Quantum-Resistant Security for Software Updates on Low-power Networked Embedded Devices

Gustavo Banegas*
gustavo@cryptme.in
Inria and Laboratoire d'Informatique de l'École Polytechnique, Institut Polytechnique de Paris
Palaiseau, France

Koen Zandberg*
koen.zandberg@inria.fr
Inria
Saclay, France

Adrian Herrmann
adrian.herrmann@fu-berlin.de
Freie Universität Berlin
Berlin, Germany

Emmanuel Baccelli
emmanuel.baccelli@inria.fr
Inria
Saclay, France

Benjamin Smith
smith@lix.polytechnique.fr
Inria and Laboratoire d'Informatique de l'École Polytechnique, Institut Polytechnique de Paris
Palaiseau, France

ABSTRACT

As the Internet of Things (IoT) rolls out today to devices whose lifetime may well exceed a decade, conservative threat models should consider attackers with access to quantum computing power. The SUIT standard (specified by the IETF) defines a security architecture for IoT software updates, standardizing the metadata and the cryptographic tools—namely, digital signatures and hash functions—that guarantee the legitimacy of software updates. SUIT performance has been evaluated in the pre-quantum context, but it has not yet been studied in a post-quantum context. Taking the open-source implementation of SUIT available in RIOT as a case study, we overview post-quantum considerations, and quantum-resistant digital signatures in particular, focusing on low-power, microcontroller-based IoT devices which have stringent resource constraints in terms of memory, CPU, and energy consumption. We benchmark a selection of proposed post-quantum signature schemes (LMS, Falcon, and Dilithium) and compare them with current pre-quantum signature schemes (Ed25519 and ECDSA). Our benchmarks are carried out on a variety of IoT hardware including ARM Cortex-M, RISC-V, and Espressif (ESP32), which form the bulk of modern 32-bit microcontroller architectures. We interpret our benchmark results in the context of SUIT, and estimate the real-world impact of post-quantum alternatives for a range of typical software update categories.

CCS CONCEPTS

• **Computer systems organization → Embedded systems**.

KEYWORDS

Post-quantum, Security, IoT, Microcontroller, Embedded systems

1 INTRODUCTION

Decades of experience with the Internet and networked software has brought about the motto *you can't secure what you can't update*. Meanwhile, recent technological and societal trends have fuelled the massive deployment of cyberphysical systems; these systems are increasingly pervasive, and we are increasingly dependent on their functionalities. A so-called Internet of Things (IoT) emerges, weaving together an extremely wide variety of machines (embedded software and hardware) which are required to cooperate via the network, at large scale.

Unpatched devices—or worse, unpatchable devices—quickly become liabilities. Exploits weaponizing compromised IoT devices are demonstrated time and again, sometimes spectacularly as with botnets such as Mirai [2, 9]. The twist is, however, that the cure can become a disease: software updates are themselves an attack vector. Attacks can consist in lacing a legitimate software update with malware, compromising the updated device in effect [53]. Once IoT devices are deployed, up and running, it thus becomes crucial to have answers ready to the following questions:

- How, and when, is software embedded in IoT devices updated?
- How are software updates secured, and what level of security is provided?

In this paper we tackle these questions, focusing on low-power IoT devices, and anticipating adversaries equipped with quantum computers.

Low-power IoT characteristics. A prominent and particularly challenging component of IoT deployments consists in integrating low-power, resource-constrained IoT devices into the distributed system. These devices are typically based on low-cost microcontrollers (e.g., ARM Cortex M, RISC-V, ESP), which interconnect via a low-power radio (e.g., BLE, IEEE 802.15.4, LoRa) or via a wired communication bus. An estimated 250 billion microcontrollers are in use today around the globe [36]. Compared to microprocessor-based devices, microcontrollers aim for a different trade-off: They offer much smaller capacity in computing, networking, memory [20], in order to achieve radically lower energy consumption and a tiny price tag (<1$ unit price). To give an idea, it is not uncommon to have a memory budget of 64 kB of RAM and 500 kB of ROM (flash) in total for the whole embedded system software—including drivers, crypto libraries, OS kernel, network stack and application

*These authors contributed equally to this work.

logic. Nonetheless, the functionalities and services provided by constrained microcontroller-based devices are as crucial as those of other, less constrained elements in the cyberphysical system.

Quantum adversaries. Robust and commoditized quantum computers still sound futuristic today. While it would be hazardous to predict the imminence of a breakthrough, progress in this domain has picked up substantially. Among others, prominent Big Tech (including Google, IBM, Intel, Microsoft) have already been designing, building and operating small quantum computers over the last years. Currently, the capacity of existing quantum computers are being incrementally enhanced with more quantum bits, which steadily improves their performance. As the lifetime of IoT devices rolled out today can largely exceed a decade, conservative threat models should consider attackers which benefit from quantum-computing power, on top of traditional computing power.

Post-quantum cryptography. Post-quantum cryptosystems (and in particular, post-quantum signature schemes) are designed to run on contemporary hardware, yet resist adversaries equipped with both classical and quantum computers. There are many signature schemes that claim post-quantum security, some old and some new, but until now none has seen wide deployment. Recent international research on post-quantum schemes has revolved around the National Institute of Standards and Technology (NIST) Post-Quantum Cryptography project [54], which aims to distinguish a limited number of candidate schemes for eventual standardization. This process is currently in its third round, and draft standards are expected between 2022 and 2024.

Post-quantum security for low-power IoT.. Let's get back to the motto *you can't secure what you can't update (securely).* In our quest for post-quantum security, a first order priority is to guarantee in a future-proof manner that software updates received via the network on low-power IoT devices are legitimate. When verifying the legitimacy of a software update, the crucial cryptographic tool is a digital signature. Open standards targeting IoT security (such as the IETF [59]) specify the safe usage of a variety of digital signature schemes to secure software updates on low-power devices, including one scheme (LMS [48]) that offers quantum resistance.

Implementation approaches. It is common to develop cryptographic implementations that tackle specific problems, such as speed or size. Most of the time, the implementation tries to take advantage of special instructions or hardware. However, that narrows the applicability of the implementation to specific architectures, which does not fully reflect the reality of IoT. Usually, operating systems (OS) need to support more than one architecture.

Typically, new cryptographic algorithm implementations are demonstrated as stand-alone applications—a key first step in proving feasibility. But in practice, the OS does not have only the cryptography package: it has other modules, a network stack, and the kernel.

In this paper, we present a use case for cryptographic signatures, and we study the impact of the pre-quantum to post-quantum transition. Focusing on portability and wide deployment, we did not use any "enhanced" implementations such as tuned assembly, or platform-specific instructions: we only modified the implementations to fit real-life conditions, such as those imposed by RIOT for our use-case (for example: not dedicating the entire stack to crypto).

In this context, we aim to answer the following questions in our paper:

- *How do post-quantum security costs compare to typical pre-quantum security costs?*
- *What is the footprint of quantum resistance security, relative to typical low-power operating system footprints?*
- *What are the potential alternatives for post-quantum digital signature schemes to secure IoT software updates?*
- *What hash functions should be used in this context?*

We will address these questions under the assumption that we want to achieve, and maintain, 128-bit conventional security (matching current internet security standards) and NIST Level 1 post-quantum security.

Paper contributions & outline. The main contributions of this paper are:

- We provide an overview of the SUIT specification for secure software updates on low-power IoT devices, using its open-source implementation in a common operating system (RIOT) as case study;
- We show how crypto primitives including digital signatures and hash functions are used in compliance with SUIT;
- We analyze post-quantum considerations for SUIT-compliant hash functions, which we benchmark on low-power 32-bit microcontrollers;
- We survey post-quantum signature schemes, and derive a selection of schemes most applicable for the secure IoT software update use case;
- We benchmark signature schemes on heterogeneous low-power IoT hardware based on popular 32-bit microcontrollers (ARM Cortex-M, RISC-V and ESP32);
- We compare the performance of post-quantum signature schemes (LMS, Dilithium, and Falcon) against that of typical pre-quantum schemes (Ed25519, and secp256);
- We conclude on the cost of post-quantum security, and outline perspectives for low-power IoT.

2 RELATED WORK

The performance of pre-quantum digital signature schemes in the context of secure software updates on various Cortex-M microcontrollers is evaluated in [62]. Various NIST candidate post-quantum schemes are compared as component algorithms in TLS 1.3 in [58], analyzing performance, security, and key and signature sizes, as well as the impact of post-quantum authentication on TLS 1.3 handshakes in realistic network conditions, while [45] shows a real life experiment with clients using two post-quantum schemes: an isogeny-based algorithm (SIKE) and a lattice-based algorithm (HRSS). More recently, another experiment with different schemes was conducted by Cloudflare [23, 56].

