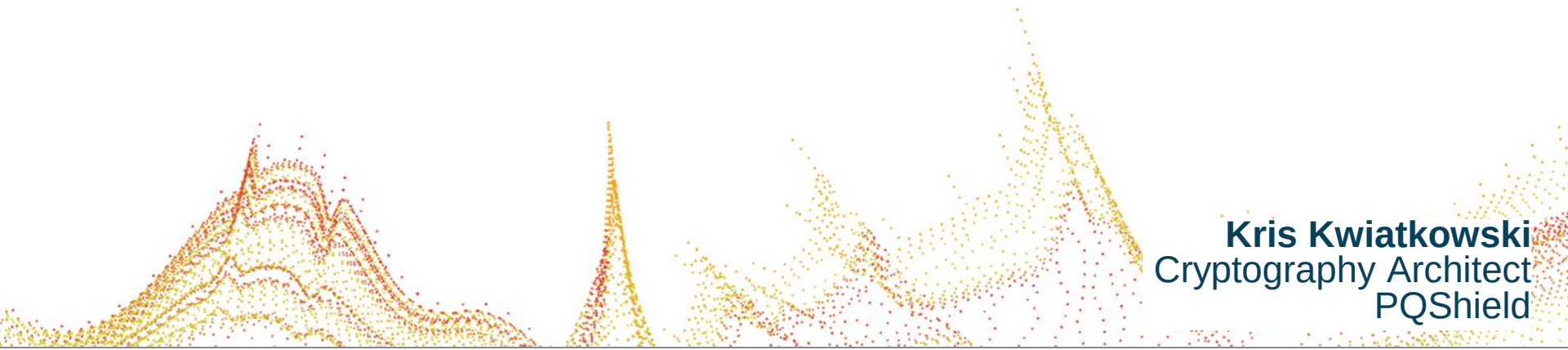
Post-Quantum Cryptography
The For IoT Edge



Kris Kwiatkowski
Cryptography Architect
PQShield

One-Minute Introduction to PQC

Released in October ’20

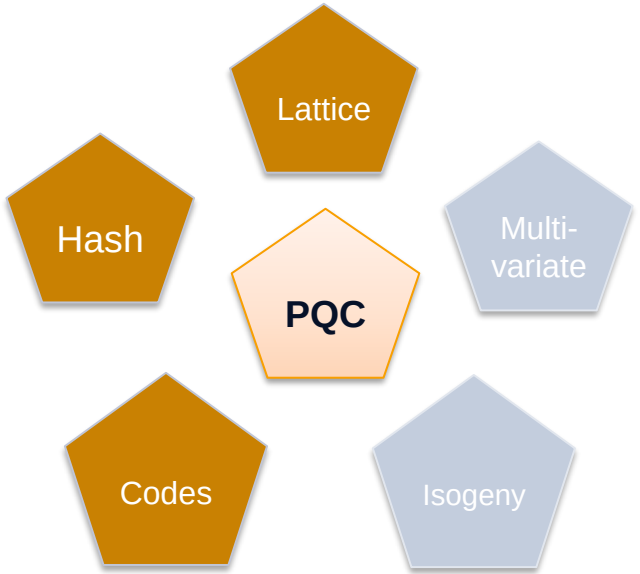
- SP800-208 - **LMS** (RFC8554) and **XMSS** (RFC 8391) . Statefull Hash-based Digital Signatures, standardized by IETF already in 2019. Part of CNSA 2.0 suite, to be used for software/firmware updates

Released in August ’24

- FIPS 203 - **ML-KEM** ("Kyber") for Key Establishment. Replaces EC Diffie-Hellman key exchange (example: TLS handshake) and RSA in Encryption.
- FIPS 204 - **ML-DSA** ("Dilithium") for Signatures. Replaces {Ed,EC}DSA and RSA signatures in web authentication, PKI certificates.
- FIPS 205 - **SLH-DSA** ("SPHINCS+") Stateless Hash-based Digital Signature Algorithm. Likely to see use in "root of trust" applications

