Post-quantum signatures in practice: *Securing IoT software updates*

Benjamin **Smith**

SQIParty // Universitat de Lleida // 29/04/2025

Équipe-Projet **GRACE** // **Inria** Saclay

Constrained environments: Low-end IoT

Limited power

- Often battery-powered: need to minimise power consumption
- CPU is not the only thing consuming power: memory and network, too.

Limited memory and **storage**

- Very little RAM, especially once you include the enveloping application
- Small ROM/Flash: need to minimize code size and complexity

Operational constraints

- Often communicating over low-power radio
- Side-channel attack surface is often extremely large

$\implies$ *Hybrid pre-/post-quantum crypto highly relevant*

Case study:
Post-quantum software updates for low-end IoT devices

Target platform: RIOT OS

RIOT is a free, community-drive open-source OS for **low-end** IoT devices.

- Supports $\geq$ 73 **CPUs** (8-, 16-, and 32-bit)
- Supports $\geq$ 276 **different boards**
- Application development: **C, C++, Rust**
- Modular microkernel design
- Find out more: **`https://riot-os.org`**

Target platform: RIOT OS

RIOT is a free, community-drive open-source OS for **low-end** IoT devices.

- Supports $\geq$ 73 **CPUs** (8-, 16-, and 32-bit)
- Supports $\geq$ 276 **different boards**
- Application development: **C, C++, Rust**
- Modular microkernel design
- Find out more: **`https://riot-os.org`**

Question: what is the practical cost of switching RIOT crypto from pre-quantum to post-quantum cryptography?

Post-quantum software updates for IoT

You can’t secure what you can’t update

Post-quantum software updates for IoT

You can’t secure what you can’t update, *securely*.

Post-quantum software updates for IoT

You can't secure what you can't update, *securely*.

Problem: updating **low-end IoT devices** (low power, low memory, low price).

RIOT supports **SUIT** (RFC 9019): **S**ecure **U**pdates for the **I**nternet of **T**hings.
Critical cryptographic component: ***elliptic-curve digital signatures**.*

Question: what is the real cost of adding post-quantum security to SUIT?

Post-quantum software updates for IoT

You can't secure what you can't update, *securely*.

Problem: updating **low-end IoT devices** (low power, low memory, low price).

RIOT supports **SUIT** (RFC 9019): **S**ecure **U**pdates for the **I**nternet of **T**hings.
*Critical cryptographic component: **elliptic-curve digital signatures**.*

Question: what is the real cost of adding post-quantum security to SUIT?

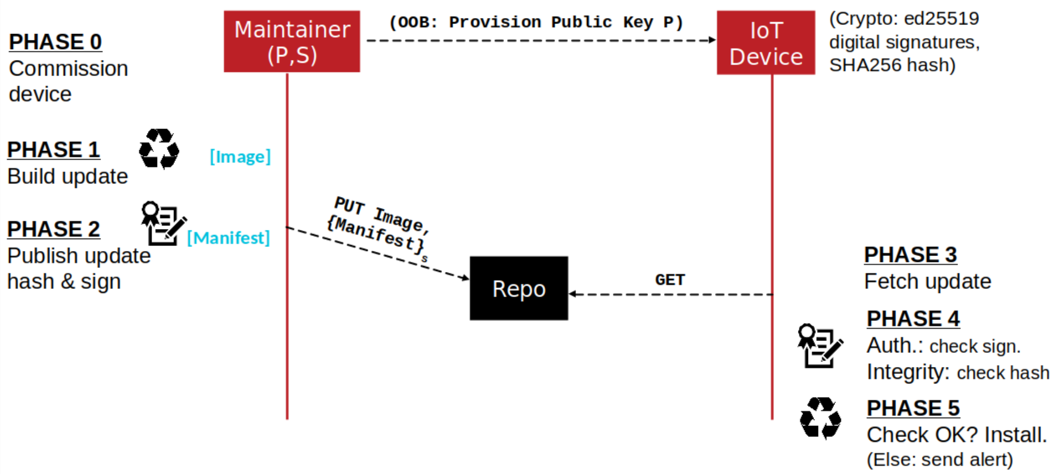
Banegas–Herrmann–Zandberg–Baccelli–**S.** (ACNS + RWC 2022): transverse study

→ **Dilithium** vs **Falcon** vs **LMS** vs Elliptic Curves

→ **ARM Cortex-M4** vs **ESP** vs **RISC-V**

→ Small firmware updates vs full software packages

SUIT: Software Updates for the Internet of Things



What we measured: pre- and post-quantum signature schemes

Pre-quantum baseline (SUIT standard) and **Post-quantum alternatives**

	Private key		Public key		Signature		SUIT Manifest	
Algorithm	Bytes	*Ratio*	Bytes	*Ratio*	Bytes	*Ratio*	Bytes	*Ratio*
Ed25519 *or* **ECDSA**	32	*1×*	32	*1×*	64	*1×*	483	*1×*
Dilithium	2528	*79×*	1312	*41×*	2420	*37.8×*	2839	*5.88×*
Static[1] **Dilithium**	18912	*591×*	17696	*553×*				
Falcon	1281	*40×*	897	*28×*	666	*10.4×*	1085	*2.24×*
LMS[2] (RFC8554)	64	*2×*	60	*0.94×*	4756	*74.3×*	5175	*10.7×*
SQIsign	353	11×	65	2×	148	2.31×	567	1.17×

[1] *Static Dilithium* = matrices expanded from seed and stored.