For pure post-quantum cryptographic implementation work targeting microcontrollers, [21] evaluates the performance of stateful LMS on Cortex-M4 microcontrollers, while `pqm4` [42] aims to implement and benchmark NIST candidate schemes on Cortex-M4,

with M4 assembly subroutines plugged into some of the PQClean implementations. (Note that among the NIST candidate signature schemes, PQClean implements only Dilithium, Falcon, Rainbow, and SPHINCS+; of these, pqm4 implements only Dilithium and Falcon.) Software verifying SPHINCS, RainbowI, GEMSS, Dilithium2, and Falcon-512 signatures in Cortex-M3 using less than 8 kB of RAM is presented in [33].

Many post-quantum digital signature schemes use SHA3 hashing primitives under the hood. While the performance of SHA3 implementations in hardware (FGPA) have been studied in work such as [34, 41, 43], surprisingly few studies focus on the performance of software implementations of this standard hashing primitive on low-power microcontrollers. Some prior work exists in this domain with particular focus on 8-bit microcontrollers [13, 44]. Another prior study has focused on comparing the performance of Keccak variants on 32-bit ARM Cortex-M microcontrollers in [35].

3 CASE STUDY: LOW-POWER SOFTWARE UPDATES WITH SUIT

The IETF standardizes the Software Updates for Internet of Things [50, 51] specifications (SUIT), which define a security architecture, standard metadata and cryptographic schemes able to secure IoT software updates, applicable on microcontroller-based devices. An open-source implementation of the SUIT workflow is for example available in RIOT [61], a common operating system for low-power IoT devices [12] which we use as base for our case study.

3.1 SUIT Workflow

The SUIT workflow is shown in Fig. 1. This workflow consists of a preliminary phase (*Phase 0*) whereby the authorized maintainer produces and flashes the IoT devices with commissioning material: the bootloader, the initial image, and authorized crypto material.

Once the IoT device is commissioned, up and running, the maintainer can trigger iterations through cycles of phases 1-5, whereby the authorized maintainer can build a new image (*Phase 1*), hash and sign the corresponding standard metadata (the so-called SUIT manifest, *Phase 2*) and transfer to the device over the network via a repository (e.g. a CoAP resource directory). The IoT device can then fetch the update and the SUIT manifest from the repository (*Phase 3*), proceed to verify the signature and the hash (*Phase 4*), and upon successful verification, the new software is installed and booted (*Phase 5*), otherwise the update is dropped.

Using the metadata and the cryptographic primitives as specified by SUIT, the IoT device can mitigate attacks using software updates as vector, such as:

- **Tampered/Unauthorized Firmware Update Attacks:** An attacker may try to update the IoT device with a modified and intentionally flawed firmware image. To counter this threat, SUIT specifies the use of digital signatures on a hash of the image binary and the metadata to ensure integrity of both the firmware and its metadata.
- **Firmware Update Replay Attacks:** An attacker may try to replay a valid, but old (known-to-be-flawed) firmware. This threat is mitigated by using a sequence number used in the metadata, which is increased with every new firmware update.
- **Firmware Update Mismatch Attacks:** An attacker may try replaying a firmware update that is authentic, but for an incompatible device. To counter this threat, SUIT specifies the inclusion of device-specific conditions, which can be verified before installing a firmware image, thereby preventing the device from using an incompatible firmware image.

For a more complete documentation of attacks that are mitigated with SUIT we refer readers to [49].

SUIT Cryptographic Tools. As depicted in Fig. 1, cryptographic tools on which software updates in general and SUIT in particular rely are a digital signature scheme and a hash function. On the one hand, the digital signature authenticates the software update binary. On the other hand, to make the signature verification less cumbersome, the signature is not performed on the software update binary itself, but on a hash of the software update binary. Thus, this hash function is also a crucial cryptographic primitive in the SUIT workflow. Fig. 1 depicts this workflow combining SHA-256 hashing and Ed25519 signatures.

3.2 Hash Functions with SUIT

The metadata of the update (specified by the SUIT Manifest [50]) includes a cryptographic hash of the sofware update binary. To be considered secure, a cryptographic hash function $H : \{0,1\}^* \rightarrow \{0,1\}^l$, where $l > 0$, must have the following properties:

- Preimage resistance: Given a hash value h, it should be infeasible to find any input m such that $h = H(m)$;
- Second preimage resistance: Given an input m_1, it should be infeasible to find a different input m_2 such that $H(m_1) = H(m_2)$;
- Collision resistance: It should be infeasible to find two different inputs m_1 and m_2 such that $H(m_1) = H(m_2)$.

The SUIT standard specification [50] allows for the use of the following hash functions:

- SHA-2: either 224-, 256-, 384-, or 512-bit output;
- SHA-3: either 224-, 256-, 384-, or 512-bit output.

Background on SHA-2. This algorithm is a well-known secure hash function developed in the early 90's and standardized in 2001 by NIST in [1]. SHA-2 is based on the Merkle-Damgård construction, and executes several rounds of a compression function. SHA-2 presents 4 different digest sizes (i.e., output lengths): 224, 256, 384, and 512 bits. SHA-2 is widely used in several applications, including TLS, SSL, PGP, SSH, and many others. The main reason for this is that it is a stable and secure function: The best known attacks against SHA-2 break preimage resistance for 52 out of 64 rounds (for SHA-256), and 57 out of 80 rounds (for SHA-512). For collision resistance, the only attack is against 46 out of 64 rounds of SHA-256.

Background on SHA-3. This hash algorithm was standardized in 2015 by NIST in FIPS 202 [30]. SHA-3's basic primitive is called Keccak [18], which is built using the sponge construction (differing from SHA-2's Merlke–Damgård construction). This gives SHA-3 the advantage of resisting known attacks on Merkle–Damgård hash functions (like SHA-2). Like SHA-2, SHA-3 presents 4 different digest sizes: 224, 256, 384, and 512 bits. Sponge-based functions can "absorb" any amount of data, and then "squeeze" any amount of

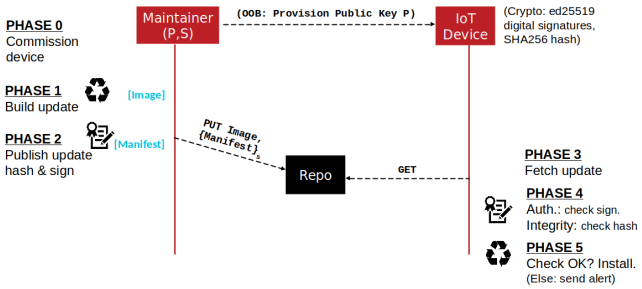

Figure 1: SUIT secure software update workflow.

output, thus providing an extendable output function (XOF) that can be used for purposes beyond just hashing. The Keccak-based XOF specified in FIPS 202 is called SHAKE.

3.2.1 *Post-Quantum Considerations.*

There are few quantum attacks against SHA-2 and SHA-3 in literature. Recently, [14] showed that it is possible to parallelize Grover's algorithm to find preimages of hash functions, and this attack applies to both Merkle–Damgård hashes (e.g. SHA-2) and Sponge-based hashes (e.g. SHA-3). For collision resistance, the state-of-the-art in quantum collision search does not drastically reduce the complexity with respect to classical algorithms, as shown in [24]. On the other hand, classical attacks for SHA-2 might become a reality, as shown in [29].

3.2.2 *Low-Power IoT Considerations.*

Low-power systems should be able to run hash functions in the smallest amount of time and using as little power as possible. Minimal memory (RAM and flash) usage are also desirable. In this context, since we aim for 128-bit security, the two functions we should consider for SUIT are SHA-256 and SHA3-256.

Table 1 shows the amount of flash memory taken up by RIOT's default implementation of SHA-256, and compares it with the footprint of two different SHA3-256 implementations: one optimized for flash memory, and the other optimized for speed on an ARM Cortex-M4 microcontroller (ARMv7M architectures). Next, Figure 2 shows the execution speed of hash operations using these different implementations on an ARM Cortex M4 microcontroller. Finally, Figure 3 compares the RAM (stack) memory used by these implementations. We observe that RAM usage is roughly equivalent across the different implementations, but speed and flash can vary widely for the SHA-3 implementations. Basically, SHA3-256 can offer slightly faster execution compared to SHA-256, but at the price of a 10× larger flash footprint. For a flash footprint similar to SHA-256, the comparative speed of SHA3-256 diminishes drastically for larger inputs. For more detailed analysis, we refer readers to a previous study [35] comparing different Keccak variants on microcontrollers.

