 2020, https://eprint.iacr.org/2020/898. [Online]. Available: https://eprint.iacr.org/2020/898

[48] P. Kampanakis, P. Panburana, M. Curcio, C. Shroff, and M. M. Alam, "Post-quantum lms and sphincs+ hash-based signatures for uefi secure boot," Cryptology ePrint Archive, Paper 2021/041, 2021, https://eprint.iacr.org/2021/041. [Online]. Available: https://eprint.iacr.org/2021/041

[49] R. Román and I. Baturone, "A quantum-resistant and fast secure boot for iot devices using hash-based signatures and sram pufs," in *The Fifth International Conference on Safety and Security with IoT*, A. Nayyar, A. Paul, and S. Tanwar, Eds. Cham: Springer International Publishing, 2023, pp. 121–136.

[50] D. McGrew, P. Kampanakis, S. Fluhrer, S.-L. Gazdag, D. Butin, and J. Buchmann, "State management for hash-based signatures," in *Security Standardisation Research*, L. Chen, D. McGrew, and C. Mitchell, Eds. Cham: Springer International Publishing, 2016, pp. 244–260.

[51] F. Campos, T. Kohlstadt, S. Reith, and M. Stöttinger, "Lms vs xmss: Comparison of stateful hash-based signature schemes on arm cortex-m4," in *Progress in Cryptology - AFRICACRYPT 2020*, A. Nitaj and A. Youssef, Eds. Cham: Springer International Publishing, 2020, pp. 258–277.

[52] B. Moran, H. Tschofenig, D. Brown, and M. Meriac, "A Firmware Update Architecture for Internet of Things," RFC 9019, Apr. 2021. [Online]. Available: https://www.rfc-editor.org/info/rfc9019

[53] B. Moran, H. Tschofenig, and H. Birkholz, "A Manifest Information Model for Firmware Updates in Internet of Things (IoT) Devices," RFC 9124, Jan. 2022. [Online]. Available: https://www.rfc-editor.org/info/rfc9124

[54] G. Banegas, K. Zandberg, E. Baccelli, A. Herrmann, and B. Smith, "Quantum-resistant software update security on low-power networked embedded devices," in *Applied Cryptography and Network Security*, G. Ateniese and D. Venturi, Eds. Cham: Springer International Publishing, 2022, pp. 872–891.

[55] B. Halak, T. Gibson, M. Henley, C.-B. Botea, B. Heath, and S. Khan, "Evaluation of performance, energy, and computation costs of quantum-attack resilient encryption algorithms for embedded devices," *IEEE Access*, vol. 12, pp. 8791–8805, 2024.