To be released

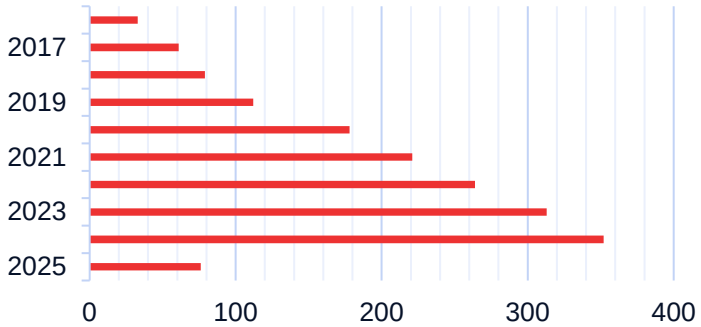
- FIPS 206 **FN-DSA** ("Falcon") for Signatures. Replaces use cases that prefer smaller and faster schemes at the cost of higher complexity.
- **HQC** - Key Encapsulation Mechanism, selected ‘25, finalist of Round 4
- “Additional” signature schemes are going to be standardised latter



Standardization

Academia

A decade of research has built strong confidence in PQC security.



Number of PQC publications according to DBLP

Industry / Protocol Standardization

NIST

- ‘16 NIST starts a project to standardize quantum-resistant cryptography

Feedback from deployments and experimentations

- ’16, ’19 & ’21: Experimental TLS deployments of CECPQ1/2 schemes by Google and Cloudflare
- ’23 Signal updates X3DH protocol design to include PQ
- ’24 Apple upgrades iMessage to use PQ3 protocol
- ‘24 Large-scale deployment of PQ key exchange by major browser vendors and cloud providers.

IETF

- ’20 Hybrid-PQ TLS and IKEv2 start to be discussed
- ’22 IETF starts PQC effort to integrate PQC in PKI
- ‘25 PQC became a mainstream topic in TLS, X.509, VPN, SSH working groups

Challenges of PQC deployments on IoT

- **Longterm support is critical**
  - HW can't be updated
  - PQC implementations are still evolving
- **Key agreement**
  - Public key / ciphertext: ~25x bigger
- **Digital signature** (MLDSA, general purpose)
  - Public key: ~40x bigger
  - Signature: ~35 bigger

Key sizes: Key agreement

	Security*	Public key	Ciphertext	Secret
ECDH/p256	128	32 (x-only)	N/A	32
ML-KEM 512	128	800	768	32
ML-KEM 768	256	1184	1088	32
ML-KEM 1024	256	1568	1568	32

Key sizes: Digital signatures

	Security	Public key	Signature
ECDSA/p256	128	**32** (x-only)	64
LMS-SHA2-M32-H15-W8	256	52	1612
ML-DSA-65	192	**1952**	3309
SLH-DSA-SHA2-256s	256	64	**29792**

Challenges of PQC deployments on IoT

- **Memory**
  - Typically 8-16KB RAM in constrained devices
- **Performance**
  - Optimizing memory sometimes impacts performance.

Memory usage

	Keygen	Sign	Verify
EdDSA	7.5	7.5	3
ML-DSA-65 (small**)**	4.5	11.8	4.6
ML-DSA-65 (fast)	60.8*	68.8*	9.8*
ML-KEM-768 (small)	9	9.1	12.8
ML-KEM-768 (fast, x86)	19.6	19.9	26.7

Performance (10k cycles, Cortex-M4)

	Keygen	Sign or Encaps	Verify or Decpas
ML-DSA-65 (small**)**	3412*	24421*	5732*
ML-DSA-65 (fast)	2516*	6193*	2415*
ML-KEM-768 (portable C)	988*	1138*	1387*
ML-KEM-768 (small&fast)	644*	664*	714*