3.2.3 *Conclusions on SUIT Hash Functions for Post-Quantum.*

Based on our analysis, there are no *direct* post-quantum aspects to consider here. Rather, the decision of which hash function to use should be driven by low-power criteria, and by other *indirect* post-quantum aspects detailed below.

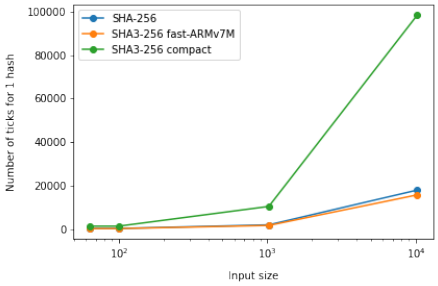

Figure 2: Execution speed of SHA2 and SHA3, versus input size in bytes, on an ARM Cortex-M4 microcontroller.

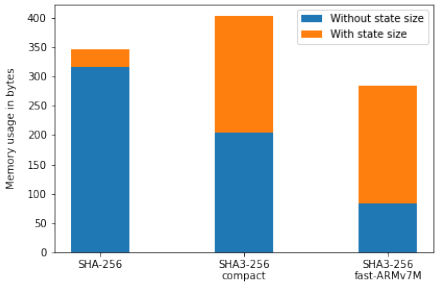

Figure 3: RAM (stack) usage of SHA2 and SHA3 on an ARM Cortex-M4 microcontroller.

Table 1: ROM (flash memory) footprint of SHA2 and SHA3 on an ARM Cortex-M4 microcontroller.

SHA-256	1008 bytes
SHA3-256 compact	1692 bytes
SHA3-256 fast-ARMv7M	11548 bytes

Let's distinguish broad categories for low-power IoT software updates:

(1) Software module update (~5kB)
(2) Small firmware update without crypto (~50kB)
(3) Small firmware update with crypto (~50kB)
(4) Large firmware update (~250kB)

In cases (1) and (2), the updated software does not include the hash function implementation (the cryptographic tools are external, e.g., in a bootloader). In such cases, the flash memory footprint overhead for the hash function in use is of no concern, and SHA3-256 (optimized for speed) is the best choice. In cases (2) and (3), however, the updated software includes the cryptographic tools and the hash function code; thus, a tradeoff appears. For small firmware updates, a 10 kB flash overhead represents a significant 25% bump in what needs to be stored on the device and transmitted over the network. As updates are infrequent, execution speed may be considered less of a priority, and thus both SHA-256 and flash-optimized SHA3-256 are valid options. For larger updates, the storage and transfer overhead is negligible, so speed-optimized SHA3-256 is the best option again.

Let us now consider a complementary perspective: We observe that most of the quantum-resistant digital signature schemes use SHA-3 in their constructions. In fact, the NIST competition candidates for the upcoming post-quantum signature standard are required to be SHA-3/SHAKE compatible, because that is the current US standard. In this respect, as code footprint on IoT devices is very limited, factorization is typically desirable: There is an opportunity to implement a single hash function (used both for hashing and for signing) in order to use less flash memory.

For these reasons, we SHA3-256 is the primary choice in our case-study.

3.3 Digital Signatures with SUIT

The SUIT architecture relies on the software update distributor (i.e., the authorized maintainer in Figure 1) issuing a long-term public-private key pair, and the public key being pre-installed on the IoT device(s) to be updated, during the commissioning (*Phase 0*) shown in Fig. 1). This key pair is used to generate and verify a digital signature on the IoT software update. Digital signature use in SUIT is specified in the COSE standard [55], which defines how to sign and encrypt compact (CBOR) binary serialized objects. COSE standardizes the use of elliptic-curve digital signature schemes using the following state-of-the-art elliptic curves:

- Curve25519 (Ed25519), Curve448 (Ed448);
- NIST P-256, P-384, P-521.

These elliptic-curve schemes are desirable as they offer very small public (and private) keys at 32 bytes each and 64-byte signatures. To give some concrete perspective, Figure 4 shows the memory footprint of SUIT and related software components when using Ed25519, compared to the whole software embedded on the IoT device. For this measurement we used the available open-source RIOT implementation, which we build for, and run on a popular low-power IoT board based on an ARM Cortex-M4 microcontroller (the Nordic nRF52840 Development Kit). The flash memory footprint of this firmware is 52.5 kB and the RAM (stack) usage is 16.3 kB.

In particular, we observe that in this typical pre-quantum configuration, out of a $\approx$ 50 kB total flash footprint, the crypto represents a minor part of the footprint (under 15%). Furthermore, the SUIT manifest itself can remain small, because the elliptic-curve signature adds 15% to the size of the SUIT manifest metadata and less than

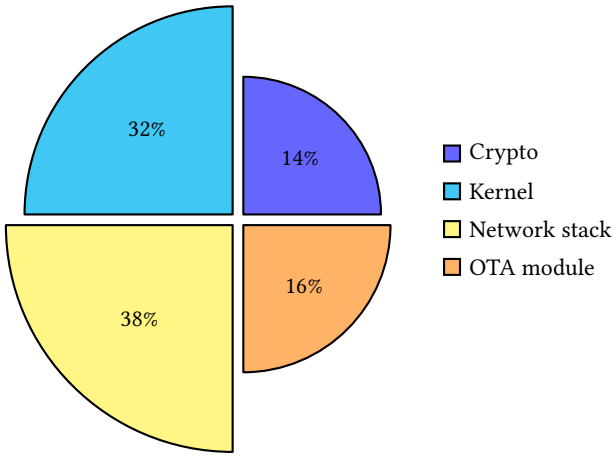

Figure 4: Flash memory compostion of SUIT-enabled RIOT firmware with typical pre-quantum configuration using Ed25519 signatures and SHA-256 hashing (total footprint 52.5 kB).

0.1% to the data that must be transferred over the network, counting the manifest and the firmware binary as depicted in Table 2.

Table 2: SUIT firmware update network transfer cost, using minimal metadata, Ed25519 signatures, SHA-256 hashing.

	Network Payload
SUIT metadata	419 bytes
SUIT signature	64 bytes
OS firmware	52485 bytes

3.3.1 *Post-Quantum Considerations.*

Elliptic-curve schemes are advantageous because they provide high security guarantees even though keys and signatures are very small. However, the security of elliptic-curve signatures is guaranteed by the hardness of the elliptic-curve Discrete Logarithm Problem, which can be solved efficiently on large quantum computers using Shor's algorithm [15, 37, 57].

It is important to note that a breakthrough in quantum computing at a time T will not affect the security of elliptic-curve signatures generated before T, but it would certainly destroy the security of any elliptic-curve signatures generated after T. In our use case, the distributor's key pair has a very long planned lifetime, possibly equal to that of the devices to be updated; securely updating the key itself will be impossible, or at least undesirable. We therefore need to build-in resistance to the quantum threat in anticipation of such a development.

3.3.2 *Low-Power IoT Considerations.*

The range of post-quantum signature schemes considered as potential replacements for elliptic-curve signatures is wide and diverse, and the idiosyncrasies that distinguish the various schemes are exaggerated by the constraints of low-power IoT devices. However, all of these schemes have public key and signature sizes that

are one or two orders of magnitude larger than the elliptic-curve equivalents. Post-quantum signatures are therefore far from drop-in replacements; they represent a significant research challenge for microcontroller and IoT implementations.

Nevertheless, the IETF recently initiated the standardization of alternative signature schemes with COSE/SUIT which aim to offer quantum-resistant security levels, such as LMS [48]. In the next sections, we survey alternative quantum-resistant schemes, and give an experimental comparison of their performance against that of state-of-the-art pre-quantum digital signature schemes as used in SUIT.

4 POST-QUANTUM DIGITAL SIGNATURES

The signature schemes that we consider target at least NIST Level 1 for *post-quantum security*. This is the basic security level proposed by NIST as part of its Post-Quantum Cryptography (PQC) Standardization Project [54]. Level 1 security includes both 128 bits of classical security, and an equivalent level of security with respect to some model of quantum computation. That is, an adversary should require on the order of 2^{128} operations to gain any non-negligible advantage when attacking the scheme, even if this adversary benefits from quantum computing power. For example, this is the amount of required work for an adversary to have any chance of forging a signature on a new message, under a given public key. This 128-bit security level is now standard in mainstream internet applications requiring long-term security.

