Post-Quantum Cryptography
The For IoT Edge

Kris Kwiatkowski
Staff Cryptography Architect
Cryptography Team of PQShield

Background story

1994

Peter Shor

Introduces quantum attack on classical **asymmetric** cryptosystems.

In practice, it means all currently deployed cryptosystems can be broken on *large-scale* quantum computers.

1996

Lov Grover

Introduces quantum algorithm which improves searching in the unordered set, introducing a potential threat to **symmetric** cryptography algorithms.
The problem can be easily solved by switching to twice longer secret keys.

2016

NIST starts the selection process

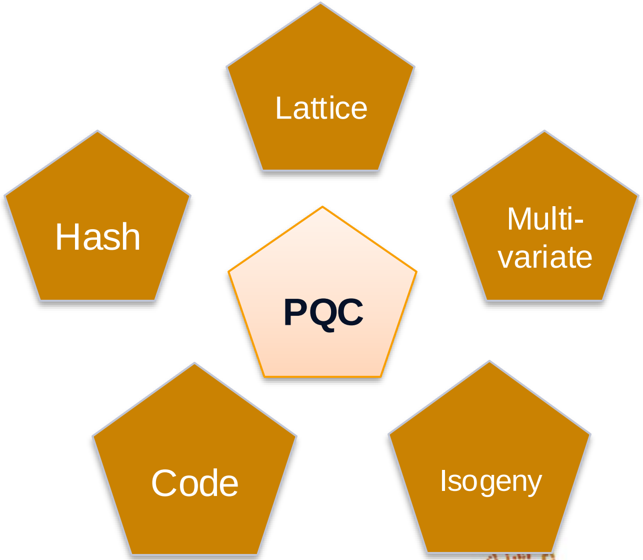
NIST (National Institute of Standards and Technology) starts a multi-year project to select new **asymmetric** cryptosystems resistant to potential attacks by quantum adversaries. The planned end date is Jan 2024.

NIST PQC standardization process

In 2016, NIST requested 2 types of asymmetric cryptosystems for:
- Digital signature
- Key Encapsulation Mechanism (KEM) for key agreement

The 3-round process with each round lasting for around 2 years
- **82** schemes were submitted, **69** candidates were accepted, 5 different categories, each representing a different underlying hard problem
- Round 1 ('18) - 64 accepted (19 digital signatures / 45 KEMs)
- Round 2 ('20) - 26 accepted (9 digital signatures / 17 KEMs)
- Round 3 ('23) - 4 schemes selected for standardization in '24/'25 (3 digital signature and 1 KEM scheme)

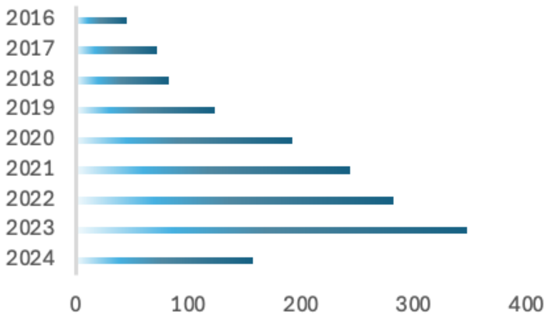
Different and more complicated than AES/SHA-3 standardization
- Larger problem space
- Must integrate well with comms / Internet protocols
- KEM is not a drop-in replacement for DH



Support from cryptographic community

Academia

- PQC has been a very active research area in the past few decades
- Main contribution - design and cryptoanalysis of the candidate schemes
- Number of PQC papers according to DBLP:



Industry / Protocol Standardization

- Feedback from deployments and experimentations
  - ’16, ’19 & ’21: Experimental TLS deployments of CECPQ1/2 schemes by Google and Cloudflare
  - ’23 Google enables Kyber in the Chrome browser
  - ’23 Signal updates X3DH protocol design to include PQ
  - ’24 Apple upgrades iMessage to use PQ3 protocol
  - ’24 Zoom announces rollout of PQC for E2EE
- IETF:
  - ’20 Hybrid-PQ TLS and IKEv2 start to be discussed
  - ’22 IETF starts PQC effort to integrate PQC in PKI
- NIST
  - ’23 NCCoE releases SP1800-38 describing migration to post-quantum

New NIST Standards of PQ cryptographic schemes

Released in October '20

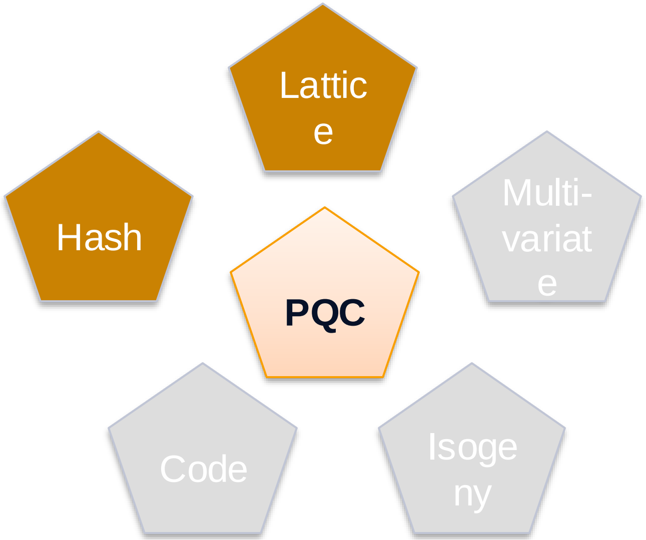
- SP800-208 - **LMS** (RFC8554) and **XMSS** (RFC 8391) . Statefull Hash-based Digital Signatures, standardized by IETF already in 2019. Part of CNSA 2.0 suite, to be used for software/firmware updates

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"), KEMs from Round 4 and additional signature schemes are going to be standardized latter



Lattice

Hash

Multi-variate

PQC

Code

Isogeny

Future PQC cryptographic schemes

Round 4 KEM

- NIST to choose additional KEM scheme: ***BIKE, Classic McEliece, HQC***

Additional PQC Digital Signature Scheme Candidates

- NIST started a new process for PQC standardization to diversify the digital signature alternatives. In the coming years, they aim to standardize new post-quantum signature schemes which support short signatures and fast verification.