* Results of pqm4 library

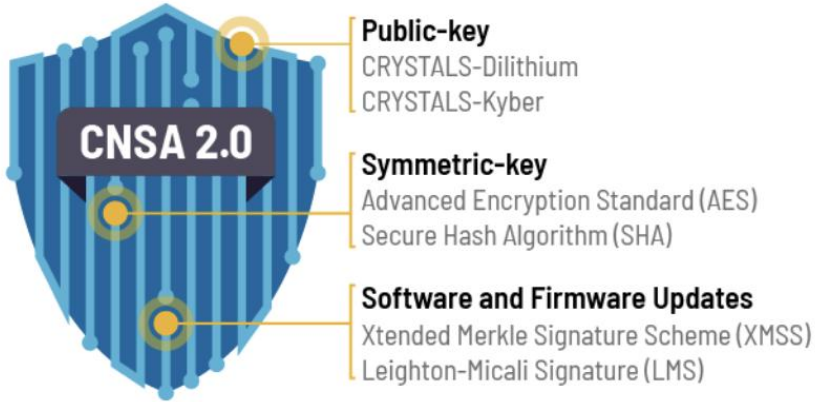
Use case

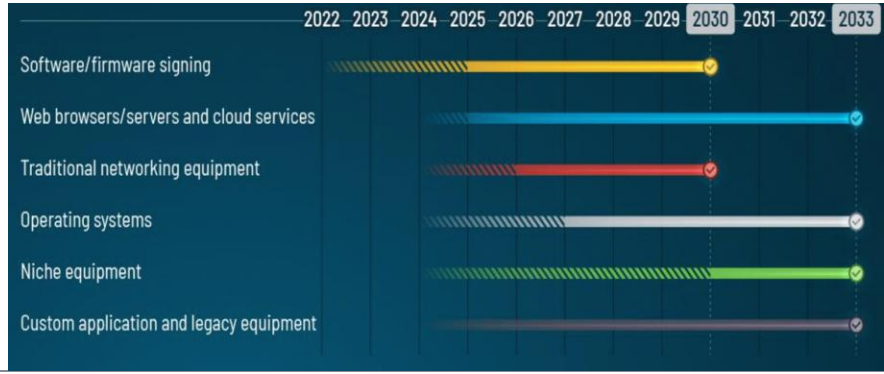
Let's assume the following theoretical use case for the "*embedded*" device that wants to exchange data with the cloud service.

- **Key agreement**
  - To agree on symmetric encryption keys
    => **ML-KEM** (FIPS 203)

- **Authentication**
  - Device uses mutual authentication to authenticate to the cloud service (i.e. TLS)
  - Signature size is important
    => **ML-DSA** (FIPS 204)

- **The secure boot of the embedded device**
  - The firmware is signed with the hash-based signature
  - The signing is done on the HSM
  - Verification of the firmware must be fast
    => **LMS** (RFC8554)





Lattice-based schemes

ML-KEM
Key exchange scheme

- Replacement for ECDH/RSA key agreement
- Three security levels: ML-KEM-{512, 768, 1024)
- Implementations work on matrices of size kxk (k=2,3,4)
- Vectors are composed of polynomials of degree 255 with coefficients in a ring Z_q, $q=13\cdot2^8+1$ (**12-bit**)
- Produces full entropy shared secret. No need to apply KDF to get full entropy.
- IND-CCA2 security: ensures the confidentiality of the plaintext and resistance against chosen-ciphertext attacks (higher bar vs ECDH)

ML-DSA
Digital signature scheme

- Replacement for ECDSA/EdDSA
- Three security levels: ML-DSA-{44,65,87}
- The design follows the *Fiat-Shamir with* ***Aborts*** framework introduced by Lyubashevsky
- Implementations work on vectors of size kxl (kxl=4x4,6x5,8x7)
- Vectors composed of polynomials of degree 255 with coefficients in a ring Z_q, $q=2^{23}+2^{13}+1$ (**23-bit**)
- Uses uniformly-distributed random number sampling over small **integers** for computing coefficients in error vectors