[2] LMS = Leighton–Micali, stateful hash-based signatures. State is not a problem for this application.

Three boards representing the 32-bit microcontroller landscape

RIOT supports $\geq$ **272 platforms**: we have to emphasize **portability**.

- No assembly, no platform-specific tricks.
- Open implementations (notably `PQClean`)
- Minimal modifications for RIOT compatibility: removing `malloc`, etc.

We took **three** representative 32-bit boards:

Architecture	Board	Speed	RAM (kB)	Flash (kB)
ARM Cortex-M4	Nordic nRF52480	64MHz	256	1024
Espressif ESP32	WROOM-32	80MHz	520	448
RISC V	Sipeed Longan Nano	72MHz	32	128

Signature benchmarks: Verification on ARM Cortex-M4

Algorithm	Base library	Flash (B)	Stack (B)	Time (ms)
Ed25519	C25519	5106	1300	1953
Ed25519	Monocypher	13852	1936	40
ECDSA	Tinycrypt	6498	1024	313
Dynamic **Dilithium**	PQClean	11664	**36058**	53
Static **Dilithium**	PQClean	**26672**	**19504**	23
Falcon	PQClean	**57613**	4744	15
LMS (RFC8554)	Cisco	12864	1580	**123**
SQIsign	Reference -O3	FIXME	**31016**	2483
	Reference -Os	FIXME	**30604**	3575

- Similar figures for ESP32 and RISC-V
- Dynamic Dilithium cannot run on the Sipeed Nano (RISC-V): only 32kB RAM

Impact on data transfer

Example: suppose we want to update RIOT firmware for the `nRF52480` board. The firmware itself is a ≈ 46kB binary, and the (pre-quantum) crypto is ≈ 6kB.

How much data do we need to transmit?

SUIT				Data Transfer	
Signature	Hash	Flash	Stack	**no crypto**	**crypto incl.**
Ed25519	SHA256	52.4kB	16.3kB	47kB	53kB
Dilithium	SHA3-256	+30%	+210%	+4.3%	+34%
Falcon	SHA3-256	+120%	+18%	+1.1%	+120%
LMS	SHA3-256	+34%	+1.2%	+9%	+43%

Recommendations for four typical software updates

1. **Small software module update**: $\approx$ 5kB $\implies$ prefer **Falcon**
   Speed and signature size are critical

Recommendations for four typical software updates

1. **Small software module update**: ≈ 5kB $\Longrightarrow$ prefer **Falcon**
   Speed and signature size are critical
2. **Small firmware update** ≈ 50kB *without* crypto libs $\Longrightarrow$ prefer **Falcon**
   Again, speed and signature size are critical

Recommendations for four typical software updates

1. **Small software module update**: $\approx$ 5kB $\implies$ prefer **Falcon**
   Speed and signature size are critical
2. **Small firmware update** $\approx$ 50kB *without* crypto libs $\implies$ prefer **Falcon**
   Again, speed and signature size are critical
3. **Small firmware update** $\approx$ 50kB *plus* crypto libs $\implies$ prefer **LMS**
   Larger crypto lib transfer $\implies$ *higher energy cost on low-power networks.*
   It takes 30-60s to transfer 50kB on a low-power IEEE802.15.4 radio link,
   but signature verification only varies by 2s between all candidates...
   LMS has the best tradeoff between code size, stack, network costs, and speed

Recommendations for four typical software updates

1. **Small software module update**: $\approx$ 5kB $\implies$ prefer **Falcon**
   Speed and signature size are critical
2. **Small firmware update** $\approx$ 50kB *without* crypto libs $\implies$ prefer **Falcon**
   Again, speed and signature size are critical
3. **Small firmware update** $\approx$ 50kB *plus* crypto libs $\implies$ prefer **LMS**
   Larger crypto lib transfer $\implies$ *higher energy cost on low-power networks.*
   It takes 30-60s to transfer 50kB on a low-power IEEE802.15.4 radio link,
   but signature verification only varies by 2s between all candidates...
   LMS has the best tradeoff between code size, stack, network costs, and speed
4. **Large firmware update** $\approx$ 250kB $\implies$ **no preference**
   Network transfer costs overwhelm other factors, reducing relative advantages

Conclusions

Post-quantum IoT software updates with SUIT are **feasible now**.

- **Falcon** is best for smaller module and firmware updates;
- **LMS** is better when the crypto lib is transferred;
- but there is no clear winner for much larger updates.

`https://ia.cr/2021/781`

Consider using RIOT for easy, portable, open IoT crypto development.

`https://riot-os.org/`