Non-NIST competitions

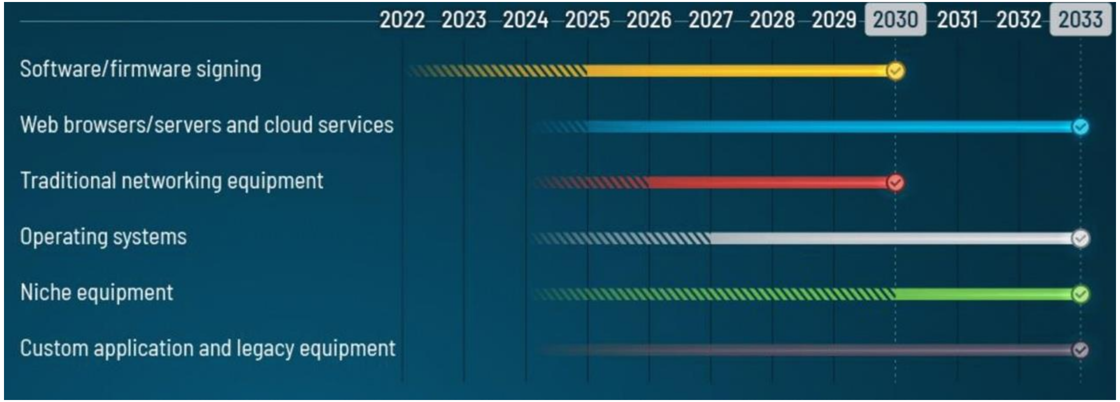
- ISO committee will standardize *Classic* ***McEliece*** and ***FrodoKEM***
- CACR (*Chinese Association for Cryptologic Research)*
  - Held a competition to identify post-quantum cryptographic algorithms during 2018 and 2019
  - Two lattice-based schemes were selected (**Aigis-enc**, **Aigis-sig** and **LAC.PKE**)
- KpqC (Korean Post-Quantum Cryptography) started in 2022 and is ongoing.

Others…

Quantum Computers

It is **irrelevant** whether Cryptographically Relevant Quantum Computers are a threat to public key crypto.

Implementers will need to align with standards.



- The addition of those schemes to FIPS 140-3 certification builds the credibility further
- CNSA 2.0 requirements:
  - New software and firmware signing with PQ by 2025.
  - Transitioning all deployed software and firmware to CNSA 2.0-compliant signatures by 2030.
  - Ideally, before quantum computers are available…

PQC vs Classical Crypto

Key agreement

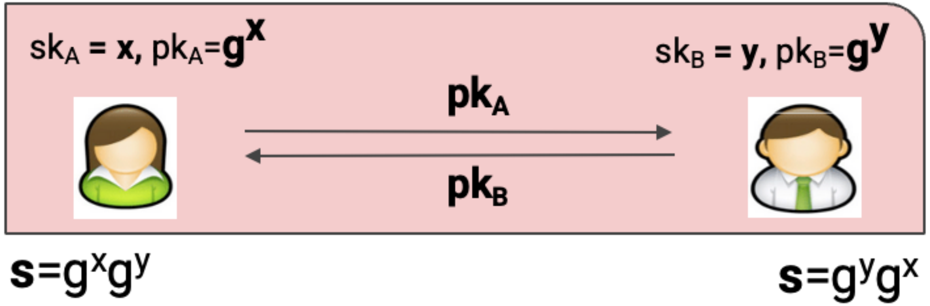
KEM Interface

- *Triple of algorithms*: key generation, encapsulation, decapsulation
- *Asymmetric*: Encapsulation outputs 2 results, decapsulation outputs 1
- Doesn't fit into DH interfaces

IND-CCA2 security

- Shared secrets are always indistinguishable from random ones (even if the attacker can decapsulate arbitrary ciphertexts)
- Security against an active attacker

DH



$sk_A = \mathbf{x}$, $pk_A = \mathbf{g^x}$

$sk_B = \mathbf{y}$, $pk_B = \mathbf{g^y}$

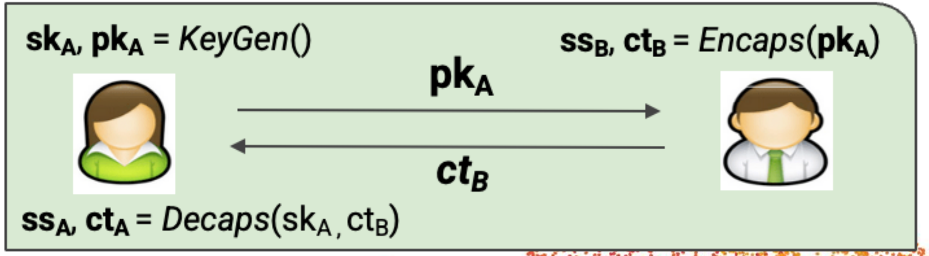
$\mathbf{pk_A}$

$\mathbf{pk_B}$

$\mathbf{s = g^x g^y}$

$\mathbf{s = g^y g^x}$

KEM



$\mathbf{sk_A, pk_A} = KeyGen()$

$\mathbf{ss_B, ct_B} = Encaps(\mathbf{pk_A})$

$\mathbf{pk_A}$

$\mathbf{ct_B}$

$\mathbf{ss_A, ct_A} = Decaps(sk_A, ct_B)$

PQC vs Classical Crypto

Digital signatures

Pre-Hash

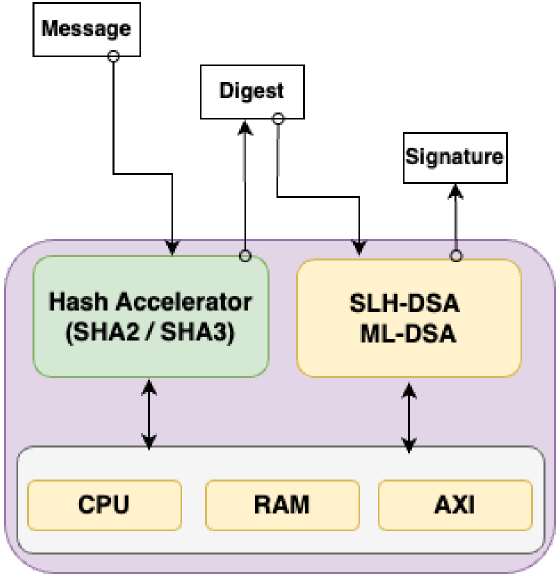
- Messages can be pre-hashed with hash accelerators. Signing/Verification algorithm works directly on a digest of a message
- Specified for ML-DSA and SLH-DSA (as well as EdDSA)

ML-DSA: Variable signing time

- The signing function performs rejection sampling until generated values are in the expected range.

