Quantum-Resistant Software Update Security on Low-Power Networked Embedded Devices

Gustavo Banegas, Koen Zandberg, Emmanuel Baccelli, Adrian Herrmann, Benjamin Smith

Quantum-Resistant Software Update Security on Low-power Networked Embedded Devices

Gustavo Banegas*[1], Koen Zandberg*[2], Emmanuel Baccelli[2,3], Adrian Herrmann[3], and Benjamin Smith[1]

[1] Inria and Laboratoire d'Informatique de l'École Polytechnique, Institut Polytechnique de Paris Palaiseau, France
gustavo@cryptme.in, smith@lix.polytechnique.fr
[2] Inria Saclay, France
{koen.zandberg, emmanuel.baccelli}@inria.fr
[3] Freie Universität Berlin
Berlin, Germany
adrian.herrmann@fu-berlin.de

Abstract. As the Internet of Things (IoT) rolls out today to devices whose lifetime may well exceed a decade, conservative threat models should consider adversaries with access to quantum computing power. The IETF-specified SUIT standard defines a security architecture for IoT software updates, standardizing metadata and cryptographic tools—digital signatures and hash functions—to guarantee the update legitimacy. SUIT performance has been evaluated in the pre-quantum context, but not yet in a post-quantum context. Taking the open-source implementation of SUIT available in RIOT as a case study, we survey post-quantum considerations, and quantum-resistant digital signatures in particular, focusing on low-power, microcontroller-based IoT devices with stringent memory, CPU, and energy consumption constraints. We benchmark a range of pre- and post-quantum signature schemes on a range of IoT hardware including ARM Cortex-M, RISC-V, and Espressif (ESP32), which form the bulk of modern 32-bit microcontroller architectures. Interpreting our benchmarks in the context of SUIT, we estimate the real-world impact of transition from pre- to post-quantum signatures.

Keywords: Post-quantum, Security, IoT, Microcontroller, Embedded Systems

1 Introduction

Decades of experience with the Internet and networked software has shown that *you can't secure what you can't update*. Meanwhile, recent technological and societal trends have fuelled the massive deployment of cyberphysical systems; these

* These authors contributed equally to this work.

This work was funded by the European Commission through H2020 SPARTA, https://www.sparta.eu/,

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 [7]. However, the cure can become a disease: software updates are themselves an attack vector. Legitimate software updates laced with malware can compromise the updated device [46]. Once IoT devices are deployed, up and running, it thus becomes crucial to understand how, and when, software embedded in IoT devices is updated; how software updates are secured; and what level of security is provided.

In this paper, we study the impact of the pre- to post-quantum transition on IoT software updates, assuming that we want to maintain 128-bit conventional security (matching current internet security standards) while reaching NIST Level 1 post-quantum security. We aim to answer the following questions:

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

1.1 Low-power IoT and post-quantum cryptography

Low-power IoT characteristics. One prominent and highly 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), interconnected via low-power radio or wired communication. An estimated 250 billion microcontrollers are in use today around the globe [31]. Compared to microprocessor-based devices, microcontrollers aim for a different trade-off: They offer much smaller capacity in computing, networking, memory [17], in order to achieve radically lower energy consumption and a tiny price tag (<$1 unit price). It is not uncommon to have a total memory budget of 64kB of RAM and 500kB of ROM (flash) for the whole embedded system software—including drivers, crypto libraries, OS kernel, network stack and application logic. Nonetheless, the functionalities and services provided by constrained microcontroller-based devices are as crucial as those of less constrained elements in the cyberphysical system.

Post-quantum cryptography. Post-quantum cryptosystems are designed to run on contemporary hardware, yet resist adversaries equipped with both classical and quantum computers. Many signature schemes claim post-quantum security, some old and some new, but until now none has seen wide deployment. Recent research in post-quantum cryptography has revolved around the National Institute of Standards and Technology (NIST) Post-Quantum Cryptography project [47],

which will select a limited number of candidate schemes for standardization. This process is currently in its third round; draft standards are expected by 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, the first priority is to guarantee the legitimacy of software updates received via the network on low-power IoT devices. The crucial cryptographic tool here is a digital signature. Open standards targeting IoT security (such as the IETF [52]) specify a variety of signature schemes to secure software updates on low-power devices, including one scheme (LMS [41]) that offers quantum resistance.

Implementation approaches. Cryptographic implementations are often developed to tackle specific problems, such as speed or size. Most implementations take advantage of special instructions or hardware, but this narrows their applicability to specific architectures, which does not fully reflect the reality of IoT. Usually, operating systems (OS) must 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.

Focusing on portability and wide deployment, our experimental work did not use any 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).