4.1 Post-quantum signature paradigms

Post-quantum signatures, like other post-quantum protocols, form natural families according to the sources of underlying hard problems that guarantee their security:

Hash-based signatures. Hash-based signatures are among the oldest digital signature schemes. Their security is based on the difficulty of inverting cryptographic hash functions. The security assumptions have been well studied, which gives an academic maturity to the problem. Important contemporary examples include XMSS [38], LMS [48], and the NIST Round 3 alternatate candidate SPHINCS+ [10]. Hash-based signatures tend to offer very fast verification, though this comes at the cost of very large signatures.

Lattice-based signatures. "Lattice-based" schemes are based on hard problems in Euclidean lattices, and related problems like Learning With Errors (LWE). Contemporary examples of lattice-based signatures include the NIST Round 3 finalists Dilithium [11], based on the module-LWE problem, and Falcon [32], based on the NTRU problem. These schemes offer fast signing and verification, at the cost of relatively large signatures.

Multivariate signatures. The security of "multivariate" schemes is based on the difficulty of solving certain low-degree polynomial systems in many variables. Important contemporary examples of multivariate signatures include the NIST Round 3 finalist Rainbow [26], and alternate candidate GeMSS [22]. Multivariate signatures like Rainbow and GeMSS are interesting because they offer extremely small signature sizes, though this comes at the cost of extremely large public keys. However, recent analysis as in [19] has brought their security levels into question.

Isogeny-based signatures. Isogeny-based cryptosystems are based on the difficulty of computing unknown isogenies between elliptic curves. They inherit small key sizes from conventional elliptic-curve cryptography (ECC), which makes them interesting for microcontroller applications, but they also inherit (and increase) ECC's burden of heavy algebraic calculations. SIKE [39] is a NIST Round 3 alternate candidate for key establishment, but no isogeny-based signatures were submitted to the NIST PQC project—for the simple reason that no reasonable algorithms existed before the project deadline. Recent proposals such as SQISign [31] offer small parameters that are attractive for microcontroller applications, albeit at the cost of very slow runtimes. However, these signature schemes have not yet been subjected to extensive security analysis, and implementation work is still at an early stage.

Code-based signatures. Code-based cryptosystems are based on the difficulty of hard problems from the theory of error-correcting codes. The McEliece key exchange scheme [47] is among the oldest of all public-key cryptosystems. Code-based signatures, on the other hand, are much less well-established. While some recent proposals such as Wave [27, 28] have interesting potential, their security analysis and implementation work lags even further behind that of isogeny-based signatures.

Zero-knowledge-based signatures. There is also a category of post-quantum signatures using Zero-Knowledge (ZK) techniques, combining algorithms from symmetric cryptography with a technique known as Multi-Party Computation In The Head. The NIST Round 3 candidate Picnic [25] is an important example: It offers extremely small key sizes, but at the cost of very large signatures.

Summary. Table 3 compares signature and key sizes, and maturity of security analysis of various post-quantum signature scheme proposals, summarizing the "pros" and "cons" of each paradigm according to our requirements.

Table 3: Overview of post-quantum signature candidates from different paradigms. "Security analysis" reflects the maturity of analysis of the scheme: here we consider the age of the scheme, recent attacks, and how well-studied the underlying hard problem is. A tick or cross here is not an assertion that the scheme is proven secure, or known to be broken: it simply reflects our judgement on whether the scheme is closer to, or further from readiness for deployment.

Paradigm	Scheme	Security Analysis	Sizes (B) Signature	Public Key	Private Key
Hash-Based	LMS [48]	✓	4 756	60	64
	SPHINCS$^+$-128f [10]	✓	17 088	32	64
Lattice-Based	Dilithium [11]	✓	2 528	1 312	2 420
	Falcon [32]	✓	1 281	897	666
MQ-based	RainbowI [26]	✗	66	157 800	101 200
	GeMSS [22]	✗	417 416	14 520	48
Isogeny-based	SQISign [31]	✗	204	64	16
Code-based	WAVE [27]	✗	1625	≈ 13 000 000	N/R
Zero-knowledge-based	Picnic3-L1 [25]	✗	13 802	34	17

4.2 Selection of candidates

When choosing candidates for evaluation in our use case, we must consider not only their key and signature sizes and their runtime performance, but also their maturity with respect to security analysis. While the relatively compact parameters of some isogeny- and code-based signature schemes may make them interesting for future work targeting microcontrollers, at present these schemes are far from theoretical maturity, let alone real-world deployability. The true security level of the NIST multivariate and ZK-based candidates is a subject of current debate, though their extremely large keys and/or signatures would likely eliminate them from consideration for our applications in any case.

The NIST PQC project has dominated research in post-quantum cryptography in recent years. Its candidate cryptosystems are a natural first port of call for credible post-quantum signature algorithms, since they have had the benefit of concerted analysis from the cryptographic community—especially the Round 3 proposals, which are candidates for standardization in the coming years. However, these are not the only algorithms that we should consider. For example, among hash-based signature schemes, we might compare the older LMS scheme (which is not a NIST candidate) with the newer SPHINCS+ scheme (which is a NIST Round 3 alternate). LMS has smaller computational requirements, but the signer must maintain some state between signatures; SPHINCS+ is a heavier scheme, but it is stateless. Statelessness is an advantage for general applications. In our use case, however, statefulness is natural (it corresponds naturally to the version number on the software update), and easier to maintain—so the lighter LMS is a more natural choice.

Post-quantum choices. For the reasons above, we chose to focus our efforts on three post-quantum signature algorithms: LMS, Dilithium, and Falcon, representing the hash-based and lattice-based categories. LMS has 60-byte public keys and 4756-byte signatures. Dilithium III, targeting NIST security level 2, has 1312-byte public keys and 2420-byte signatures. Falcon-512, targeting NIST security level 1, has 897-byte public keys and 666-byte signatures.

Pre-quantum choices. To make a meaningful comparison with pre-quantum algorithms, we selected two elliptic-curve schemes: the state-of-the-art Ed25519 [17, 40] scheme, and the historic standard ECDSA based on the secp256 curve [52]. These schemes offer particularly small public keys and signatures—just 32 and 64 bytes, respectively—with an acceptable runtime and memory footprint for applications targeting microcontroller platforms.

5 BENCHMARKS

5.1 Hardware Testbed Setup

We carried out our measurements on popular, commercial, off-the-shelf IoT hardware. Our chosen platforms are representative of the landscape of modern 32-bit microcontroller architectures, including ARM Cortex-M, Espressif ESP32 and RISC-V:

- a Nordic nRF52840 Development Kit, which provides a typical microcontroller (ARM Cortex-M4) running at 64 MHz, with 256 kB RAM, 1 MB flash, and a 2.4 GHz radio transceiver compatible with both IEEE 802.15.4 and Bluetooth Low-Energy.
- a WROOM-32 board (ESP32 module with the ESP32-D0WDQ6 chip on board), which provides two low-power Xtensa® 32-bit LX6 microprocessors with integrated Wi-Fi and Bluetooth, operating at 80 MHz, with 520 kB RAM, 448 kB ROM and 16 kB RTC SRAM.
- a Sipeed Longan Nano GD32VF103CBT6 Development Board, which provides a RISC-V 32-bit core running at 72 MHz with 32 kB RAM and 128 kB flash.

IoT-Lab [8] provides some of the hardware for reproducibility on open access testbeds.

5.2 Software Setup

We used RIOT [7] as a base for our benchmarks.

Pre-quantum implementations. We used three different libraries, all currently supported in RIOT.

Ed25519: For Ed25519, we used two libraries: **C25519** (provided in [3]) and **Monocypher** [60]. The C25519 implementation contains constant-time finite-field arithmetic based on public-domain implementations of Bernstein's Curve25519 key exchange [16]. The Monocypher library also provides an implementation of Bernstein's Curve25519 and the Ed25519 signature scheme. One difference between the Monocypher and C25519 implementations is that Monocypher uses precomputed tables to speed up the computation of elliptic curve points. The precomputations are used in the "window method" for scalar multiplication. While this is known to speed up computations, it also has its drawbacks, as pointed out in [46].

ECDSA: For ECDSA, we used the Intel's **Tinycrypt** library [4], which is designed for embedded devices. The main goal of this library is to provide cryptographic standards for constrained devices. ECDSA differs from Ed25519 both in some specific details of the signature algorithm and in using the NIST standard p256 curve instead of Curve25519.