LMS/XMSS: State management

- The private key is associated with the state



PQC vs Classical Crypto

Sizes

Digital signatures

	Security	Public key	Signature
ECDSA/p256	128	32 (x-only)	64
LMS-SHA2-M32-H15-W1	256	52	9004
LMS-SHA2-M32-H15-W8	256	52	1612
ML-DSA-44	128	1312	2420
ML-DSA-65	192	1952	3309
ML-DSA-87	256	2592	4627
SLH-DSA-SHA2-128s	128	32	7856
SLH-DSA-SHA2-128f	128	32	17088
SLH-DSA-SHA2-192s	192	48	16224
SLH-DSA-SHA2-256s	256	64	29792

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 agreement**
  - Public key / ciphertext: ~25x bigger
- **Digital signature** (MLDSA, general purpose)
  - Public key: ~40x bigger
  - Signature: ~35 bigger

- Elliptic Curve Cryptography was a “**Swiss knife**” for most crypto applications
    - Small, fast, secure…

- The challenge with post-quantum cryptography is to find the right **balance** between scheme, application and implementation technique

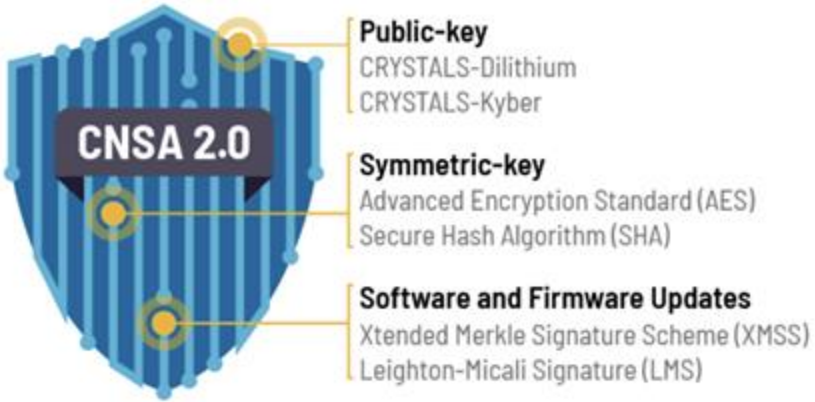
Use case

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

- **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)

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

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



LMS: Leighton-Micali Signature scheme

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

Structure of the key

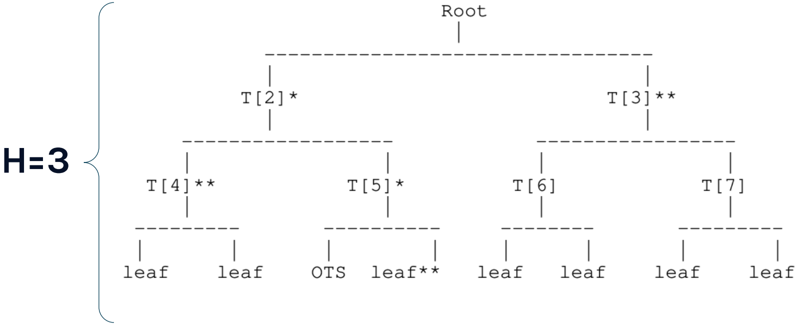
- Leaves represent a one-time event called LMOTS
- All *T[i]* are hash of two child leaves
- “Root” - a public key

Signing

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

Verification

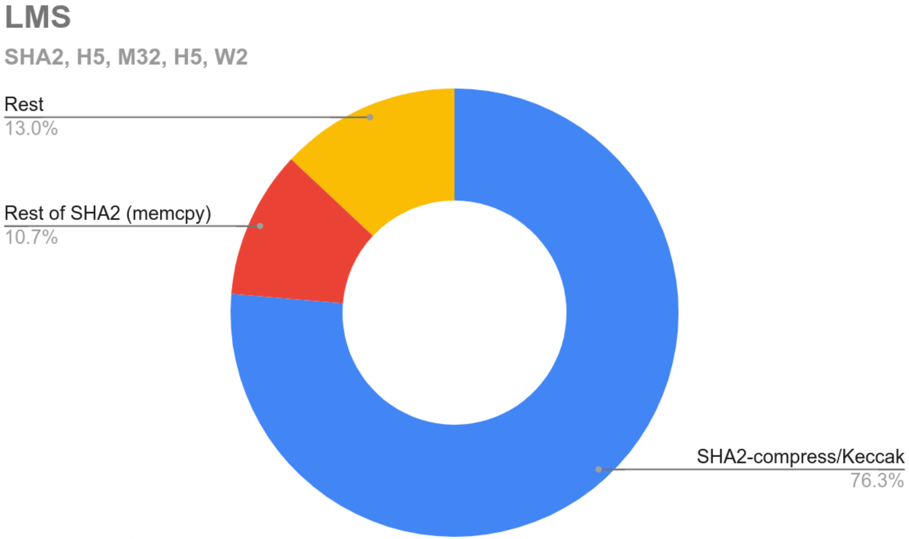
- LMOTS public key used to verify OTS part
- Hash of the authentication path
- Check if results is same as `Root`



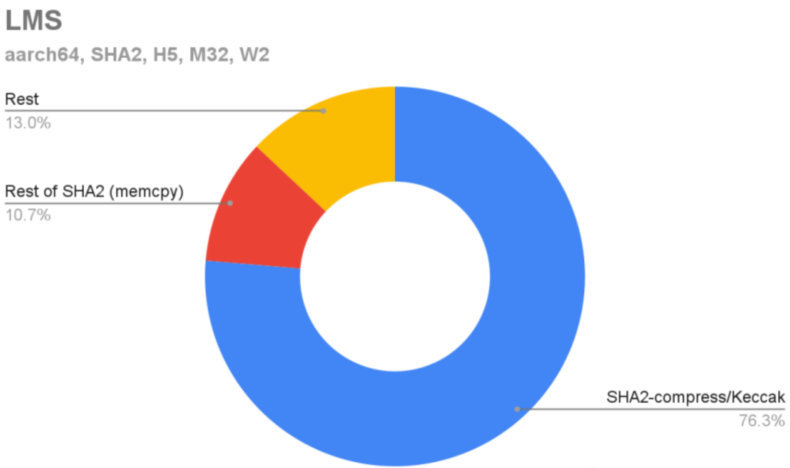
Number of signatures: $2^H = 8$

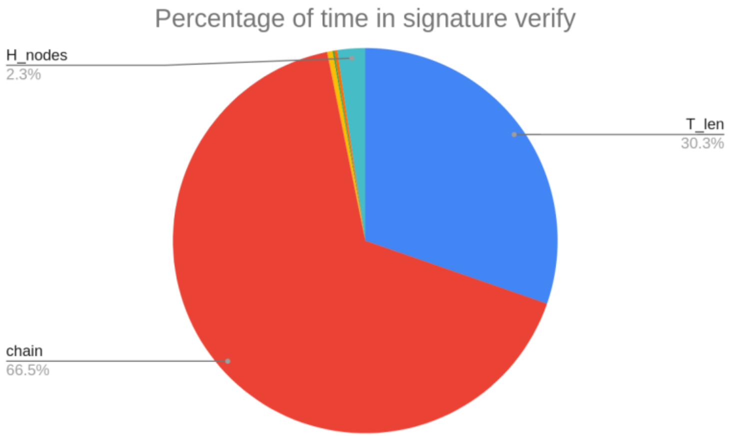
LMS performance

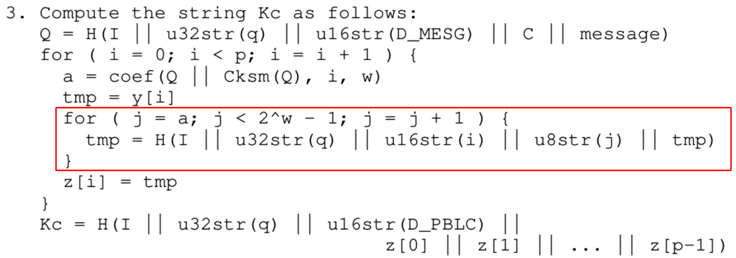
- Performance is largely dominated by runtime of the hash function
- A lot of operation on small chunks of memory