1.2 Contributions and outline

In this paper, we:

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

We begin with a survey of related work in §2. In §3, we set out our case study: we describe SUIT software updates, categorise typical software update types, detail pre-quantum cryptographic considerations and begin to identify the main issues for the transition to post-quantum cryptography. We focus on post-quantum signature schemes in §4, explaining our choice of candidate schemes for benchmarking. Our experimental results appear in §5; we interpret their impact in the context of SUIT software updates in §6, before concluding in §7.

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 [55]. Various NIST candidate post-quantum schemes are compared as component algorithms in TLS 1.3 in [51], 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 [38] 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 [20,49].

For pure post-quantum cryptographic implementation work targeting microcontrollers, [18] evaluates the performance of stateful LMS on Cortex-M4 microcontrollers, while `pqm4` [35] 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 [28].

Many post-quantum signature schemes use standard SHA3 hashing under the hood. SHA3 performance in hardware (FGPA) has been studied [36,34,29], but surprisingly few studies focus on SHA3 performance in software on low-power microcontrollers. Some prior work exists: [11] and [37] focus on 8-bit microcontrollers, while [30] compares the performance of Keccak variants on 32-bit ARM Cortex-M microcontrollers.

3 Case Study: Low-power Software Updates with SUIT

The IETF's Software Updates for Internet of Things (SUIT) specifications [43,44] 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 available in RIOT [54], a common operating system for low-power IoT devices [10] which we use as base for our case study.

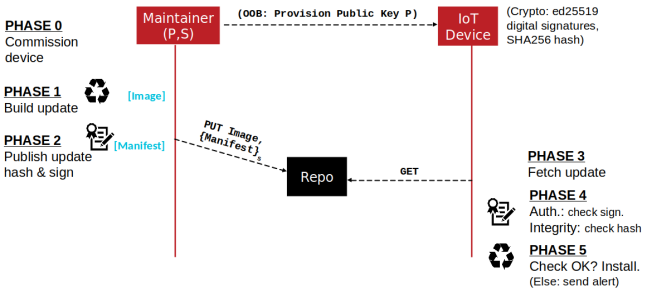

Fig. 1: SUIT secure software update workflow.

3.1 SUIT Workflow

Figure 1 shows the SUIT workflow. In the preliminary *Phase 0*, the authorized maintainer flashes the IoT device with commissioning material: the bootloader, initial image, and authorized crypto material. Once the IoT device is commissioned, up and running, we iterate a cycle 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 fetches the update and SUIT manifest from the repository (*Phase 3*), and verifies the signature (*Phase 4*). Upon successful verification, the new software is installed and booted (*Phase 5*); otherwise, the update is dropped.

The cryptographic tools needed for software updates in general, and SUIT in particular, are a digital signature scheme and a hash function. The digital signature authenticates (a hash of) the software update binary.

We distinguish four broad categories for low-power IoT software updates, defining the following four prototypical use cases:

- **U1**: Software module update ($\approx$ 5kB)
- **U2**: Small firmware update without crypto libraries ($\approx$ 50kB)
- **U3**: Small firmware update including crypto libraries ($\approx$ 50kB)
- **U4**: Large firmware update ($\approx$ 250kB)

We will see that the costs and recommendations for post-quantum SUIT are different for each of these typical updates.

3.2 Security features of SUIT

The metadata and the cryptographic primitives specified by SUIT can mitigate attacks exploiting software updates [42]. To give three simple examples:

- *Tampered/Unauthorized Firmware Update Attacks:* Adversaries may try to update the IoT device with a modified, 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 the integrity of both.

– *Firmware Update Replay Attacks:* Adversaries may replay a valid, but old (known-to-be-flawed) update. To mitigate this threat, SUIT metadata includes a sequence number that is increased with each new firmware update.
– *Firmware Update Mismatch Attacks:* Adversaries may send an authentic update to an incompatible device. To counter this, SUIT specifies the inclusion of device-specific conditions, to be verified before installing a firmware image.

3.3 Hash Functions with SUIT

The metadata of the update (the SUIT Manifest [43]) includes a cryptographic hash of the sofware update binary. The SUIT standard specification [43] allows the use of SHA-2 or SHA-3, with 224-, 256-, 384-, or 512-bit output.

Post-Quantum Considerations. There are few quantum attacks against SHA-2 and SHA-3 in the literature. Grover's algorithm may be parallelized to find hash preimages [12]; 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 [21]. On the other hand, classical attacks for SHA-2 might become a reality, as shown in [25].