Post-quantum implementations. We re-used publicly available code after making some small modifications to fit the hardware requirements.

LMS: For LMS, we used the Cisco implementation [5], removing calls to `malloc` since it can lead to memory fragmentation [**?**], and in such low level can be dangerous and slow[1]. This change might lead to some small improvements in performance, since the kernel already knows the address at compile-time rather than only at runtime. For our benchmark, we used the smallest parameters proposed in [48, Section 5]: that is, SHA-2 with 256-bit output for the hash function (since we tried to keep the code as close as possible to [5]) with tree height 5, and 32 bytes associated with each node. For the LMOTS, we use 32 bytes and 4 bits of width for Winternitz coefficients. We remove the OpenSSL call from the original code and change for a implementation of SHA256 provided in their repository [5]. Furthermore, we are using HSS with 2 layers. These parameters satisfy the

[1] More details about dynamic allocation in embedded devices are available from https://github.com/RIOT-OS/RIOT/blob/master/CODING_CONVENTIONS.md

life cycle of updates: in particular, the key lifetime will never be surpassed by the amount of updates.

Dilithium: Our Dilithium implementation was based on the PQClean implementation [6]. As the Dilithium specification [11, Sec. 3.1] states, the first step in *signing* and *verifying* is to expand a random seed given in the public key into a large matrix. For our benchmarks, we prepared two versions of Dilithium:

- **Dynamic Dilithium** is the basic PQClean implementation.
- **Static Dilithium** is a modification of the PQClean implementation where the matrix is precomputed and stored in the flash memory.

Precomputing and storing the matrix makes signing and verification both faster, though it also (by definition) requires more flash and reduces flexibility, since signatures can only be verified against the flashed key.

Falcon: We used the the Falcon implementation provided by PQClean [6], without any significant structural modifications.

5.3 Measurements

Table 4: Size of private/public keys and signatures for benchmarked signature schemes.

	Algorithm	Private Key (B)	Public Key (B)	Signature (B)
Pre-quantum	Ed25519	32	32	64
	ECDSA p256	32	32	64
Post-quantum	Falcon	1281	897	666
	Dilithium	2528	1312	2420
	LMS (RFC8554)	64	60	4756

Table 5: Benchmark of pre/post-quantum signature schemes using an ARM Cortex-M microcontroller (nRF52840 Dev. Kit).

			Sign			Verify		
	Algorithm	Flash (B)	Time (ms)	Time (KiloTicks)	Stack (B)	Time (ms)	Time (KiloTicks)	Stack (B)
Pre-quantum	Ed25519 (C25519)	5106	845	54111	1180	1953	125012	1300
	Ed25519 (Monocypher)	13852	17	1136	1420	40	2599	1936
	ECDSA p256 (Tinycrypt)	6498	294	18871	1084	313	20037	1024
Post-quantum	Falcon	57613	1172	75020	42240	15	1004	4744
	Dilithium (Dynamic)	11664	465	29788	51762	53	3407	36058
	Dilithium (Static)	26672	135	8655	35240	23	1510	19504
	LMS (RFC8554)	12864	9224	590354	13212	123	7908	1580

Table 4 gives the sizes (in bytes) of the private key, public key, and signature for each of the schemes that we measured.

Tables 5, 6, and 7 present our benchmarking results on three different architectures Cortex-M, ESP32 and RISC-V.

For ease of comparison, all of the tables follow the same format. The upper and lower halves of each table describe the pre- and post-quantum algorithms, respectively. For each scheme and implementation tested, we give the total flash memory (ROM) used by the library; then, for the signing and verification operations we list the running time in milliseconds as well as in (thousands of) "ticks",

Table 6: Benchmark of pre/post-quantum signature schemes using an Espressif ESP32 microcontroller (WROOM-32 board).

			Sign			Verify		
	Algorithm	Flash (B)	Time (ms)	Time (KiloTicks)	Stack (B)	Time (ms)	Time (KiloTicks)	Stack (B)
Pre-quantum	Ed25519 (C25519)	5608	921	73690	1312	2165	173205	1440
	Ed25519 (Monocypher)	17238	21	1709	1536	60	4864	2160
	ECDSA p256 (Tinycrypt)	6869	333	26696	1296	374	29948	1216
Post-quantum	Falcon	60358	1172	93824	42504	16	1322	4920
	Dilithium (Dynamic)	12397	87	7036	51954	43	3508	36242
	Dilithium (Static)	27197	121	9694	35412	21	1706	19620
	LMS (RFC8554)	15177	7583	606674	13488	101	8141	1808

Table 7: Benchmark of pre/post-quantum signature schemes using a RISC-V microcontroller (Sipeed Longan Nano board).

			Sign			Verify		
	Algorithm	Flash (B)	Time (ms)	Time (KiloTicks)	Stack (B)	Time (ms)	Time (KiloTicks)	Stack (B)
Pre-quantum	Ed25519 (C25519)	6024	956	68883	1312	2242	161475	1440
	Ed25519 (Monocypher)	17328	16	1194	1376	41	3013	1920
	ECDSA p256 (Tinycrypt)	7452	270	19489	1224	308	22192	1112
Post-quantum	Falcon	11122[1]	—	—	—	13	975	4756
	Dilithium (Dynamic)	—	—	—	—	—	—	—
	Dilithium (Static)	25148[2]	—	—	—	17	1237	19572
	LMS (RFC8554)	15889	9105	655614	13352	122	8808	1736

[1] Flash only contains the verification algorithms.
[2] Flash contains the verification algorithms and hard-coded keys.

which we computed from the hardware clock and time spent. We also report the amount of stack memory required to successfully run the operation.

In Table 5 we see that Monocypher's Ed25519 is the fastest for signing among all the candidates running on the Nordic board; but Falcon is the fastest for signature verification, followed by Static Dilithium.

Table 6 shows that Monocypher's Ed25519 is also the fastest to sign on the WROOM-32 board, while Falcon and Static Dilithium offer the fastest signature verification.

Table 7 gives results on a RISC-V processor. Since the RISC-V board has only 32 kB RAM, the Falcon and Dilithium signing algorithms could not be run. For signature verification, we can see that the post-quantum schemes are substantially faster than the pre-quantum schemes.

6 THE IMPACT OF POST-QUANTUM IN SUIT/COSE

With our empirical results from §5, we can now address the three outstanding questions posed in the introduction of this paper (having already addressed hash functions in §3.2.3).

6.1 The cost of post-quantum security

How do post-quantum security costs compare to typical pre-quantum security costs? A toe-to-toe comparison between pre-quantum and post-quantum algorithms must consider public key and signature sizes, running time, and memory requirements.

Table 4 shows that post-quantum algorithms always have larger public key and signature sizes, generally by well over an order

of magnitude. Compared with standard elliptic-curve signature schemes, Falcon's public keys are 28× larger and its signatures are 10.4× larger; Dilithium's public keys are 41× larger than elliptic-curve keys, and its signatures are 38× larger. LMS avoids this spectacular growth in public key sizes, with keys only 1.875× larger than elliptic-curve public keys; but its signatures are a massive 74.3× larger than elliptic-curve signatures.

Looking at running time, as we saw in §5, post-quantum signatures have their advantages and disadvantages. Signature verification is considerably faster across all the IoT devices that we tested. Signing is generally slower. A comparison of the signing algorithms in Table 5 shows that the fastest post-quantum algorithm runs in 135 ms, which is 7.94× slower than Ed25519 (Monocypher). But the tables are turned when we compare signature verification algorithms: The fastest pre-quantum algorithm runs in 40 ms, which is 2.65× slower than post-quantum Falcon. Efficient verification is a required and valuable feature (in all scenarios), but in this setting, it comes at the price of an increase in stack and flash memory.

Looking at memory requirements, we see that post-quantum flash requirements can grow to over 11× the smallest pre-quantum flash. Similarly, post-quantum algorithms impose a considerable increase in stack memory.

Moving to post-quantum signatures therefore entails an increase in memory resources (stack and flash) and bandwidth (for keys and signatures). However, the verification algorithms are faster than standard pre-quantum algorithms, which means a reduction both in latency and in energy consumption. Ultimately, when choosing between these signature schemes in practice, one must consider the target IoT device, update frequency, and bandwidth usage.