LMS performance







```
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

- Very fast verification
- Security based on the security of hash functions

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.**

ML-DSA

Lattice-based, digital signature scheme

- Based on the hardness of lattice problems over module lattices*
- The design follows the *Fiat-Shamir with **Aborts*** framework introduced by Lyubashevsky
- Uses uniformly-distributed random number sampling over small **integers** for computing coefficients in error vectors
  - Avoids using floating point arithmetic (difference with FN-DSA)
- Three security levels:
  - ML-DSA-**44**, ML-DSA-**65** and ML-DSA-**87**

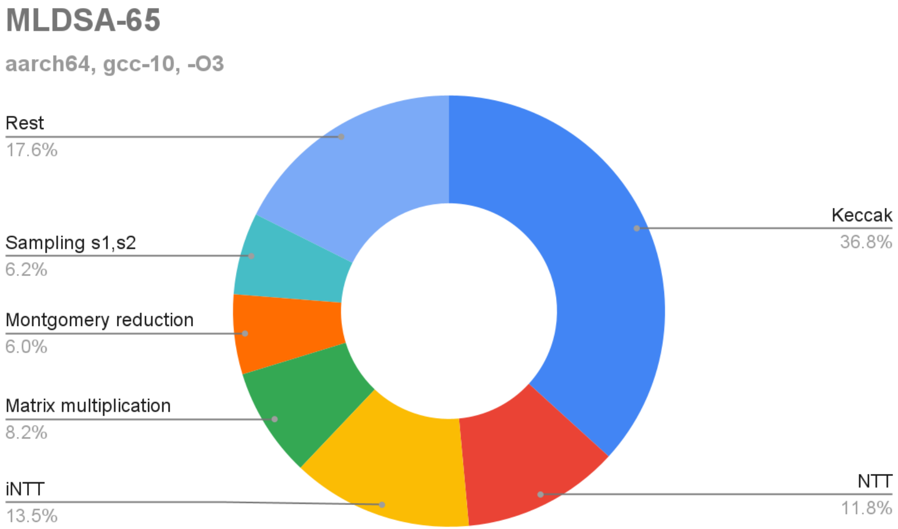
- Implementations work on vectors of size k and ℓ (k=4,6,8 and ℓ=4,5,7)
- Vectors represent polynomials of degree 255 with coefficients in a ring Z_q, with $q=2^{23} + 2^{13} + 1$ (23-bit)
- Use Number Theoretic Transform (NTT) for polynomial multiplication

---

*** *The Learning with Errors Problem*, O. Regev**
https://cims.nyu.edu/~regev/papers/lwesurvey.pdf

Analysis of hot-spots

ML-DSA in software



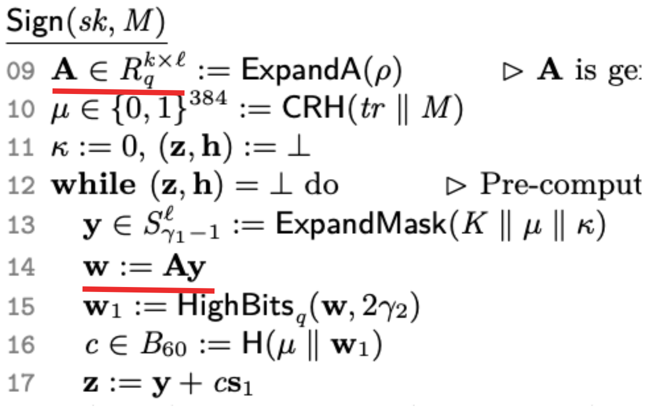
Runtime determined by:

- SHA3/SHAKE, closer to 50% when implemented on smaller devices
- Polynomial arithmetic (NTT)

MLDSA on Cortex-M

Operations

- Signing and key generation are much larger than verification
- 1KiB per polynomial (256 coefficients stored on int32_t)
- Seed must be expanded to large matrix (*A*)
  - MLDSA-44 uses matrix A of 4x4 polynomials
  - Two vectors of size 4
  - Signing operation requires ~51KB (non-opt)



```
Sign(sk, M)
$\mathbf{A} \in R_q^{k \times \ell} := \mathsf{ExpandA}(\rho)$            ▷ A is ge
$\mu \in \{0,1\}^{384} := \mathsf{CRH}(tr \,\|\, M)$
$\kappa := 0,\ (\mathbf{z}, \mathbf{h}) := \perp$
while $(\mathbf{z}, \mathbf{h}) = \perp$ do            ▷ Pre-comput
    $\mathbf{y} \in S_{\gamma_1 - 1}^{\ell} := \mathsf{ExpandMask}(K \,\|\, \mu \,\|\, \kappa)$
    $\mathbf{w} := \mathbf{A}\mathbf{y}$
    $\mathbf{w}_1 := \mathsf{HighBits}_q(\mathbf{w}, 2\gamma_2)$
    $c \in B_{60} := \mathsf{H}(\mu \,\|\, \mathbf{w}_1)$
    $\mathbf{z} := \mathbf{y} + c\mathbf{s}_1$
```

Some solutions

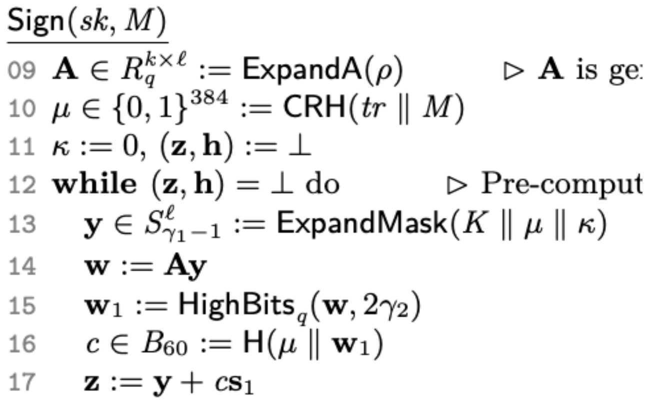
- Streamlining of A*y operation.
  - Interleaved matrix *A* expansion and matrix-by-vector multiplication [2]
- Use of flash
- Those optimizations may affect performance

Memory footprint for MLDSA-44

	Keygen	Sign	Verify
Reference	38	51	36
Optim [4]	6.4	6.5	2.7
EdDSA [9]	7.5	7.5	3

ML-DSA: *Pitfalls*

- 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.



