NTNU | Norwegian University of Science and Technology

INTRODUCTION TO THE NEW POST-QUANTUM STANDARDS

Tjerand Silde @ NDC Security 2026

Introduction

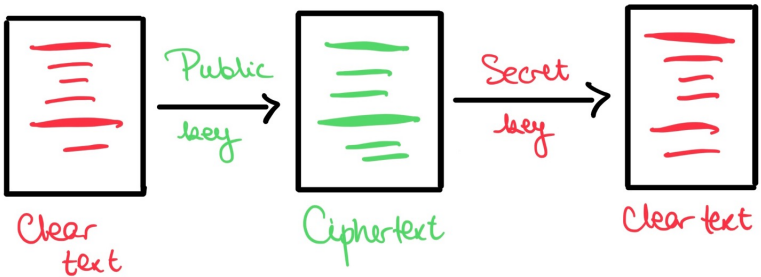
- Associate Professor in Cryptology
- Department of Information Security and Communication Technology at NTNU
- Leading NTNU Applied Cryptology Lab
- Quantum-safe cryptography, privacy applications & secure implementation
- Cryptography Expert @ Pone Biometrics

Motivation

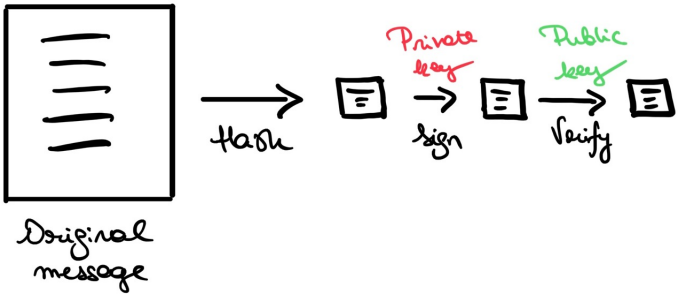
- The public key cryptographic primitives we use today are based on the hardness of factoring and discrete logarithm assumptions
- These assumptions can (in theory) be broken by Shor's algorithm on quantum computers, and quantum computing has seen recent advances
- Symmetric key encryption and hash functions seem to be fine; in the worst case, we need to double keys and outputs due to Grover's algorithm
- We need to standardize new Key Encapsulation Mechanisms (KEM) and Digital Signatures (DS) from quantum-safe assumptions, e.g., lattices
- There are also other KEM and DS standards that are not covered in this lecture, and there is an ongoing additional DS competition as we speak