6.2 The cost of post-quantum SUIT/COSE

What is the footprint of quantum-resistant security, relative to typical low-power operating system footprints? To add quantum resistance to SUIT/COSE following the workflow presented in Figure 1, we change the cryptographic algorithms from Ed25519 and SHA256 to Falcon, LMS, or Dilithium, and SHA3-256.

Impact on the SUIT Manifest. In practical terms, what changes in the size of the manifest in Phase 2, is an increase according to the signature size.

For example: Suppose we build a firmware update for RIOT, for the nRF52840dk board (based on a Cortex-M4 microcontroller). The protocol requires publishing the manifest containing the signed hash of the image: That is, we need to publish both the image and the SUIT metadata containing the signature. In §2 we had measured that the SUIT manifest with pre-quantum Ed25519 (or ECDSA) is of a total size of $419 + 64 = 483$ B. Moving to post-quantum signatures, this total becomes $419 + 666 = 1085$ B for Falcon, $419 + 2420 = 2839$ B for Dilithium, and $419 + 4756 = 5175$ B for LMS. Comparing these post-quantum sizes with the pre-quantum baseline, we see that post-quantum signatures are $\approx 2.24\times$ bigger with Falcon, $\approx 5.87\times$ bigger with Dilithium, and $\approx 9.84\times$ bigger with LMS.

Impact on the SUIT Software Update Performance. In order to grasp the whole picture, one must also consider the crucial aspect of network transfer costs. Based on our results, we evaluate the cost of the entire SUIT software update process in Table 8. We distinguish two main cases, as we did in §3.2.3. In the first, the updated software does *not* include the cryptographic libraries binary (i.e., these tools are external, e.g., in a bootloader): this corresponds to the column *Transfer* in Table 8. In the second, the updated software includes the cryptographic libraries binary; the column *Transfer w. crypto* corresponds to this case.

As we can observe in Table 8, the impact of switching to quantum-resistance security level on the SUIT update process varies widely in terms of network transfer costs, ranging from negligible increase (1%) to major impact (3× more), depending on the software update use case.

6.3 Post-quantum signatures for IoT

What are the potential alternatives for post-quantum digital signature schemes to secure IoT software updates? There are many possible deployments of IoT, and several possible scenarios for IoT software updates. The authorized maintainer is responsible for updating the firmware, and it is safe to assume that the maintainer has a PC, or equivalent powerful hardware. Hence, the computational burden of signing is not the main concern here. On the other hand, a constrained device will be responsible for signature verification in Phases 3, 4, and 5, as Figure 1 shows.

Real-life challenges faced on embedded devices. As we show in our study cases, the cryptography package does not run standalone in the board: it must coexist with several other modules (including kernel, network stack, and libraries), and the application itself.

One of the challenges that we faced in deploying the schemes was sharing stack memory (and SRAM memory). For example, on the RISC-V environment we used (recall Table 7) the total RAM memory budget available was 32kB for the whole system—which is very small, but not an uncommon budget. We could not run Dilithium to sign or verify within these limits, because it consumed all of the stack. In fact, we needed to adapt stack use for all of the post-quantum algorithms we used.

Execution speed is another challenge. In cases where signature verification takes a long time to perform, real-time applications may be affected if special care is not taken. Typically, on low-power IoT devices, there is no parallel computing happening. For instance, in the RIOT operating system, a preemptive multithreading paradigm is used, where a single thread is running at any given time. This means that if signature verification takes a long time, and if the thread it is running in has a high priority, the system is blocking on this task until it completes. It is therefore necessary to carefully tune the priority of the crypto verification thread in order to not stop other tasks which might be functionally essential, especially if signature verification execution speed is slow.

6.4 Usability of post-quantum signatures in real world

Let us revisit the four prototypical software update categories from §3.2.3, and consider the choice of postquantum signatures for each.

In Cases (1) and (2), the package contains the software update and the signature. Hence, speed and signature size are more important than flash size. In these cases, **Falcon** has an advantage over the alternatives we considered (LMS and Dilithium).

Table 8: Relative cost increase for SUIT with quantum resistance (on ARM Cortex M-4).

SUIT	Flash	Stack	Transfer	Transfer w. crypto
base w. Ed25519 / SHA256	52.4kB	16.3kB	47kB	53kB
with Falcon / SHA3-256	+120%	+18%	+1.1%	+120%
with LMS / SHA3-256	+34%	+1.2%	+9%	+43%
with Dilithium / SHA3-256	+30%	+210%	+4.3%	+34%

Case (3) is more complicated, with flash playing a much more crucial role. Since we must transfer the update with crypto over a low-power network, the package size has a higher impact on energy costs. As a point of reference, it takes 30s to 1mn to transfer 50kB on a low-power IEEE802.15.4 radio link, depending on the radio link quality and network load (assuming non-extreme cases). This is to compare with plus-or-minus 2s of computation speed difference for signature verification among the candidate cryptosystems. In this case, as shown in Table 8, **LMS** presents the best tradeoff between flash size, network transfer costs, verification time, and stack size.

In Case (4), the large network transfer costs overwhelm the other costs, reducing the comparative advantages of one post-quantum signature over another.

From a purely cryptographic point of view, given the maturity of hash function cryptanalysis, LMS remains the safest choice. As we briefly explained in §4.2, hash-based problems are quite mature, and have received extensive cryptanalysis from the cryptographic community. In comparison, the security of structured lattice-based schemes like Falcon is less well-understood.

Nevertheless, compared to pre-quantum state of the art, LMS imposes a significant increase in signature size and running time for signing and verifying, which has a major impact on SUIT performance; thus, despite its relative lack of maturity, the performance characteristics of Falcon make it extremely tempting for applications with smaller updates.

Deployment of post-quantum security. On a positive note: even though it necessitates increased data transfer, flash use and stack use, post-quantum security can be deployed on today's IoT hardware (i.e. tomorrow's legacy hardware). In a nutshell: we can upgrade to quantum-resistant software update security on heterogeneous legacy IoT hardware without requiring vast changes in portable C code.

It is clear that we will need to pay a price in the transition of pre-quantum to post-quantum algorithms. However, operating systems (for low powered devices such as RIOT) can already offer the tools to verify quantum-resistant signatures.

7 CONCLUSION

We have made an experimental study of quantum-resistant cryptography applied to securing software updates on low-power IoT devices. Taking an open-source implementation of the IETF standard SUIT as concrete case study, we offer a direct comparison of pre-quantum cryptographic schemes (signatures and hashing) against post-quantum cryptographic schemes in the same environment and on the same hardware platforms. We evaluate the cost of upgrading the security level from pre-quantum 128-bit security to NIST Level 1 post-quantum security. To properly analyze the impact of quantum-resistant cryptography schemes in IoT updates, we surveyed and selected candidate schemes, and we compared their performance using three low-power IoT platforms (ARM Cortex-M, RISC-V, and ESP32) representative of the current landscape of 32-bit microcontrollers. We show that quantum resistance is indeed achievable today on these platforms, and we derive recommendations based on our performance analysis. We also characterize how post-quantum digital signatures take a significant toll on the memory footprint and/or on network transfer costs in the IoT software update process, compared to pre-quantum schemes.

Future work – The priority remains to stabilize the current versions of post-quantum signatures, and then to push their implementations to common low-power embedded software platforms such as RIOT. Meanwhile, NIST is still determining which candidate schemes might form a new post-quantum signature standard; should new candidates be included in a new call, a new analysis will become necessary.

ACKNOWLEDGMENTS

H2020 SPARTA and the RIOT-fp project partly funded this work. We would like to Russ Housley and Christine van Vredendaal for comments in our draft.