```
Sign$(sk, M)$
$\mathbf{A} \in R_q^{k \times \ell}$ := ExpandA$(\rho)$        ▷ $\mathbf{A}$ is ge
$\mu \in \{0,1\}^{384}$ := CRH$(tr \parallel M)$
$\kappa := 0,\ (\mathbf{z}, \mathbf{h}) := \perp$
while $(\mathbf{z}, \mathbf{h}) = \perp$ do        ▷ Pre-comput
    $\mathbf{y} \in S_{\gamma_1 - 1}^{\ell}$ := ExpandMask$(K \parallel \mu \parallel \kappa)$
    $\mathbf{w} := \mathbf{A}\mathbf{y}$
    $\mathbf{w}_1$ := HighBits$_q(\mathbf{w}, 2\gamma_2)$
    $c \in B_{60}$ := H$(\mu \parallel \mathbf{w}_1)$
    $\mathbf{z} := \mathbf{y} + c\mathbf{s}_1$
```

Repetitions (see explanation below)	4.25	5.1	3.85

ML-KEM

Lattice-based, key encapsulation mechanism

- Based on the hardness of lattice problems over module lattices*
- IND-CCA2 security: ensures the confidentiality of the plaintext and resistance against chosen-ciphertext attacks (higher bar vs ECDH)
- Produces full entropy shared secret
  - No need to apply KDF to get full entropy
  - Still may be needed, but for a different reason*

*** See "Binding" property in the IETF draft draft-ietf-pquip-pqc-engineers**

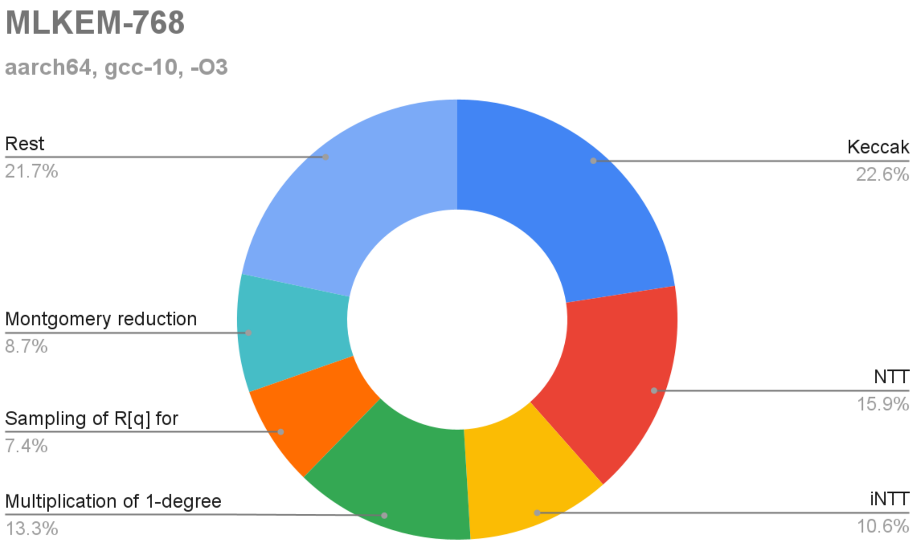
- Implementations work on vectors of size k (k=2,3,4)
- Three security levels:
  - ML-KEM-512, ML-KEM-768 and ML-KEM-1024
- Vectors represent polynomials of degree 255 with coefficients in a ring Z_q, $q=13 \cdot 2^8+1$ (12-bit)

Memory footprint for MLKEM-768

	ECDH/p256 (HW)	ML-KEM (SW)
RAM	1	x5
Timing	1	x4
Data transfer	1	x12

Analysis of hot-spots

ML-KEM in software

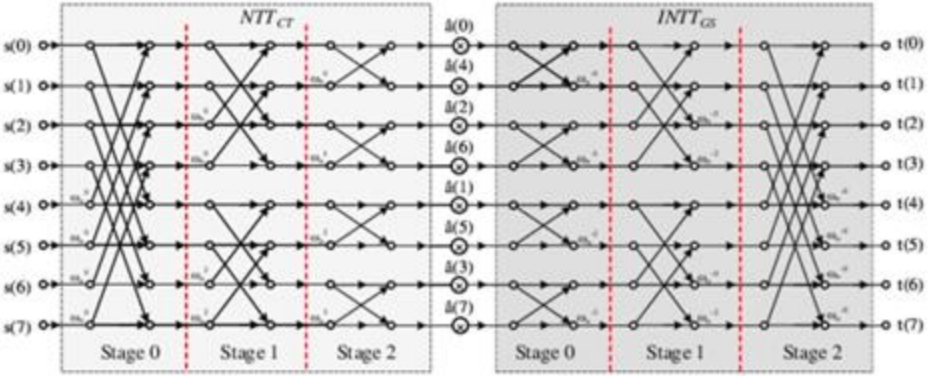


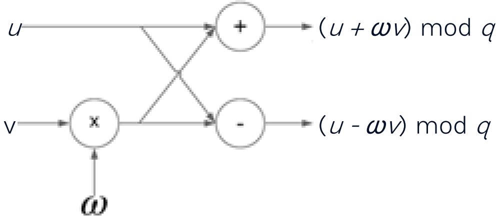
NTT

NTT = *Number Theoretic Transform*
(FFT in finite ring, usage similar as CRT in RSA)

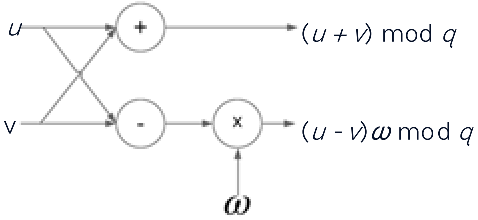
- Used in both MLKEM and MLDSA
- Complexity:
    - Transformation: O(n log n)
    - Multiplication : O(n)

- Polynomial arithmetic done in the NTT-domain
- $\mathbf{x} * \mathbf{y} = \mathrm{NTT}^{-1}(\mathrm{NTT}(\mathbf{x}) * \mathrm{NTT}(\mathbf{y}))$
- Example of usage in MLKEM:
  In theory - pubkey: $\mathbf{t} = \mathbf{A}\mathbf{s} + \mathbf{e}$
  But in MLKEM :
  $\hat{\mathbf{t}} = \mathrm{NTT}(\mathbf{A}) * \mathrm{NTT}(\mathbf{s}) + \mathrm{NTT}(\mathbf{e})$





Cooley-Tukey Butterfly



Gentleman-Sande Butterfly

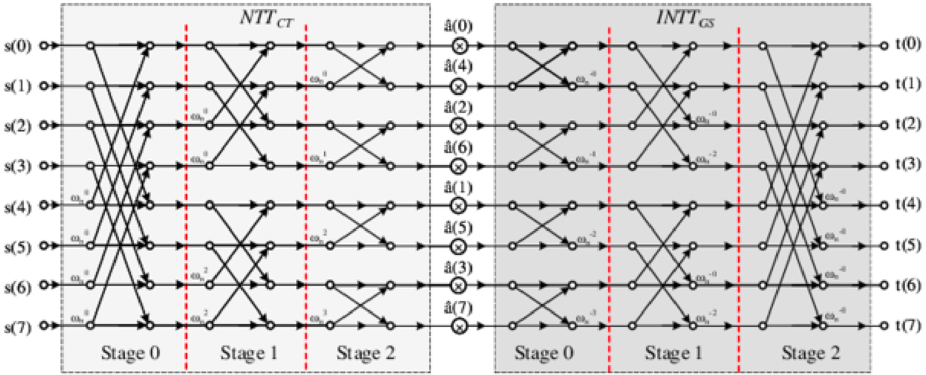
NTT - performance improvements

Scalar implementations (Cortex-M)

- Accumulate in double-width and reduce lazily, as late as possible [6],[3]
- Use `smull` and `smlal` for non-constant time Montgomery multiplication [1], [2]
- Balance between different multiplication methods **Plantard**[7] or **Montgomery**

Vectorized implementations (Cortex-M55/85)

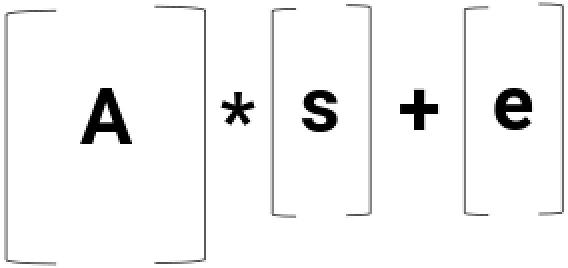
- Transform to NTT domain is amenable to vectorization with SIMD type of parallel processing



Keccak (SHA3/SHAKE) - performance improvements

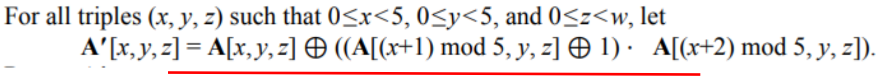
Keccak (SHA3/SHAKE) is a main main optimization target

- Expansion of matrix A is a big contributor to runtime
  - MLKEM-768: (3x3)x256 12-bit coefficients
  - MLDSA-65: (6x5)x256 23-bit coefficients
- Fast Keccak could speed up matrix A generation



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

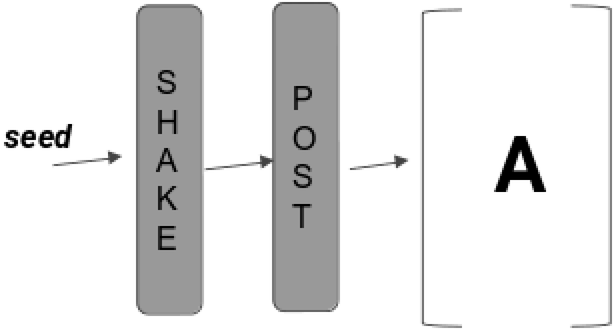
- Possibility to leverage BIC instruction from ARM ISA (A&~B) and ROR with barrel shifter



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

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

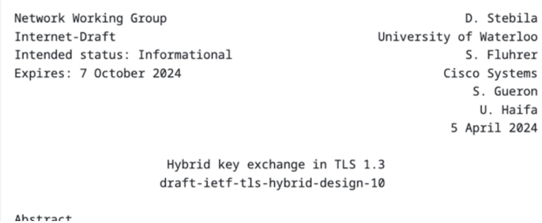
- SIMD can be used to perform Keccak on multiple inputs in parallel



HW-based SHA-3 accelerator to improve performance!

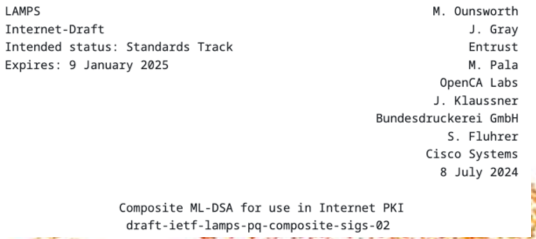
Conclusion

- ***Classical*** **- Elliptic Curve Cryptography**
    - Small and Fast - Crypto operation is 1 simple formula
        - ECDH shared secret = [a*b]*P *in GF(p)*
- ***Post-Quantum*** **– Lattice and Hash-based Cryptography**
    - Elements in a polynomial ring GF(p)[x]/(x^n + 1) ^k
    - Heavy use of hash functions
    - Bigger keys and signatures. Larger memory footprint.
- ***Hybrid schemes*** **– security in depth**
    - Techniques that mixing both PQ and classical schemes
    - Safe - migration strategy towards fully post-quantum schemes
    - Recommended by ANSSI, BSI, ETSI
    - Key agreement can be FIPS-certified (SP800-56Cr2)
    - Scheme ***X25519+MLKEM768*** is currently being deployed by Google and Mozilla in theirs browsers.