Contents

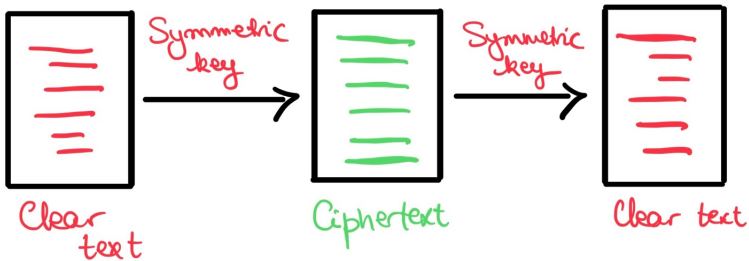
Public-key Encryption



Digital Signatures

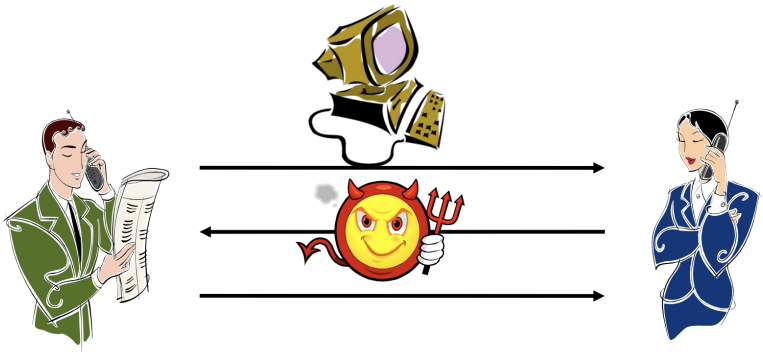


Symmetric-key Cryptography



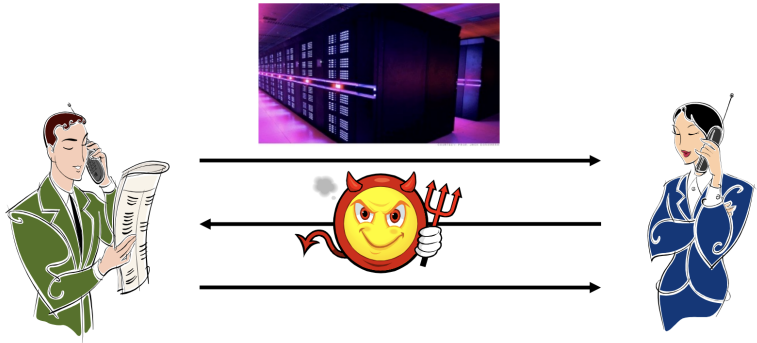
Cryptography Today

Allows for secure communication in the presence of malicious parties



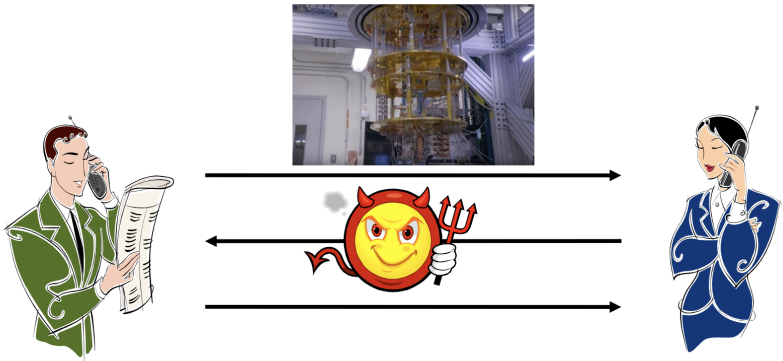
Cryptography Today

Large increase in the adversary's computing power
requires only a small increase in the key size



Cryptography Tomorrow

A quantum computer is outside the classical model of computation for efficiency purposes

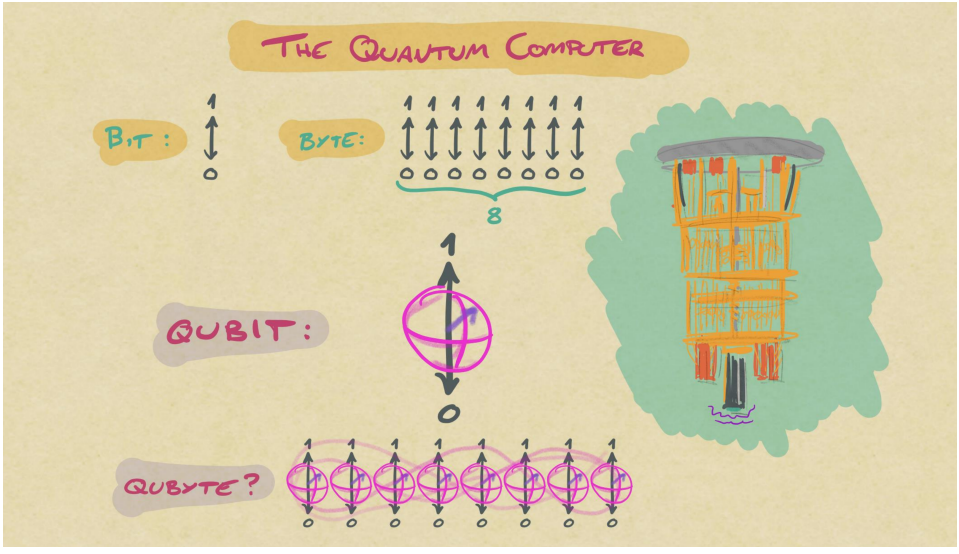


Cryptography Tomorrow

Shor's quantum algorithm can factorize integers and compute discrete logs essentially as fast as using them, given a large quantum computer. This will break RSA, (EC)DH, (EC)DSA schemes and others relying these assumptions. To achieve future secrecy, there is an urgent need to replace those algorithms.

Grover's quantum algorithm can improve exhaustive search by a square root factor, potentially speeding up key recovery and finding hash collisions.