ML-KEM and ML-DSA

Lattice-based, key encapsulation and digital signature algorithms

- Both schemes operate on a large matrix (*A*) of polynomials
  - **MLKEM-1024** uses matrix of 4x4 polynomials
    - Each polynomial has 256 coefficients, each 12-bit
    - Total is **8KB** of memory "just" for *A*
  - **MLDSA-87** uses matrix A of 8x7 polynomials
    - Each polynomial has 256 coefficients, each 23-bit
    - Total is **56KB** of memory "just" for *A*
  - Additional memory required by vectors r,e_1,e_2 and temporary results

Input : $pk = (\boldsymbol{t}, \boldsymbol{\rho})$, $\boldsymbol{m} \in \{0,1\}^{256}$, random $\boldsymbol{\mu} \in \{0,1\}^{256}$

Output: ciphertext (u, v)

$A \in R_q^{k \times k} \leftarrow sampleUniform(\boldsymbol{\rho})$

$r \in R_q^{k} \leftarrow sampleCBD^{\eta 1}(\boldsymbol{\mu})$

$e_1 \in R_q^{k}$, $e_2 \in R_q^{k} \leftarrow sampleCBD^{\eta 2}(\boldsymbol{\mu})$

$\boldsymbol{u \leftarrow A^T r + e_1}$

$\boldsymbol{v} \leftarrow t^T r + e_2 + \text{Encode}(m)$

$\boldsymbol{U'} = \text{Encode}(\textbf{Compress}_q(\boldsymbol{u}, du))$,

$\boldsymbol{V'} = \text{Encode}(\textbf{Compress}_q(\boldsymbol{v}, dv))$

MLKEM.Encrypt

Return ct = $(\boldsymbol{U'}, \boldsymbol{V'})$

ML-DSA

- The design follows the *Fiat-Shamir with* ***Aborts***

- Signing time is variable and depends on:
  - Public key
  - Message being signed
  - The random value generated during the signing

- FIPS-204 provides the expected number of loops per parametrization as well as guidance regarding max number of repetitions.

Repetitions (see explanation below)	4.25	5.1	3.85

$\underline{\mathsf{Sign}(sk, M)}$

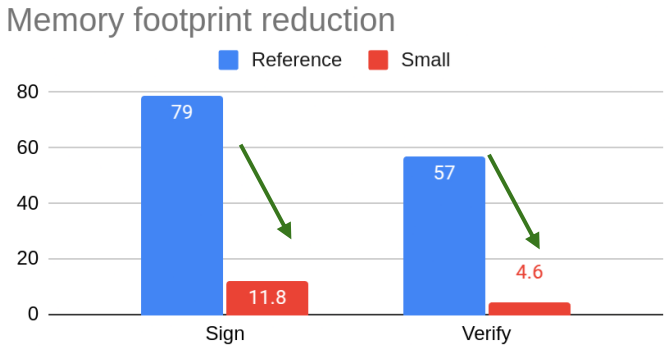
09 $\mathbf{A} \in R_q^{k \times \ell} := \mathsf{ExpandA}(\rho)$ ▷ $\mathbf{A}$ is ge
10 $\mu \in \{0,1\}^{384} := \mathsf{CRH}(tr \parallel M)$
11 $\kappa := 0,\ (\mathbf{z}, \mathbf{h}) := \perp$
12 **while** $(\mathbf{z}, \mathbf{h}) = \perp$ do ▷ Pre-comput
13 $\mathbf{y} \in S^{\ell}_{\gamma_1 - 1} := \mathsf{ExpandMask}(K \parallel \mu \parallel \kappa)$
14 $\mathbf{w} := \mathbf{Ay}$
15 $\mathbf{w}_1 := \mathsf{HighBits}_q(\mathbf{w}, 2\gamma_2)$
16 $c \in B_{60} := \mathsf{H}(\mu \parallel \mathbf{w}_1)$
17 $\mathbf{z} := \mathbf{y} + c\mathbf{s}_1$