```
Network Working Group                                        D. Stebila
Internet-Draft                                  University of Waterloo
Intended status: Informational                                S. Fluhrer
Expires: 7 October 2024                                    Cisco Systems
                                                               S. Gueron
                                                                U. Haifa
                                                            5 April 2024

                      Hybrid key exchange in TLS 1.3
                      draft-ietf-tls-hybrid-design-10

Abstract
```

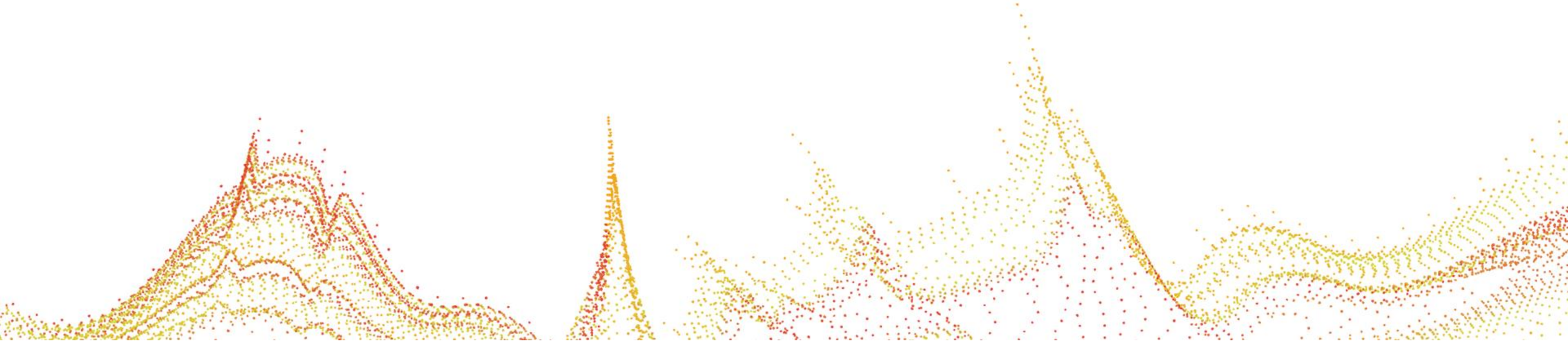
```
Transport Layer Security                                  K. Kwiatkowski
Internet-Draft                                                  PQShield
Intended status: Informational                             P. Kampanakis
Expires: 27 February 2025                                            AWS
                                                         B. E. Westerbaan
                                                               Cloudflare
                                                               D. Stebila
                                                  University of Waterloo
                                                          26 August 2024