Quantum Computers



Quantum Computers

- Quantum computers are not better; they are different
- They will generally be worse, but do specific things better
- In theory, they can break public key encryption and digital signatures based on factoring and discrete log assumptions
- There are many recent developments in quantum computing

Quantum Computers

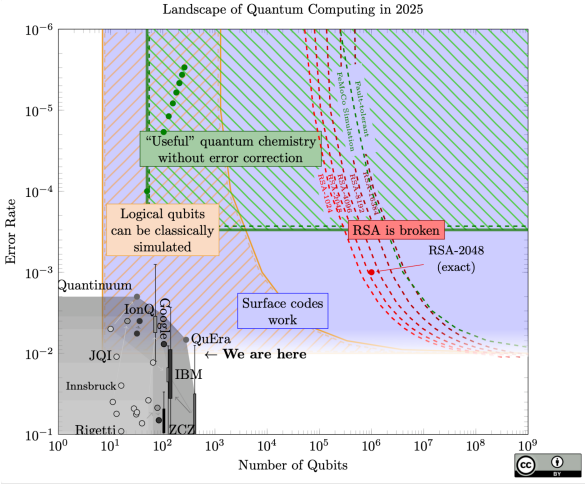


Figure: https://sam-jaques.appspot.com/quantum_landscape_2025

Factoring RSA

How to factor 2048 bit RSA integers with less than a million noisy qubits

Craig Gidney

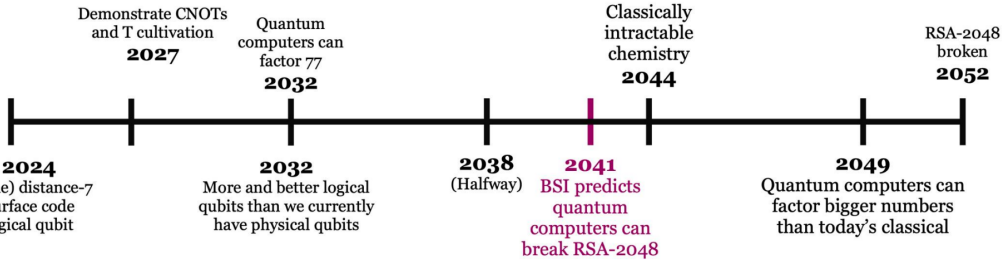
Google Quantum AI, Santa Barbara, California 93117, USA

June 9, 2025

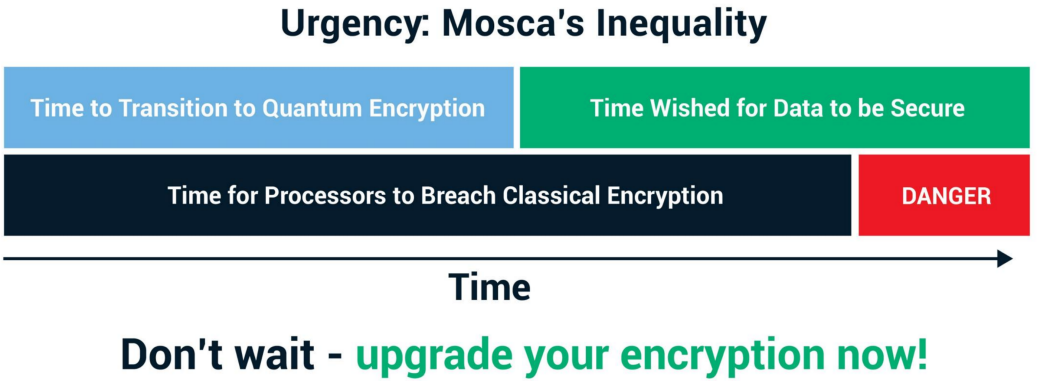
Planning the transition to quantum-safe cryptosystems requires understanding the cost of quantum attacks on vulnerable cryptosystems. In Gidney+Ekerå 2019, I co-published an estimate stating that 2048 bit RSA integers could be factored in eight hours by a quantum computer with 20 million noisy qubits. In this paper, I substantially reduce the number of qubits required. I estimate that a 2048 bit RSA integer could be factored in less than a week by a quantum computer with less than a million noisy qubits. I make the same assumptions as in 2019: a square grid of qubits with nearest neighbor connections, a uniform gate error rate of 0.1%, a surface code cycle time of 1 microsecond, and a control system reaction time of 10 microseconds.