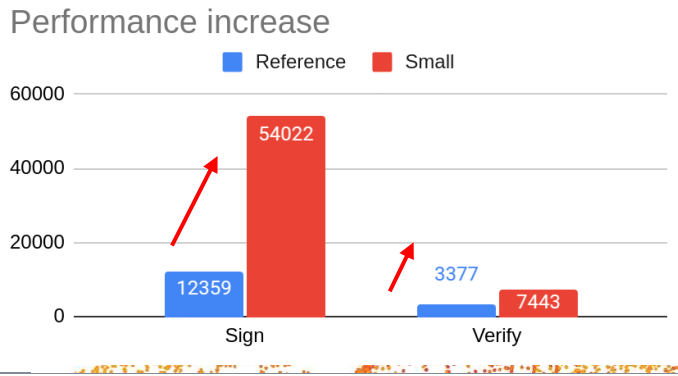
Low Memory Footprint Implementations

ML-DSA-65

Solutions

- **Streamlining** of A*y operation.
  - Interleaved matrix ***A*** expansion and matrix-by-vector multiplication [2]
- **Use of flash**
  - Key generated once and then reused for signing, hence we can store whole matrix
  - s1, s2 and t0 use NTT representation - store them directly in flash after key generation (part of prv key)
- Compression polynomial coefficients to buffers
- Usage of **seed** for generating private key
  - Avoids to store private key in expanded form
- Use **working buffer** - pre-allocated external memory for storing temporary variables.





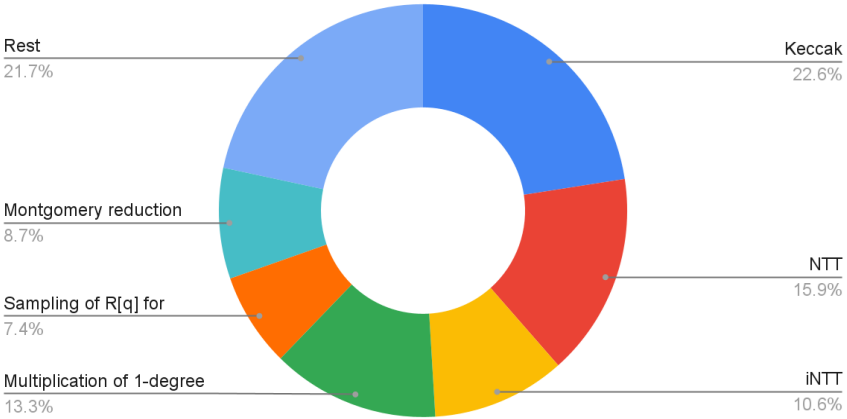
ML-DSA-65

Analysis of hot-spots

ML-KEM/ML-DSA in software

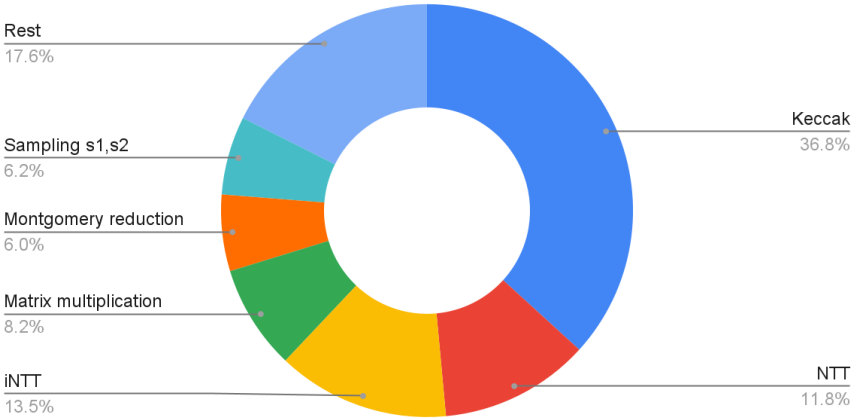
MLKEM-768

aarch64, gcc-10, -O3



MLDSA-65

aarch64, gcc-10, -O3



Runtime determined by:

- SHA3/SHAKE, closer to 50% when implemented in memory-constrained environments
- Polynomial arithmetic

Keccak (SHA3/SHAKE)

Performance improvements

Keccak (SHA3/SHAKE) is a main main optimization target

- Expansion of matrix A is a big contributor to runtime
  - MLKEM-768: (3x3)x256 coefficients
  - MLDSA-65: (6x5)x256 coefficients
- Fast Keccak could speed up matrix A generation, as well as rejection sampling

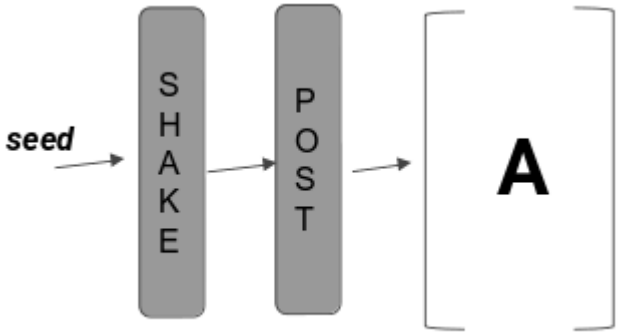
HW-assisted implementations of SHA-3 possible today (ARM):

- Possibility to leverage **BIC** instruction from ARM ISA (**A&~B**) and **ROR** with barrel shifter

For all triples (x, y, z) such that $0 \le x < 5$, $0 \le y < 5$, and $0 \le z < w$, let

$$\mathbf{A}'[x,y,z] = \mathbf{A}[x,y,z] \oplus ((\mathbf{A}[(x+1) \bmod 5, y, z] \oplus 1) \cdot \mathbf{A}[(x+2) \bmod 5, y, z]).$$

- SIMD can be used to perform Keccak on multiple inputs in parallel



HW-based SHA-3 accelerator to improve performance!

LMS: Leighton-Micali Signature Scheme

Hash-based, stateful, signature scheme (NIST SP800-208)

Structure of the key

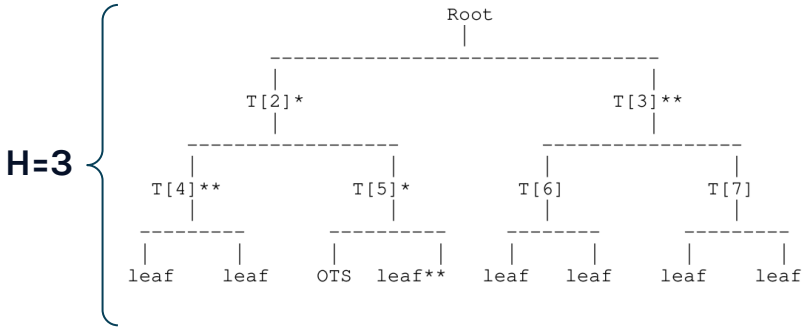
- Leaves represent a one-time event called LMOTS
- All *T[i]* are hash of two child leaves
- “Root” - a public key

Signing

- Message is signed with LMOTS secret key
- Authentication path: `leaf**, T[4]**, T[3]**`
- The signature includes index of the leaf