## Appendix

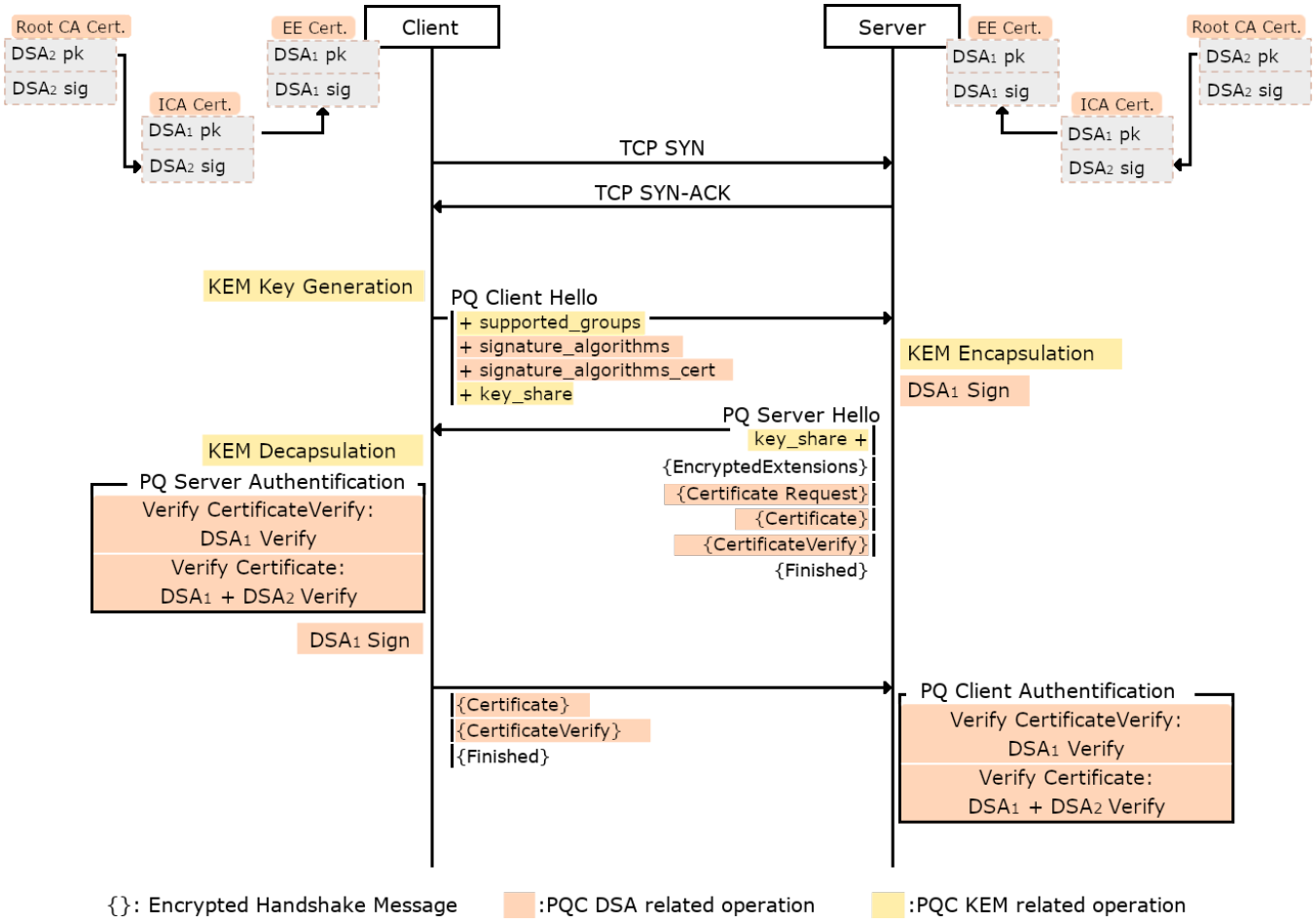


Figure 3. Full PQ-TLS Handshake, including en exemplary PKI structure for client and server certificates

Table 7. PQ TLS PAPERS

| Reference | Approach | Network Topology | PKI Structure | Main Findings |
|---|---|---|---|---|
| Bürstinghaus et al. [40] | Prove feasibility of PQ TLS 1.2 on different embedded devices as client and server with Sphincs and Kyber | Ethernet, direct Connection | Single pre-shared certificate - Server authentication | - Full PQ TLS 1.2 is possible on different embedded platforms<br>- Kyber can outperform ECDH<br>- Sphincs signing process creates bottleneck |
| Tasopoulos et al. [39] | Benchmark of PQ TLS 1.3 on ARM Cortex M4 with all NIST round 3 candidates | Ethernet, direct Connection | Homogeneous two-layered hierarchy - Mutual authentication | - Choice of DSA more impactful than choice of KEM<br>- Dilithium-Kyber combination can outperform conventional TLS with elliptic curves<br>- Handshake bandwidth increases by factor 5-10 depending on algorithm combination |
| Tasopoulos et al. [41] | Energy Consumption Evaluation of Post-Quantum TLS 1.3 for Resource-Constrained Embedded Devices | Ethernet, direct | hierarchy unstated, Mutual and server authentication | - PQC can offer equal or greater efficiency compared to traditional TLS ciphers<br>- power consuption varies with ciphers and scenarios |
| Schöffel et al. [37] | Benchmark PQ TLS 1.2 on ARM Cortex M4 with all NIST round 3 candidates | Bluetooth Low Energy, Connection over Gateway | Heterogeneous two-layered hierarchy - Mutual authentication | - Bandwidth-restricted environments suffer from large public keys and signatures from PQ DSAs<br>- Heterogeneous PKIs can reduce bandwidth and runtime<br>- Falcon-Kyber combination provides fastest handshake performance |
| Gonzales et al. [42] | Comparison of KEMTLS 1.3 to regular PQ TLS on ARM Cortex M4 | Varies see Table 3 | Heterogeneous two-layered hierarchy - Server authentication | - In low-bandwidth networks, DSA is the main factor for latency due to public key and signature size, otherwise PQC computation is the bottleneck<br>- KEMTLS can not outperform Falcon with server authentication only |
| Halac et al. [55] | Evaluation of Performance, Energy, and Computation Costs of Quantum-Attack Resilient Encryption Algorithms for Embedded Devices | Ethernet, direct Connection | hierarchy unstated, Mutual authentication | - Effects are mostly defined by the DSA<br>- Latency compares well to Elliptic Curve Cryptography (ECC)<br>- Memory usage varies heavily between algorithms and stack or heap utilization<br>- Dilithium |
| Alnahawi et al. [15] | Survey on Post-Quantum TLS | Varies | Server authentication | - Both purely and hybrid PQ algorithms are competitive with traditional ciphers<br>- All proposals for PQ key exchange use KEMs<br>- Especially KEMTLS shows promise in performance and possible scenarios<br>- Findings vary in non-ideal network conditions |