Low-Power IoT Considerations. Low-power systems must run hash functions quickly, using as little power as possible; minimal memory (RAM and flash) usage is 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 compares the memory usage and speed of three hash function implementations on an ARM Cortex M4 microcontroller: RIOT's default implementation of SHA-256, a compact implementation of SHA3-256 optimized to minimize flash memory, and an implementation of SHA3-256 optimized for speed on Cortex-M4 ARMv7M architectures. Stack is roughly equivalent across the different implementations, but speed and flash vary widely: SHA3-256 can offer slightly faster execution than 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 of different Keccak variants on microcontrollers, see [30].

Table 1: SHA2 and SHA3 performance on an ARM Cortex-M4 microcontroller.

Hash function	Flash (B)	Stack (B)	Time (KTicks) to hash 64B	100B	1024B	10240B
SHA-256 (RIOT OS)	1008	384	277	278	1943	17933
SHA3-256 Compact	1692	404	1336	1342	10402	98448
SHA3-256 fast-ARMv7M	11548	284	223	228	1672	15732

Conclusions. Based on our analysis, there are no *direct* post-quantum aspects to consider here. Rather, the choice hash function should be driven by low-power criteria, and by other *indirect* post-quantum aspects detailed below. Recall the four prototypical use cases from §3.1. In **U1** and **U2**, 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 overhead for the hash function is of no concern, and SHA3-256 (optimized for speed) is the best choice. In **U3** and **U4**, however, the update includes the cryptographic tools and the hash function code; thus, a tradeoff appears. For small firmware updates as in **U3**, 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 as in **U4**, the storage and transfer overhead is negligible, so speed-optimized SHA3-256 is the best option again.

Let us now consider a complementary perspective: most post-quantum signature scheme proposals use SHA-3 in their constructions. Indeed, candidates for the upcoming NIST post-quantum signature standard are required to be SHA-3/SHAKE compatible, because that is the current US standard. Since space for code on IoT devices is very limited, factorization is typically desirable: using a single hash function for both hashing and signing reduces the flash footprint.

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

3.4 Digital Signatures with SUIT

The SUIT architecture relies on the software update distributor (the authorized maintainer in Figure 1) issuing a long-term public-private key pair used to generate and verify digital signatures on IoT software updates. The public key is pre-installed on the IoT device(s) to be updated during commissioning (*Phase 0*).

Digital signature use in SUIT is specified in the COSE standard [48], which defines how to sign and encrypt compact (CBOR) binary serialized objects. For the 128-bit classical security level, COSE specifies the elliptic-curve signature schemes Ed25519 and ECDSA on NIST P-256. These schemes offer very small public (and private) keys at 32B each, and 64B signatures.

To give some concrete perspective, Table 2 shows the memory footprint of SUIT and related software components using Ed25519, compared to the whole software embedded on the IoT device. This measurement uses the open-source RIOT implementation on the Nordic nRF52840 Development Kit, a popular low-power IoT board based on an ARM Cortex-M4 microcontroller. The flash memory footprint of this firmware is 52.5kB; the RAM (stack) usage is 16.3kB.

In this typical pre-quantum configuration, the crypto represents a small part of the flash footprint: under 15% of a ≈ 50 kB total. The elliptic-curve signature adds 15% to the size of the SUIT manifest metadata and less than 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: Network transfer cost and decomposition of SUIT firmware update (for nRF52840 Dev Kit) using minimal metadata with Ed25519+SHA-256.

	SUIT		OS firmware				
	Metadata	Signature	**Total**	Crypto	Kernel Modules	Network Modules	OTA
Size (B)	419	64	52485	7161	17039	20113	8172

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 [50,32,13].

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.

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 began standardizing alternative signature schemes with COSE/SUIT for post-quantum security, such as LMS [41]. In the next sections, we survey alternative quantum-resistant schemes, comparing their performance against state-of-the-art pre-quantum schemes 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 [47]. 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. The 128-bit security level is now standard in mainstream internet applications requiring long-term security.

4.1 Post-quantum signature paradigms

We can classify the post-quantum signatures into the 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. Hash-based signatures tend to offer very fast verification, though this comes at the cost of very large signatures.

Lattice-based signatures. These schemes are based on hard problems in Euclidean lattices, and related problems like Learning With Errors (LWE). These schemes offer fast signing and verification, but have 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. A recent analysis in [16] 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. Recent isogeny-based signature schemes such as SQISign [26] inherit small parameter sizes from conventional elliptic-curve cryptography (ECC), making them interesting for microcontroller applications, but they also inherit and increase ECC's burden of heavy algebraic calculations, which makes for very slow runtimes. These signature schemes have not yet been subjected to extensive security analysis.