Figure: https://arxiv.org/pdf/2505.15917

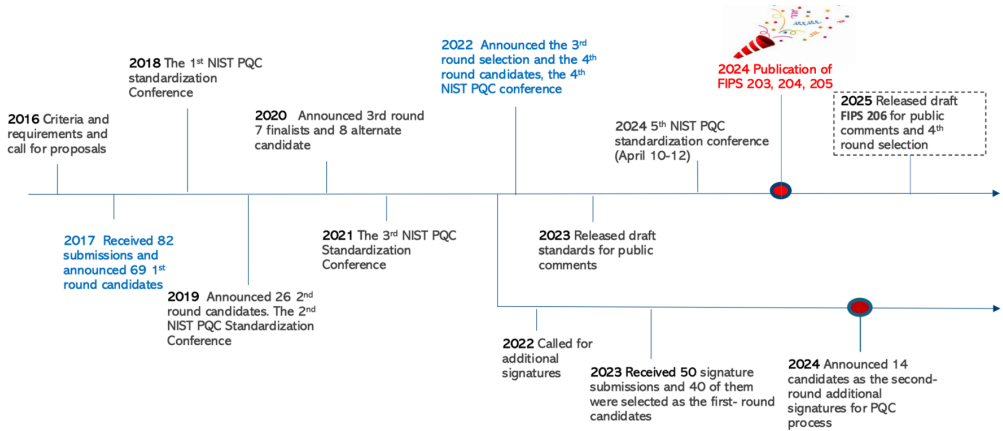
BSI Timeline



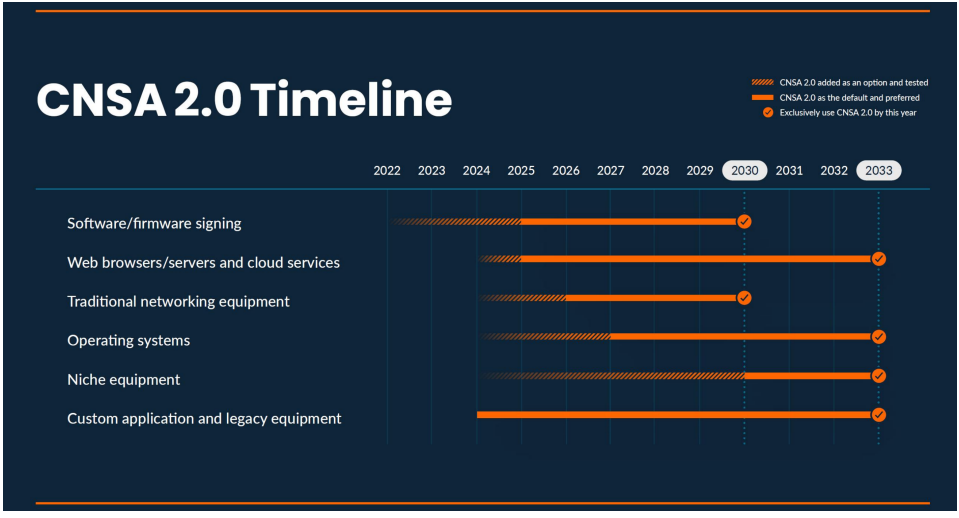
Harvest now, decrypt later



NIST Timeline



NSA Timeline



Hybrid

Quantum-resistant schemes

- Establishing a shared secret between two parties:

Scheme	*Status*
ML-KEM	D

ML-KEM must be used in hybrid mode with a quantum-vulnerable key establishment scheme using an appropriate KEM combiner. The recommended parameter sets are ML-KEM-768 and ML-KEM-1024.