REFERENCES

[1] 2002. FIPS PUB 180-2, Secure Hash Standard (SHS). U.S.Department of Commerce/National Institute of Standards and Technology.
[2] 2016. *WIRED Magazine: The Botnet That Broke the Internet Isn't Going Away.* https://www.wired.com/2016/12/botnet-broke-internet-isnt-going-away/.
[3] 2017. *Curve25519 and Ed25519 for low-memory systems.* https://www.dlbeer.co.nz/oss/c25519.html.
[4] 2018. *TinyCrypt Cryptographic Library.* https://github.com/01org/tinycrypt.
[5] 2021. *LMS Hash-Based Signature Implementation.* https://github.com/cisco/hash-sigs/.
[6] 2021. *PQClean.* https://github.com/PQClean/PQClean.
[7] 2021. *RIOT Operating System.* http://www.riot-os.org.
[8] Cedric Adjih et al. 2015. FIT IoT-LAB: A Large Scale Open Experimental IoT Testbed. In *IEEE WF-IoT.*
[9] Manos Antonakakis, Tim April, Michael Bailey, Matt Bernhard, Elie Bursztein, Jaime Cochran, Zakir Durumeric, J Alex Halderman, Luca Invernizzi, Michalis Kallitsis, et al. 2017. Understanding the Mirai Botnet. In *USENIX.*
[10] Jean-Philippe Aumasson, Daniel J. Bernstein, Ward Beullens, Christoph Dobraunig, Maria Eichlseder, Scott Fluhrer, Stefan-Lukas Gazdag, Andreas Hülsing, Panos Kampanakis, Stefan Kölbl, Tanja Lange, Martin M. Lauridsen, Florian Mentel, Ruben Niederhagen, Christian Rechberger, Joost Rijneveld, Peter Schwabe, and Bas Westerbaan. [n.d.]. *SPHINCS+ Stateless hash-based signatures.* https://sphincs.org/
[11] Roberto Avanzi, Joppe Bos, Léo Ducas, Eike Kiltz, Tancrède Lepoint, Vadim Lyubashevsky, John M. Schanck, Peter Schwabe, Gregor Seiler, and Damien Stehle. [n.d.]. *CRYSTALS/Dilithium.* https://pq-crystals.org/.
[12] Emmanuel Baccelli et al. 2018. RIOT: an Open Source Operating System for Low-end Embedded Devices in the IoT. *IEEE Internet of Things Journal* (2018).
[13] Josep Balasch et al. 2012. Compact Implementation and Performance Evaluationof Hash Functions in ATtiny Devices. In *International Conference on Smart Card Research and Advanced Applications.* Springer, 158–172.

[14] Gustavo Banegas and Daniel J. Bernstein. 2017. Low-Communication Parallel Quantum Multi-Target Preimage Search. In *Selected Areas in Cryptography - SAC 2017 - 24th International Conference, Ottawa, ON, Canada, August 16-18, 2017, Revised Selected Papers (Lecture Notes in Computer Science, Vol. 10719)*, Carlisle Adams and Jan Camenisch (Eds.). Springer, 325–335. https://doi.org/10.1007/978-3-319-72565-9_16

[15] Gustavo Banegas, Daniel J. Bernstein, Iggy van Hoof, and Tanja Lange. 2021. Concrete quantum cryptanalysis of binary elliptic curves. *IACR Trans. Cryptogr. Hardw. Embed. Syst.* 2021, 1 (2021), 451–472. https://doi.org/10.46586/tches.v2021.i1.451-472

[16] Daniel J Bernstein. 2006. Curve25519: new Diffie-Hellman speed records. In *International Workshop on Public Key Cryptography*. Springer, 207–228.

[17] Daniel J. Bernstein, Niels Duif, Tanja Lange, Peter Schwabe, and Bo-Yin Yang. 2012. High-speed high-security signatures. *Journal of Cryptographic Engineering* 2 (2012), 77–89.

[18] Guido Bertoni, Joan Daemen, Michaël Peeters, and Gilles Van Assche. 2013. Keccak. In *Advances in Cryptology - EUROCRYPT 2013, 32nd Annual International Conference on the Theory and Applications of Cryptographic Techniques, Athens, Greece, May 26-30, 2013. Proceedings (Lecture Notes in Computer Science, Vol. 7881)*, Thomas Johansson and Phong Q. Nguyen (Eds.). Springer, 313–314. https://doi.org/10.1007/978-3-642-38348-9_19

[19] Ward Beullens. 2020. Improved Cryptanalysis of UOV and Rainbow. *IACR Cryptol. ePrint Arch.* 2020 (2020), 1343. https://eprint.iacr.org/2020/1343

[20] Carsten Bormann et al. 2014. RFC 7228: Terminology for constrained node networks. IETF Request For Comments.

[21] Fabio Campos, Tim Kohlstadt, Steffen Reith, and Marc Stöttinger. 2020. LMS vs XMSS: Comparison of Stateful Hash-Based Signature Schemes on ARM Cortex-M4. In *Progress in Cryptology - AFRICACRYPT 2020 - 12th International Conference on Cryptology in Africa, Cairo, Egypt, July 20-22, 2020, Proceedings (Lecture Notes in Computer Science, Vol. 12174)*, Abderrahmane Nitaj and Amr M. Youssef (Eds.). Springer, 258–277. https://doi.org/10.1007/978-3-030-51938-4_13

[22] Antoine Casanova, Jean-Charles Faugère, Gilles Macario-Rat, Jacques Patarin, Ludovic Perret, and Jocelyn Ryckeghem. [n.d.]. *GeMSS: A GrEat Multivariate Short Signature*. https://www-polsys.lip6.fr/Links/NIST/GeMSS.html.

[23] Sofía Celi and Thom Wiggers. 2020. *KEMTLS: Post-quantum TLS without signatures*. https://blog.cloudflare.com/kemtls-post-quantum-tls-without-signatures/.

[24] André Chailloux, María Naya-Plasencia, and André Schrottenloher. 2017. An Efficient Quantum Collision Search Algorithm and Implications on Symmetric Cryptography. In *Advances in Cryptology - ASIACRYPT 2017 - 23rd International Conference on the Theory and Applications of Cryptology and Information Security, Hong Kong, China, December 3-7, 2017, Proceedings, Part II (Lecture Notes in Computer Science, Vol. 10625)*, Tsuyoshi Takagi and Thomas Peyrin (Eds.). Springer, 211–240. https://doi.org/10.1007/978-3-319-70697-9_8

[25] Melissa Chase, David Derler, Steven Goldfeder, Daniel Kales, Jonathan Katz, Vladimir Kolesnikov, Claudio Orlandi, Sebastian Ramacher, Christian Rechberger, Daniel Slamanig, Xiao Wang, and Greg Zaverucha. [n.d.]. Picnic: A Family of Post-Quantum Secure Digital Signature Algorithms. https://eprint.iacr.org/2020/1240.

[26] Ming-Shing Chen, Jintai Ding, Matthias Kannwischer, Jacques Patarin, Albrecht Petzoldt, Dieter Schmidt, and Bo-Yin Yang. [n.d.]. *Rainbow Signature*. https://www.pqcrainbow.org/.

[27] Thomas Debris-Alazard, Nicolas Sendrier, and Jean-Pierre Tillich. 2019. Wave: A New Family of Trapdoor One-Way Preimage Sampleable Functions Based on Codes. In *Advances in Cryptology - ASIACRYPT 2019 - 25th International Conference on the Theory and Application of Cryptology and Information Security, Kobe, Japan, December 8-12, 2019, Proceedings, Part I (Lecture Notes in Computer Science, Vol. 11921)*, Steven D. Galbraith and Shiho Moriai (Eds.). Springer, 21–51. https://doi.org/10.1007/978-3-030-34578-5_2

[28] Thomas Debris-Alazard, Nicolas Sendrier, and Jean-Pierre Tillich. [n.d.]. *WAVE: A code-based digital signature scheme*. http://wave.inria.fr/en/.

[29] Christoph Dobraunig, Maria Eichlseder, and Florian Mendel. 2015. Analysis of SHA-512/224 and SHA-512/256. In *Advances in Cryptology - ASIACRYPT 2015 - 21st International Conference on the Theory and Application of Cryptology and Information Security, Auckland, New Zealand, November 29 - December 3, 2015, Proceedings, Part II (Lecture Notes in Computer Science, Vol. 9453)*, Tetsu Iwata and Jung Hee Cheon (Eds.). Springer, 612–630. https://doi.org/10.1007/978-3-662-48800-3_25

[30] Morris Dworkin. 2015. SHA-3 Standard: Permutation-Based Hash and Extendable-Output Functions. https://doi.org/10.6028/NIST.FIPS.202

[31] Luca De Feo, David Kohel, Antonin Leroux, Christophe Petit, and Benjamin Wesolowski. 2020. SQISign: Compact Post-quantum Signatures from Quaternions and Isogenies. In *Advances in Cryptology - ASIACRYPT 2020 - 26th International Conference on the Theory and Application of Cryptology and Information Security, Daejeon, South Korea, December 7-11, 2020, Proceedings, Part I (Lecture Notes in Computer Science, Vol. 12491)*, Shiho Moriai and Huaxiong Wang (Eds.). Springer, 64–93. https://doi.org/10.1007/978-3-030-64837-4_3

[32] Pierre-Alain Fouque, Jeffrey Hoffstein, Paul Kirchner, Vadim Lyubashevsky, Thomas Pornin, Thomas Prest, Thomas Ricosset, Gregor Seiler, William Whyte, and Zhenfei Zhang. [n.d.]. *Falcon: Fast-Fourier lattice-based compact signatures over NTRU*. https://falcon-sign.info.