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 [40] is among the oldest of all public-key cryptosystems. Code-based signatures, on the other hand, are much less well-established.

Zero-knowledge-based signatures. A new category of post-quantum signatures uses Zero-Knowledge (ZK) techniques, combining algorithms from symmetric cryptography with a technique known as Multi-Party Computation In The Head.

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.

4.2 Selection of candidates

When choosing candidate signature schemes, we must consider key and signature sizes, runtime performance, and 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. 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

Table 3: Overview of post-quantum signature candidates. "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.

Paradigm	Scheme	Security Analysis	Sizes (B) Signature	Public Key	Private Key
Hash-based	LMS [41]	mature	4 756	60	64
	SPHINCS$^+$-128f [8]	mature	17 088	32	64
Lattice-based	Dilithium [9]	less mature	2 528	1 312	2 420
	Falcon [27]	less mature	1 281	897	666
MQ-based	RainbowI [23]	not mature	66	157 800	101 200
	GeMSS [19]	not mature	417 416	14 520	48
Isogeny-based	SQISign [26]	not mature	204	64	16
Code-based	WAVE [24]	not mature	1625 $\approx$ 13 000 000		N/R
Zero-knowledge-based	Picnic3-L1 [22]	not mature	13 802	34	17

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 60B public keys and 4756-byte signatures. Dilithium II, targeting NIST security level 2, has 1312B public keys and 2420B signatures. Falcon-512, targeting NIST security level 1, has 897B public keys and 666B signatures.

Pre-quantum choices. To make a meaningful comparison with pre-quantum algorithms, we selected two elliptic-curve schemes: the Ed25519 [15,33] scheme, and the historic standard ECDSA based on the secp256 curve [45]. These schemes offer particularly small 32B public keys and 64B signatures.

5 Benchmarks

5.1 Hardware Testbed Setup

Our benchmarks were run on popular, commercial, off-the-shelf IoT hardware, representative of the landscape of modern 32-bit microcontroller architectures:

– **ARM Cortex-M4**: the **Nordic nRF52840 Development Kit** provides a typical ARM Cortex-M4 microcontroller 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.
- **Espressif ESP32**: the **WROOM-32 board** (ESP32 module with the ESP32-D0WDQ6 chip on board) 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.
- **RISC-V**: the **Sipeed Longan Nano GD32VF103CBT6** Development Board provides a RISC-V 32-bit core running at 72 MHz with 32 kB RAM and 128 kB flash.

IoT-Lab [6] provides this hardware for reproducibility on open access testbeds.

5.2 Software Setup

We used RIOT [5] 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 [1]) and **Monocypher** [53]. Both contain constant-time finite-field arithmetic based on public-domain implementations [14]. One difference between Monocypher and C25519 is that Monocypher uses precomputed tables to speed up the computation of elliptic curve points.

ECDSA: For ECDSA, we used Intel's **Tinycrypt** library [2], which is designed 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 [3], removing calls to `malloc` since it can lead to memory fragmentation [39], and in such low level can be dangerous and slow[4]. 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 [41, 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 [3]) 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 [3]. Furthermore, we are using HSS with 2 layers. These parameters satisfy the life cycle of updates: in particular, the key lifetime will never be surpassed by the amount of updates.

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

Dilithium: We prepared two Dilithium implementations based on PQClean [4].
- **Dynamic Dilithium** is the basic PQClean implementation. The first step in *signing* and *verifying* is to expand a random seed given in the public key into a large matrix (cf. [9, Sec. 3.1]).
- **Static Dilithium** modifies the PQClean implementation to precompute the matrix and store it in the flash memory. This makes signing and verification both faster, though it also requires more flash and reduces flexibility, since signatures can only be verified against the flashed key.

Falcon: We used the Falcon implementation provided by PQClean [4], without any significant structural modifications.

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

Table 4: Key and signature sizes 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

5.3 Pre- and post-quantum signature benchmarks

Tables 5, 6, and 7 present our benchmarking results on our three target architectures: Cortex-M, ESP32 and RISC-V. For each implementation we give the total flash memory used by the library, and for the signing and verification operations we list the running time in milliseconds and in thousands of "ticks" (computed from the hardware clock and time spent), and the stack required.

We see that Monocypher's Ed25519 is the fastest for signing among all the candidates, on all three boards. (Since the RISC-V board has only 32 kB RAM, the Falcon and Dilithium signing algorithms could not be run there.) Falcon offers the fastest verification on all three boards, followed by Static Dilithium.

6 The impact of post-quantum in SUIT/COSE

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- and post-quantum signatures 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

Table 5: Signature benchmarks: ARM Cortex-M (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 6: Signature benchmarks: Espressif ESP32 (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

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.