Verification

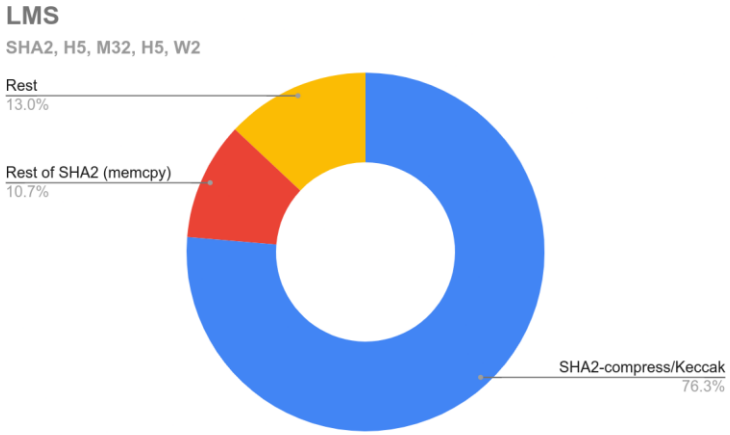
- LMOTS public key used to verify OTS part
- Hash of the authentication path
- Check if results is same as `Root`

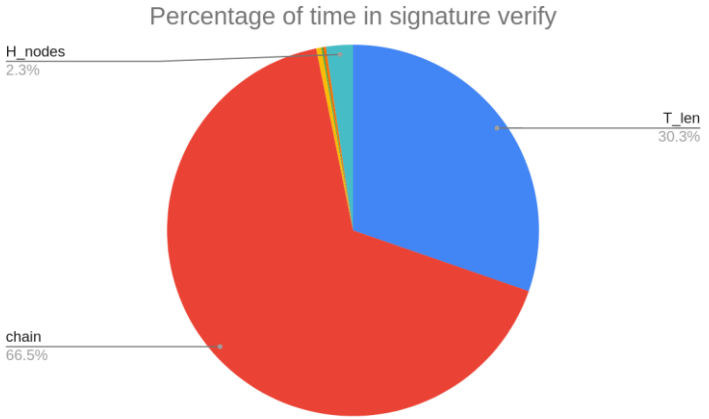


Number of signatures: $2^H = 8$

LMS performance

- Performance is largely dominated by runtime of the hash function
- A lot of operation on small chunks of memory






```
3. Compute the string Kc as follows:
   Q = H(I || u32str(q) || u16str(D_MESG) || C || message)
   for ( i = 0; i < p; i = i + 1 ) {
     a = coef(Q || Cksm(Q), i, w)
     tmp = y[i]
     for ( j = a; j < 2^w - 1; j = j + 1 ) {
       tmp = H(I || u32str(q) || u16str(i) || u8str(j) || tmp)
     }
     z[i] = tmp
   }
   Kc = H(I || u32str(q) || u16str(D_PBLC) ||
                                z[0] || z[1] || ... || z[p-1])
```

LMS

Performance/size tradeoffs

- Large number of parametrizations (80)
- Can be instantiated with SHA2 or SHAKE256
- Number of signatures
- Operation runtime
- Signature size
- Security based on the security of hash functions

- Secure boot is a main use-case
  - Fast verification
  - Signature and key generated in controlled environment

Pitfalls

- Stateful scheme
- Reuse of LMOTS key for signing two different messages compromises security guarantees
  - *Solution*: **SLH-DSA** (FIPS-205)
- Limited applicability (not suitable for generic use)
- Software implementations not FIPS-approved

- Slow and memory “hungry” key generation and signing time (need to rebuild Merkle Tree)

Recommended by NSA in the **CNSA 2.0 for firmware signing.**

Conclusion & Takeaway

- **Classical : ECC**
  - The **“Swiss knife”** of crypto - small, fast, secure
  - One simple formula: [a*b]*P *in GF(p)*

- **Post-Quantum Challenges:**
  - Bigger keys and signatures.
  - Higher memory usage
  - Complex schemes with multiple components
    - Heavy reliance on Keccak
  - Secure implementations are much harder

- **Current State & Direction:**
  - PQC implementations still evolving
  - HW/SW co-design can reuse core building blocks
  - **<u>The key challenge: finding the right balance between scheme, application and implementation technique</u>**

- More schemes under development - the landscape is still changing

PQShield: PQC Security Suite

Thank you for your time

Questions?