Figure: https://nsm.no/regelverk-og-hjelp/veiledere-og-handboker/kryptografiske-anbefalinger-en-veileder-fra-nsm

Cloudflare

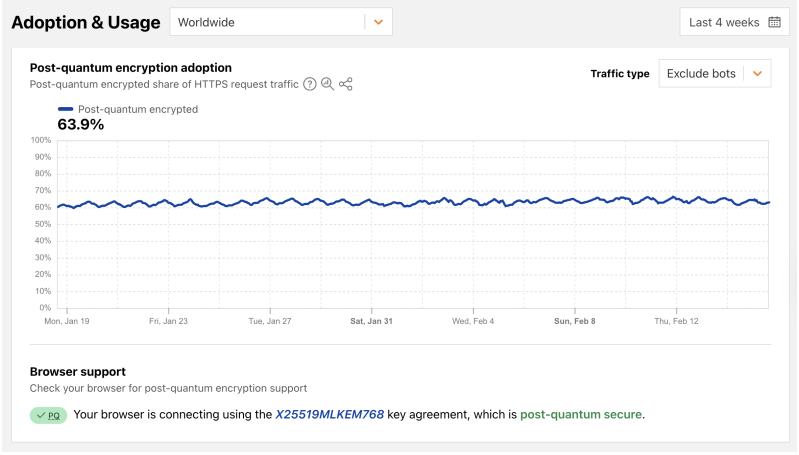


Figure: https://radar.cloudflare.com/adoption-and-usage#post-quantum-encryption-adoption

SSH

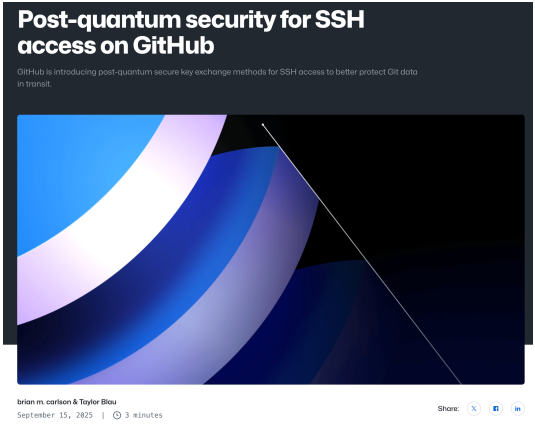


Figure: https://github.blog/engineering/platform-security/post-quantum-security-for-ssh-access-on-github/

Signal

Quantum Resistance and the Signal Protocol

ehrenkret on 19 Sep 2023

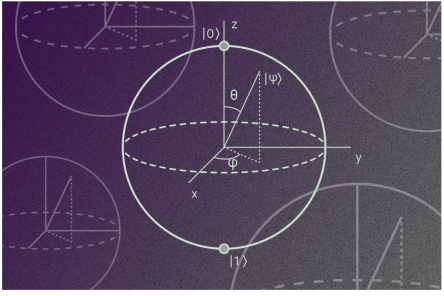


Figure: https://signal.org/blog/pqxdh

iMessage

Quantum-Secure Cryptography in Messaging Apps

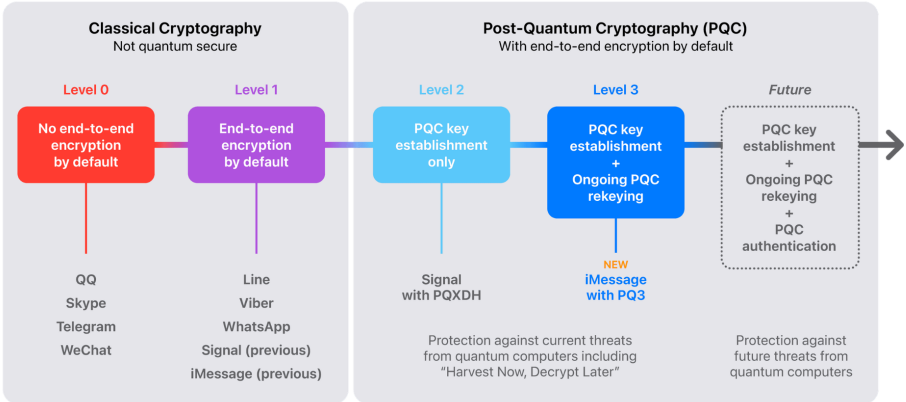


Figure: https://security.apple.com/blog/imessage-pq3

Crypto Categories

No Changes Necessary

Symmetric Cryptography:
- AES
- SHA-256 / SHA-3
- HMAC
- etc.