Table 7: Signature benchmarks: RISC-V (Sipeed Longan Nano board). *Falcon flash only contains the verification algorithm. Static Dilithium flash contains the verification algorithm and hard-coded public key.*

			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	—	—	—	13	975	4756
	Dilithium (Dynamic)	—	—	—	—	—	—	—
	Dilithium (Static)	25148	—	—	—	17	1237	19572
	LMS (RFC8554)	15889	9105	655614	13352	122	8808	1736

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? As a concrete example: consider a firmware update for RIOT on the nRF52840dk. In the classification of §3.1, the update is

- type **U2**, where the update does *not* include the cryptographic libraries binary (i.e., these tools are external, e.g., in a bootloader), or
- type **U3**, where the update includes the cryptographic libraries binary.

We want to add quantum resistance to SUIT/COSE by changing the cryptographic algorithms from Ed25519 and SHA256 to Falcon, LMS, or Dilithium, and SHA3-256.

Impact on the SUIT manifest. In practical terms, the size of the SUIT manifest increases according to the new signature size. In §2 we saw that the SUIT manifest with pre-quantum Ed25519 (or ECDSA) has total size $419+64 = 483$B. Moving to post-quantum signatures, this total becomes

- Falcon: $419 + 666 = 1085$B, a $\approx 2.24\times$ increase;
- Dilithium: $419 + 2420 = 2839$B, a $\approx 5.87\times$ increase; and
- LMS: $419 + 4756 = 5175$B, a $\approx 9.84\times$ increase.

Impact on SUIT software update performance. Now consider the crucial aspect of network transfer costs, and the memory resources required to actually apply the firmware update on the IoT device. Table 8 uses our measurements to evaluate the relative cost of the entire SUIT software update process. We see that impact of switching to quantum-resistant security in SUIT varies widely in terms of network transfer costs, ranging from negligible increase ($\sim 1\%$) to major impact ($3\times$ 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,

Table 8: Relative costs for SUIT with quantum resistance (ARM Cortex M4).

			Data Transfer	
SUIT	Flash	Stack	**U2**	**U3**
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%

and several possible scenarios for IoT software updates. It is safe to assume that the authorized maintainer, responsible for updating the firmware, has 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 of the SUIT workflow in Figure 1.

As we have seen above, 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 challenge that we faced in deploying the schemes was sharing stack memory (and SRAM memory). For example, on our RISC-V platform (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. Slow signature verification may impact real-time applications if special care is not taken. Typically, on low-power IoT devices, there is no parallel computing. For instance, RIOT OS uses a preemptive multithreading paradigm, where a single thread is running at any given time. If signature verification takes a long time, running in a high-priority thread, then the system blocks on this task until completion. It is therefore necessary to carefully tune the priority of the crypto verification thread so as not to stop other functionally essential tasks, especially if signature verification is slow.

6.4 Real-world usability of post-quantum signatures

Let us revisit the four prototypical software update categories from §3.1, and consider the choice of postquantum signatures for each.

In use cases **U1** (a small module update) and **U2** (small firmware update without crypto libraries), 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 LMS and Dilithium.

The use case **U3** (small firmware update with crypto libraries) 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 30-60s to transfer 50kB

on a low-power IEEE802.15.4 radio link, depending on 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 use case **U4** (larger updates), the large network transfer costs overwhelm the other costs, reducing the comparative advantages of one post-quantum signature over another.

From the point of view of cryptographic maturity, LMS is the safest choice. As noted in §4.2, hash-based problems have received extensive cryptanalysis from the cryptographic community, while the security of structured lattice-based schemes like Falcon is less well-understood. Nevertheless, compared to the pre-quantum state of the art, LMS imposes a significant increase in signature size and running time, 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, and stack, 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 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 the transition from pre- to post-quantum 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 compare the performance of standard pre-quantum and selected post-quantum candidates for the required cryptographic schemes (signatures and hashing), in the same environment (RIOT) on three low-power IoT platforms (ARM Cortex-M, RISC-V, and ESP32) representative of the current landscape of 32-bit microcontrollers. We show that upgrading from classical 128-bit security to NIST Level 1 post-quantum security is indeed achievable today on these platforms, and we derive recommendations based on our performance analysis. We also characterize the toll of the pre- to post-quantum transition on memory footprints and network transfer costs in the IoT software update process.

Future work. The priority remains to stabilize the current versions of post-quantum signatures before pushing their implementations to common low-power embedded software platforms such as RIOT. Meanwhile, NIST has yet to determine the new post-quantum signature standard; should new candidates be included in a new call, more analysis will be necessary.