     Post-quantum hybrid ECDHE-MLKEM Key Agreement for TLSv1.3
               draft-kwiatkowski-tls-ecdhe-mlkem-01
```



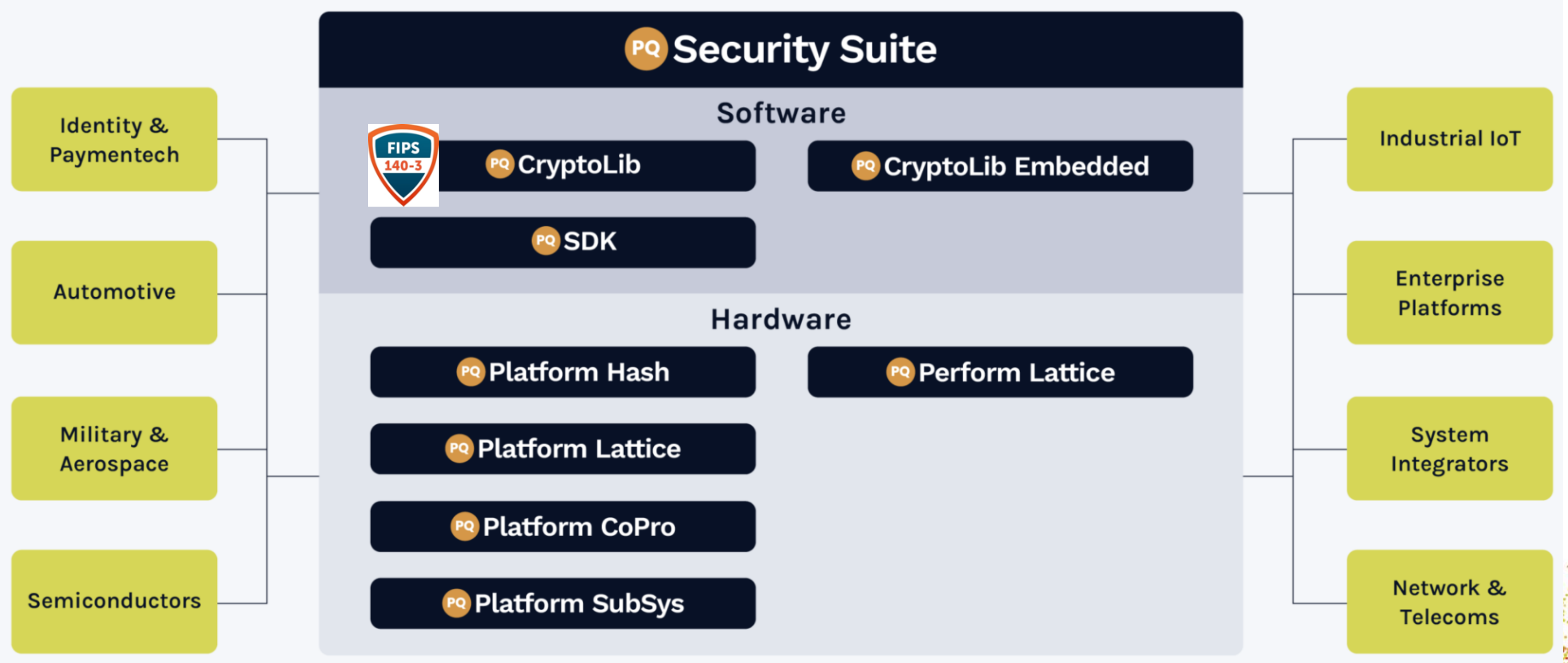
```
LAMPS                                                      M. Ounsworth
Internet-Draft                                                  J. Gray
Intended status: Standards Track                                Entrust
Expires: 9 January 2025                                         M. Pala
                                                         OpenCA Labs
                                                         J. Klaussner
                                                Bundesdruckerei GmbH
                                                           S. Fluhrer
                                                        Cisco Systems
                                                          8 July 2024