[33] Ruben Gonzalez, Andreas Hülsing, Matthias J. Kannwischer, Juliane Krämer, Tanja Lange, Marc Stöttinger, Elisabeth Waitz, Thom Wiggers, and Bo-Yin Yang. 2021. Verifying Post-Quantum Signatures in 8 kB of RAM. To appear at PQCrypto 2021. https://eprint.iacr.org/2021/662.

[34] Xu Guo, Sinan Huang, Leyla Nazhandali, and Patrick Schaumont. 2010. Fair and comprehensive performance evaluation of 14 second round SHA-3 ASIC implementations. In *The Second SHA-3 Candidate Conference*. Citeseer.

[35] Adrian Herrmann. [n.d.]. The Challenge of Security in IoT – A Study of Cryptographic Sponge Functions and an Implementation for RIOT. https://github.com/emmanuelsearch/Keccak-Bachelor-Thesis/raw/main/thesis.pdf.

[36] Huston Collins. [n.d.]. *Why TinyML is a giant opportunity*. https://venturebeat.com/2020/01/11/why-tinyml-is-a-giant-opportunity/

[37] Thomas Häner, Samuel Jaques, Michael Naehrig, Martin Roetteler, and Mathias Soeken. 2020. Improved Quantum Circuits for Elliptic Curve Discrete Logarithms. In *Post-Quantum Cryptography - 11th International Conference, PQCrypto 2020, Paris, France, April 15-17, 2020, Proceedings (Lecture Notes in Computer Science, Vol. 12100)*, Jintai Ding and Jean-Pierre Tillich (Eds.). Springer, 425–444. https://doi.org/10.1007/978-3-030-44223-1_23

[38] Andreas Hülsing, Denis Butin, Stefan-Lukas Gazdag, Joost Rijneveld, and Aziz Mohaisen. 2018. RFC 8391: XMSS: eXtended Merkle Signature Scheme. IETF Request for Comments. https://tools.ietf.org/html/rfc8391

[39] David Jao, Reza Azarderakhsh, Matthew Campagna, Craig Costello, Luca De Feo, Basil Hess, Aaron Hutchinson, Amir Jalali, Koray Karabina, Brian Koziel, Brian LaMacchia, Patrick Longa, Michael Naehrig, Geovandro Pereira, Joost Renes, Vladimir Soukharev, and David Urbanik. [n.d.]. SIKE – Supersingular Isogeny Key Encapsulation. https://sike.org/.

[40] S. Josefsson and I. Liusvaara. 2017. Edwards-Curve Digital Signature Algorithm (EdDSA). RFC 8032. http://www.ietf.org/rfc/rfc8032.txt

[41] Bernhard Jungk and Jurgen Apfelbeck. 2011. Area-efficient FPGA implementations of the SHA-3 finalists. In *2011 International Conference on Reconfigurable Computing and FPGAs*. IEEE, 235–241.

[42] Matthias J. Kannwischer, Joost Rijneveld, Peter Schwabe, and Ko Stoffelen. [n.d.]. PQM4: Post-quantum crypto library for the ARM Cortex-M4. https://github.com/mupq/pqm4.

[43] Jens-Peter Kaps et al. 2011. Lightweight implementations of SHA-3 candidates on FPGAs. In *International Conference on Cryptology in India*. Springer, 270–289.

[44] YoungBeom Kim, Hojin Choi, and Seog Chung Seo. 2020. Efficient implementation of SHA-3 hash function on 8-bit AVR-based sensor nodes. In *International Conference on Information Security and Cryptology*. Springer, 140–154.

[45] Kris Kwiatkowski and Luke Valenta. 2019. *The TLS Post-Quantum Experiment*. https://blog.cloudflare.com/the-tls-post-quantum-experiment/.

[46] Patrick Longa and Catherine H. Gebotys. 2009. Novel Precomputation Schemes for Elliptic Curve Cryptosystems. In *Applied Cryptography and Network Security, 7th International Conference, ACNS 2009, Paris-Rocquencourt, France, June 2-5, 2009. Proceedings (Lecture Notes in Computer Science, Vol. 5536)*, Michel Abdalla, David Pointcheval, Pierre-Alain Fouque, and Damien Vergnaud (Eds.). 71–88. https://doi.org/10.1007/978-3-642-01957-9_5

[47] Robert J McEliece. 1978. A public-key cryptosystem based on algebraic. *DSN Progress Report* 42-44 (1978), 114–116.

[48] David McGrew, Michael Curcio, and Scott Fluhrer. 2019. RFC 8554: Leighton-Micali Hash-Based Signatures. IETF Request for Comments. https://tools.ietf.org/html/rfc8554

[49] Brendan Moran, Hannes Tschofenig, and Henk Birkholz. 2021. *A Manifest Information Model for Firmware Updates in IoT Devices*. Internet-Draft draft-ietf-suit-information-model-12. Internet Engineering Task Force. https://datatracker.ietf.org/doc/html/draft-ietf-suit-information-model-12 Work in Progress.

[50] Brendan Moran, Hannes Tschofenig, Henk Birkholz, and Koen Zandberg. 2021. *A CBOR-based Serialization Format for the Software Updates for Internet of Things (SUIT) Manifest*. Internet-Draft draft-ietf-suit-manifest-12. Internet Engineering Task Force. https://datatracker.ietf.org/doc/html/draft-ietf-suit-manifest-12 Work in Progress.

[51] Brendan Moran, Hannes Tschofenig, David Brown, and Milosch Meriac. 2021. A Firmware Update Architecture for Internet of Things. RFC 9019. https://doi.org/10.17487/RFC9019

[52] National Institute of Standards and Technology. [n.d.]. FIPS186-4: Digital Signature Standard (DSS). https://doi.org/10.6028/NIST.FIPS.186-4.

[53] Lily Hay Newman. 2018. Inside the Unnerving Supply Chain Attack That Corrupted CCleaner. *Wired* (2018).

[54] NIST. [n.d.]. *Post-Quantum Cryptography Project*. https://csrc.nist.gov/projects/post-quantum-cryptography.

[55] Jim Schaad. 2017. CBOR Object Signing and Encryption (COSE). RFC 8152. https://doi.org/10.17487/RFC8152

[56] Peter Schwabe, Douglas Stebila, and Thom Wiggers. 2020. Post-Quantum TLS Without Handshake Signatures. In *CCS '20: 2020 ACM SIGSAC Conference on Computer and Communications Security, Virtual Event, USA, November 9-13, 2020*, Jay Ligatti, Xinming Ou, Jonathan Katz, and Giovanni Vigna (Eds.). ACM, 1461–1480. https://doi.org/10.1145/3372297.3423350

[57] Peter W. Shor. 1994. Algorithms for Quantum Computation: Discrete Logarithms and Factoring. In *35th Annual Symposium on Foundations of Computer Science, Santa Fe, New Mexico, USA, 20-22 November 1994*. IEEE Computer Society, 124–134. https://doi.org/10.1109/SFCS.1994.365700

[58] Dimitrios Sikeridis, Panos Kampanakis, and Michael Devetsikiotis. 2020. Post-Quantum Authentication in TLS 1.3: A Performance Study. In *27th Annual Network and Distributed System Security Symposium, NDSS 2020, San Diego, California, USA, February 23-26, 2020*. The Internet Society. https://www.ndss-symposium.org/ndss-paper/post-quantum-authentication-in-tls-1-3-a-performance-study/

[59] Hannes Tschofenig and Emmanuel Baccelli. 2019. Cyberphysical Security for the Masses: A Survey of the Internet Protocol Puite for Internet of Things Security. *IEEE Security & Privacy* 17, 5 (2019), 47–57.

[60] Loup Vaillant et al. [n.d.]. *Monocypher*. https://monocypher.org/.

[61] Koen Zandberg and Kaspar Schleiser. 2020. SUIT Reference Implementation. RIOT. http://api.riot-os.org/group__sys__suit.html

[62] Koen Zandberg, Kaspar Schleiser, Francisco Acosta, Hannes Tschofenig, and Emmanuel Baccelli. 2019. Secure Firmware Updates for Constrained IoT Devices Using Open Standards: A Reality Check. *IEEE Access* 7 (2019), 71907–71920.