References

1. Curve25519 and Ed25519 for low-memory systems, October 2017. `https://www.dlbeer.co.nz/oss/c25519.html`.
2. TinyCrypt Cryptographic Library, July 2018. `https://github.com/01org/tinycrypt`.
3. LMS Hash-Based Signature Implementation, 2021. `https://github.com/cisco/hash-sigs/`.
4. PQClean, 2021. `https://github.com/PQClean/PQClean`.
5. RIOT Operating System, 2021. `http://www.riot-os.org`.
6. Cedric Adjih, Emmanuel Baccelli, Eric Fleury, Gaetan Harter, Nathalie Mitton, Thomas Noel, Roger Pissard-Gibollet, Frederic Saint-Marcel, Guillaume Schreiner, Julien Vandaele, and Thomas Watteyne. Fit iot-lab: A large scale open experimental iot testbed. In *2015 IEEE 2nd World Forum on Internet of Things (WF-IoT)*, pages 459–464, 2015.
7. Manos Antonakakis, Tim April, Michael Bailey, Matt Bernhard, Elie Bursztein, Jaime Cochran, Zakir Durumeric, J. Alex Halderman, Luca Invernizzi, Michalis Kallitsis, Deepak Kumar, Chaz Lever, Zane Ma, Joshua Mason, Damian Menscher, Chad Seaman, Nick Sullivan, Kurt Thomas, and Yi Zhou. Understanding the mirai botnet. In *26th USENIX Security Symposium (USENIX Security 17)*, pages 1093–1110, Vancouver, BC, August 2017. USENIX Association.
8. 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. SPHINCS+ stateless hash-based signatures. URL: `https://sphincs.org/`.
9. Roberto Avanzi, Joppe Bos, Léo Ducas, Eike Kiltz, Tancrède Lepoint, Vadim Lyubashevsky, John M. Schanck, Peter Schwabe, Gregor Seiler, and Damien Stehle. CRYSTALS/Dilithium. URL: `https://pq-crystals.org/`.
10. Emmanuel Baccelli, Cenk Gündoğan, Oliver Hahm, Peter Kietzmann, Martine S. Lenders, Hauke Petersen, Kaspar Schleiser, Thomas C. Schmidt, and Matthias Wählisch. Riot: An open source operating system for low-end embedded devices in the iot. *IEEE Internet of Things Journal*, 5(6):4428–4440, 2018.
11. Josep Balasch, Barış Ege, Thomas Eisenbarth, Benoit Gérard, Zheng Gong, Tim Güneysu, Stefan Heyse, Stéphanie Kerckhof, François Koeune, Thomas Plos, Thomas Pöppelmann, Francesco Regazzoni, François-Xavier Standaert, Gilles Van Assche, Ronny Van Keer, Loïc van Oldeneel tot Oldenzeel, and Ingo von Maurich. Compact implementation and performance evaluation of hash functions in ATtiny devices. In Stefan Mangard, editor, *Smart Card Research and Advanced Applications*, pages 158–172, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg.
12. Gustavo Banegas and Daniel J. Bernstein. Low-communication parallel quantum multi-target preimage search. In Carlisle Adams and Jan Camenisch, editors, *Selected Areas in Cryptography - SAC 2017 - 24th International Conference, Ottawa, ON, Canada, August 16-18, 2017, Revised Selected Papers*, volume 10719 of *Lecture Notes in Computer Science*, pages 325–335. Springer, 2017.
13. Gustavo Banegas, Daniel J. Bernstein, Iggy van Hoof, and Tanja Lange. Concrete quantum cryptanalysis of binary elliptic curves. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2021(1):451–472, 2021.