             Composite ML-DSA for use in Internet PKI
               draft-ietf-lamps-pq-composite-sigs-02
```

Thank you for your time

Questions?



PQShield: PQ Security Suite



PQ Security Suite

Software

- FIPS 140-3
- PQ CryptoLib
- PQ CryptoLib Embedded
- PQ SDK

Hardware

- PQ Platform Hash
- PQ Perform Lattice
- PQ Platform Lattice
- PQ Platform CoPro
- PQ Platform SubSys

- Identity & Paymentech
- Automotive
- Military & Aerospace
- Semiconductors

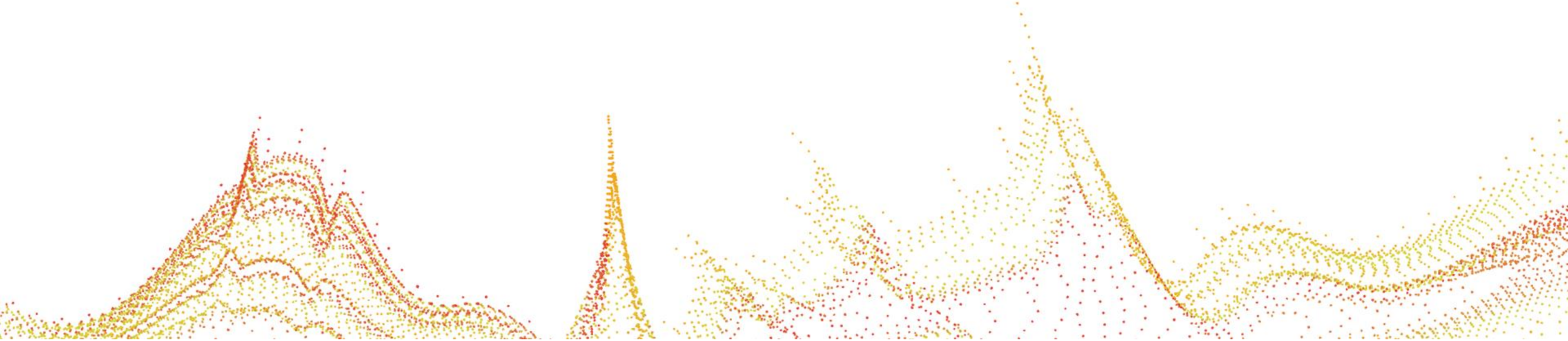
- Industrial IoT
- Enterprise Platforms
- System Integrators
- Network & Telecoms

References

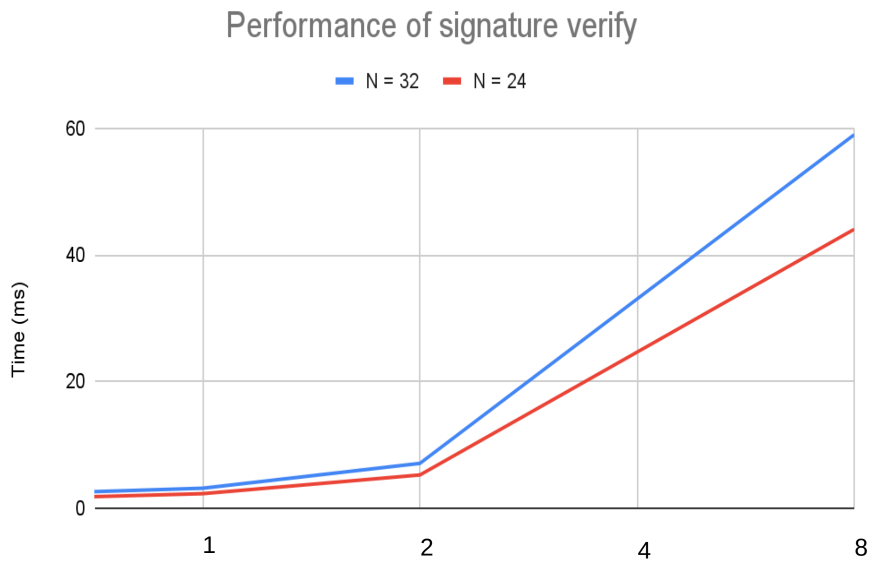
During this presentation, I’ve used some ideas previously described in the following research papers:

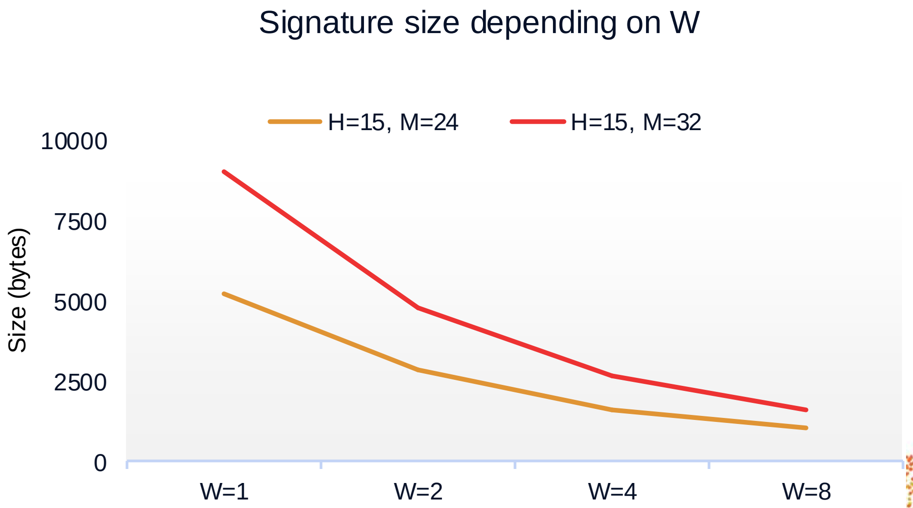
- [1] Faster AVX2 optimized NTT multiplication for Ring-LWE lattice cryptography
- [2] Compact Dilithium Implementations on Cortex-M3 and Cortex-M4
- [3] Neon NTT: Faster Dilithium, Kyber, and Saber on Cortex-A72 and Apple M1
- [4] Dilithium for Memory Constrained Devices
- [5] Hybrid scalar/vector implementations of Keccak and SPHINCS+ on AArch64
- [6] When to Barrett reduce in the inverse NTT
- [7] Improved Plantard Arithmetic for Lattice-based Cryptography
- [8] https://github.com/Emill/X25519-Cortex-M4
- [9] https://link.springer.com/chapter/10.1007/978-3-030-25283-0_6

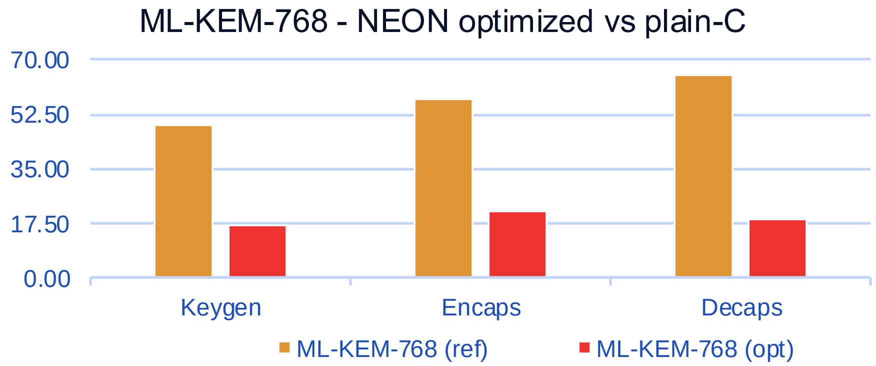
Backup

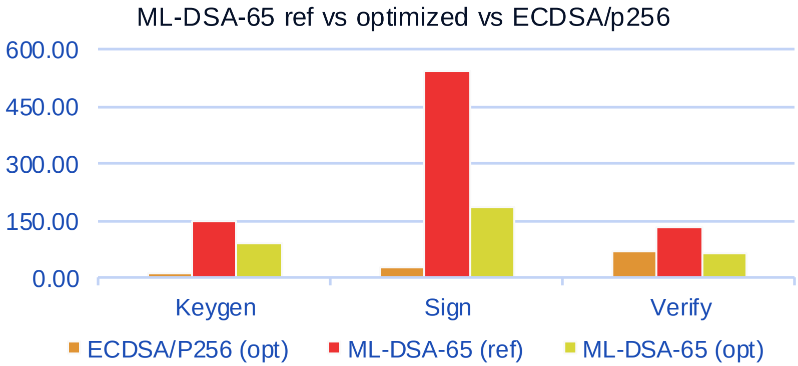


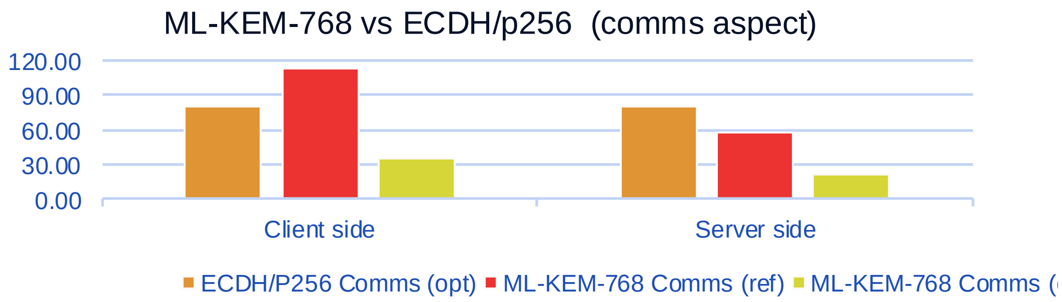
LMS parametrisation











ECDSA/ECDH from BoringSSL (optimized for NEON)
Method: https://eprint.iacr.org/2013/816