Done.

Almost Drop-in Replacements

NIST standardizations:
- Public Key Encryption
- Key Exchange
- Digital Signatures

A few other things:
- Identity-Based Encryption

Almost standards. Ready for deployment.

Serious Alterations of Protocols Required

Advanced Primitives:
- Zero-Knowledge Proofs
- Distributed Privacy
- Many blockchain privacy applications

Lots of recent progress on design. Near-optimality has just been achieved for certain primitives. Implementation starting at ZRL.

Can Only Be Done with Lattice Cryptography

- Fully-Homomorphic Encryption (FHE) - computation over encrypted data
- Some Obfuscation (still unclear if it can be efficient or have any useful applications)

Implementation / deployment of FHE at Haifa.

Contents

Today: Decisional Diffie-Hellman

Let $\mathbb{G}$ be a group of prime order p and g be a generator for $\mathbb{G}$.

Sample a, b, c uniformly at random from $\mathbb{Z}_p$. The Decisional Diffie-Hellman problem is to distinguish the two cases:

$$(g, g^a, g^b, g^{ab})$$

$$(g, g^a, g^b, g^c)$$

New: Learning With Errors (LWE)

Definition 1. For positive integers m, n, q, and $\beta < q$, the $\mathsf{LWE}_{n,m,q,\beta}$ problem asks to distinguish between the following two distributions:

1. $(\mathbf{A}, \mathbf{As} + \mathbf{e})$, where $\mathbf{A} \leftarrow \mathbb{Z}_q^{n \times m}, \mathbf{s} \leftarrow [\beta]^m, \mathbf{e} \leftarrow [\beta]^n$
2. $(\mathbf{A}, \mathbf{u})$, where $\mathbf{A} \leftarrow \mathbb{Z}_q^{n \times m}$ and $\mathbf{u} \leftarrow \mathbb{Z}_q^n$.

LWE Hardness

The Learning With Errors problem gets harder when...

- the dimension gets larger
- the secret values gets larger
- the modulus gets smaller

Today: Computational Diffie-Hellman

Let $\mathbb{G}$ be a group of prime order p and g be a generator for $\mathbb{G}$.

Sample a, b uniformly at random from $\mathbb{Z}_p$. The Computational Diffie-Hellman problem is, given g, g^a, and g^b, to find g^{ab} in $\mathbb{G}$.

New: Short Integer Solution (SIS)

Definition 4. For positive integers m, n, q, and $\beta < q$, the $\mathsf{SIS}_{n,m,q,\beta}$ problem asks to find, for a randomly-chosen matrix $\mathbf{A} \leftarrow \mathbb{Z}_q^{n \times m}$, vectors $\mathbf{s}_1 \in [\beta]^m$ and $\mathbf{s}_2 \in [\beta]^n$ such that $\mathbf{A}\mathbf{s}_1 + \mathbf{s}_2 = \mathbf{0} \pmod{q}$.

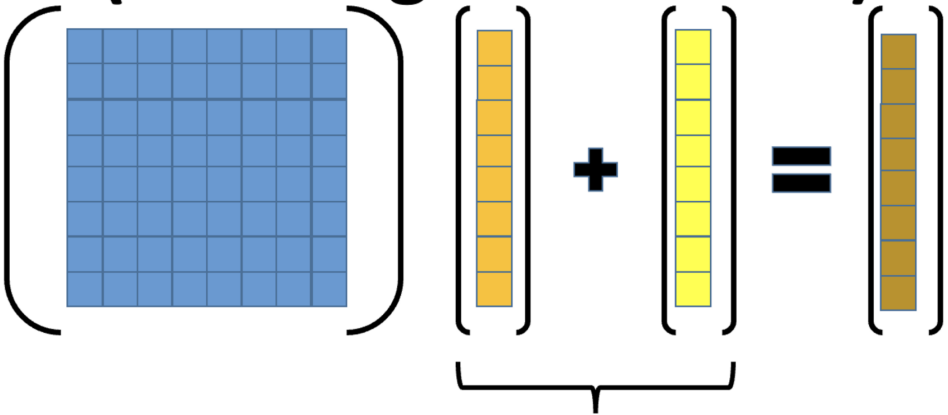
SIS Hardness

The Short Integer Solution problem gets harder when...