14. Daniel J Bernstein. Curve25519: new Diffie–Hellman speed records. In *International Workshop on Public Key Cryptography*, pages 207–228. Springer, 2006.
15. Daniel J. Bernstein, Niels Duif, Tanja Lange, Peter Schwabe, and Bo-Yin Yang. High-speed high-security signatures. *Journal of Cryptographic Engineering*, 2:77–89, 2012.
16. Ward Beullens. Improved cryptanalysis of UOV and rainbow. *IACR Cryptol. ePrint Arch.*, 2020:1343, 2020.
17. Carsten Bormann, Ari Keranen, and Mehmet Ersue. RFC 7228: Terminology for constrained node networks. IETF Request For Comments, 2014.
18. Fabio Campos, Tim Kohlstadt, Steffen Reith, and Marc Stöttinger. LMS vs XMSS: comparison of stateful hash-based signature schemes on ARM cortex-M4. In Abderrahmane Nitaj and Amr M. Youssef, editors, *Progress in Cryptology - AFRICACRYPT 2020 - 12th International Conference on Cryptology in Africa, Cairo, Egypt, July 20-22, 2020, Proceedings*, volume 12174 of *Lecture Notes in Computer Science*, pages 258–277. Springer, 2020.
19. Antoine Casanova, Jean-Charles Faugère, Gilles Macario-Rat, Jacques Patarin, Ludovic Perret, and Jocelyn Ryckeghem. GeMSS: A GrEat Multivariate Short Signature. URL: `https://www-polsys.lip6.fr/Links/NIST/GeMSS.html`.
20. Sofía Celi and Thom Wiggers. KEMTLS: Post-quantum TLS without signatures, January 2020. `https://blog.cloudflare.com/kemtls-post-quantum-tls-without-signatures/`.
21. André Chailloux, María Naya-Plasencia, and André Schrottenloher. An efficient quantum collision search algorithm and implications on symmetric cryptography. In Tsuyoshi Takagi and Thomas Peyrin, editors, *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*, volume 10625 of *Lecture Notes in Computer Science*, pages 211–240. Springer, 2017.
22. 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. Picnic: A family of post-quantum secure digital signature algorithms. `https://eprint.iacr.org/2017/279`.
23. Ming-Shing Chen, Jintai Ding, Matthias Kannwischer, Jacques Patarin, Albrecht Petzoldt, Dieter Schmidt, and Bo-Yin Yang. Rainbow signature. URL: `https://www.pqcrainbow.org/`.
24. Thomas Debris-Alazard, Nicolas Sendrier, and Jean-Pierre Tillich. Wave: A new family of trapdoor one-way preimage sampleable functions based on codes. In Steven D. Galbraith and Shiho Moriai, editors, *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*, volume 11921 of *Lecture Notes in Computer Science*, pages 21–51. Springer, 2019.
25. Christoph Dobraunig, Maria Eichlseder, and Florian Mendel. Analysis of SHA-512/224 and SHA-512/256. In Tetsu Iwata and Jung Hee Cheon, editors, *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*, volume 9453 of *Lecture Notes in Computer Science*, pages 612–630. Springer, 2015.

26. Luca De Feo, David Kohel, Antonin Leroux, Christophe Petit, and Benjamin Wesolowski. SQISign: Compact Post-quantum Signatures from Quaternions and Isogenies. In Shiho Moriai and Huaxiong Wang, editors, *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*, volume 12491 of *Lecture Notes in Computer Science*, pages 64–93. Springer, 2020.
27. Pierre-Alain Fouque, Jeffrey Hoffstein, Paul Kirchner, Vadim Lyubashevsky, Thomas Pornin, Thomas Prest, Thomas Ricosset, Gregor Seiler, William Whyte, and Zhenfei Zhang. Falcon: Fast-Fourier lattice-based compact signatures over NTRU. URL: `https://falcon-sign.info`.
28. Ruben Gonzalez, Andreas Hülsing, Matthias J. Kannwischer, Juliane Krämer, Tanja Lange, Marc Stöttinger, Elisabeth Waitz, Thom Wiggers, and Bo-Yin Yang. Verifying post-quantum signatures in 8 kb of RAM. In Jung Hee Cheon and Jean-Pierre Tillich, editors, *Post-Quantum Cryptography - 12th International Workshop, PQCrypto 2021, Daejeon, South Korea, July 20-22, 2021, Proceedings*, volume 12841 of *Lecture Notes in Computer Science*, pages 215–233. Springer, 2021.
29. Xu Guo, Sinan Huang, Leyla Nazhandali, and Patrick Schaumont. Fair and comprehensive performance evaluation of 14 second round SHA-3 ASIC implementations. In *The Second SHA-3 Candidate Conference*. Citeseer, 2010.
30. Adrian Herrmann. The Challenge of Security in IoT – A Study of Cryptographic Sponge Functions and an Implementation for RIOT, 3. `https://github.com/emmanuelsearch/Keccak-Bachelor-Thesis/raw/main/thesis.pdf`.
31. Huston Collins. Why TinyML is a giant opportunity. URL: `https://venturebeat.com/2020/01/11/why-tinyml-is-a-giant-opportunity/`.
32. Thomas Häner, Samuel Jaques, Michael Naehrig, Martin Roetteler, and Mathias Soeken. Improved Quantum Circuits for Elliptic Curve Discrete Logarithms. In Jintai Ding and Jean-Pierre Tillich, editors, *Post-Quantum Cryptography - 11th International Conference, PQCrypto 2020, Paris, France, April 15-17, 2020, Proceedings*, volume 12100 of *Lecture Notes in Computer Science*, pages 425–444. Springer, 2020.
33. S. Josefsson and I. Liusvaara. Edwards-Curve Digital Signature Algorithm (EdDSA). RFC 8032, 2017.
34. Bernhard Jungk and Jurgen Apfelbeck. Area-efficient FPGA implementations of the SHA-3 finalists. In *2011 International Conference on Reconfigurable Computing and FPGAs*, pages 235–241. IEEE, 2011.
35. Matthias J. Kannwischer, Joost Rijneveld, Peter Schwabe, and Ko Stoffelen. PQM4: Post-quantum crypto library for the ARM Cortex-M4. `https://github.com/mupq/pqm4`.
36. Jens-Peter Kaps, Panasayya Yalla, Kishore Kumar Surapathi, Bilal Habib, Susheel Vadlamudi, Smriti Gurung, and John Pham. Lightweight implementations of SHA-3 candidates on FPGAs. In Daniel J. Bernstein and Sanjit Chatterjee, editors, *Progress in Cryptology – INDOCRYPT 2011*, pages 270–289, Berlin, Heidelberg, 2011. Springer Berlin Heidelberg.
37. YoungBeom Kim, Hojin Choi, and Seog Chung Seo. Efficient implementation of SHA-3 hash function on 8-bit AVR-based sensor nodes. In *International Conference on Information Security and Cryptology*, pages 140–154. Springer, 2020.
38. Kris Kwiatkowski and Luke Valenta. The TLS Post-Quantum Experiment, October 2019. `https://blog.cloudflare.com/the-tls-post-quantum-experiment/`.

39. Jean J. Labrosse. Chapter 15 - real-time kernels. In Jack Ganssle, editor, *The Firmware Handbook*, Embedded Technology, pages 211–229. Newnes, Burlington, 2004.
40. Robert J McEliece. A public-key cryptosystem based on algebraic coding theory. *DSN Progress Report*, 42-44:114–116, 1978.
41. David McGrew, Michael Curcio, and Scott Fluhrer. RFC 8554: Leighton-Micali hash-based signatures. IETF Request for Comments, April 2019.
42. Brendan Moran, Hannes Tschofenig, and Henk Birkholz. A Manifest Information Model for Firmware Updates in IoT Devices. Internet-Draft draft-ietf-suit-information-model-12, Internet Engineering Task Force, May 2021. Work in Progress.
43. Brendan Moran, Hannes Tschofenig, Henk Birkholz, and Koen Zandberg. 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, February 2021. Work in Progress.
44. Brendan Moran, Hannes Tschofenig, David Brown, and Milosch Meriac. A Firmware Update Architecture for Internet of Things. RFC 9019, April 2021.
45. National Institute of Standards and Technology. FIPS186-4: Digital signature standard (DSS). `https://doi.org/10.6028/NIST.FIPS.186-4`.
46. Lily Hay Newman. Inside the Unnerving Supply Chain Attack That Corrupted CCleaner. *Wired*, 2018.
47. NIST. Post-Quantum Cryptography Project. URL: `https://csrc.nist.gov/projects/post-quantum-cryptography`.
48. Jim Schaad. CBOR Object Signing and Encryption (COSE). RFC 8152, July 2017.
49. Peter Schwabe, Douglas Stebila, and Thom Wiggers. Post-quantum TLS without handshake signatures. In Jay Ligatti, Xinming Ou, Jonathan Katz, and Giovanni Vigna, editors, *CCS '20: 2020 ACM SIGSAC Conference on Computer and Communications Security, Virtual Event, USA, November 9-13, 2020*, pages 1461–1480. ACM, 2020.
50. Peter W. Shor. 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*, pages 124–134. IEEE Computer Society, 1994.
51. Dimitrios Sikeridis, Panos Kampanakis, and Michael Devetsikiotis. 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, 2020.
52. Hannes Tschofenig and Emmanuel Baccelli. Cyberphysical Security for the Masses: A Survey of the Internet Protocol Puite for Internet of Things Security. *IEEE Security & Privacy*, 17(5):47–57, 2019.
53. Loup Vaillant. Monocypher. `https://monocypher.org/`.
54. Koen Zandberg and Kaspar Schleiser. SUIT Reference Implementation. RIOT, 2020. `http://api.riot-os.org/group__sys__suit.html`.
55. Koen Zandberg, Kaspar Schleiser, Francisco Acosta, Hannes Tschofenig, and Emmanuel Baccelli. Secure Firmware Updates for Constrained IoT Devices Using Open Standards: A Reality Check. *IEEE Access*, 7:71907–71920, 2019.