- the dimension gets larger
- the secret values gets smaller
- the modulus gets larger

This is opposite of LWE with respect to norms!

(Learning With Errors)



Hardness of LWE and SIS

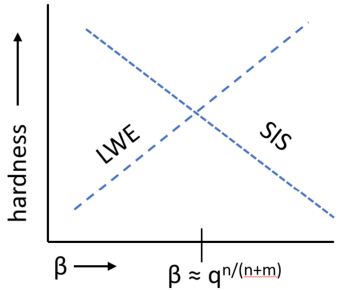


Figure 2: The hardness of $\mathsf{LWE}_{n,m,q,\beta}$ and $\mathsf{SIS}_{n,m,q,\beta}$ for fixed n, m, q, and varying β. The lines are not meant to describe the concrete hardness of these problems, but rather to illustrate the dependence of the hardness of these problems on β. The intersection point is approximately at $\beta = q^{n/(n+m)}$.

Survey

Basic Lattice Cryptography

The concepts behind Kyber (ML-KEM) and Dilithium (ML-DSA)

Vadim Lyubashevsky

IBM Research Europe, Zurich
vad@zurich.ibm.com

(Last updated: June 18, 2025)

Figure: https://eprint.iacr.org/2024/1287.pdf

Contents

FIPS 203

Federal Information Processing Standards Publication

Module-Lattice-Based Key-Encapsulation Mechanism Standard

Category: Computer Security Subcategory: Cryptography

Figure: https://nvlpubs.nist.gov/nistpubs/fips/nist.fips.203.pdf

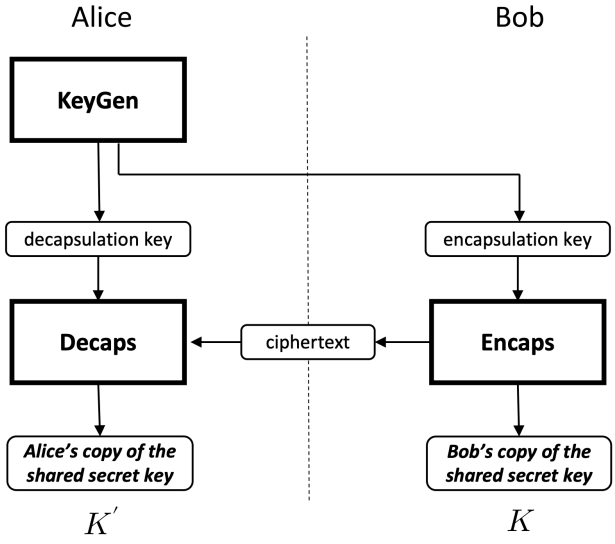
Defining a KEM

Definition 1 (Key Encapsulation Mechanism (KEM))**.** A key encapsulation mechanism is a triple of algorithms KEM $= \{\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec}\}$ with public keyspace $\mathcal{PK}$, private keyspace $\mathcal{SK}$, ciphertext space $\mathcal{C}$, and shared keyspace $\mathcal{K}$. The triple of algorithms is defined as:

- $\mathrm{KEM}.\mathsf{KeyGen}() \xrightarrow{\$} (sk, pk)$ Randomized algorithm that outputs a secret (private) key $sk \in \mathcal{SK}$, and a public key $pk \in \mathcal{PK}$.
- $\mathrm{KEM}.\mathsf{Enc}(pk) \xrightarrow{\$} (k, c)$ Randomized algorithm that, given a public key $pk \in \mathcal{PK}$, outputs a shared key $k \in \mathcal{K}$, and a ciphertext $c \in \mathcal{C}$.
- $\mathrm{KEM}.\mathsf{Dec}(c, sk) \rightarrow y \in \{k, \perp\}$ Deterministic algorithm that, given a secret key, $sk \in \mathcal{SK}$ and a ciphertext $c \in \mathcal{C}$, returns the shared key $k \in \mathcal{K}$. In case of rejection, this algorithm returns $\perp$.

Figure: https://cic.iacr.org/p/1/1/21/pdf

KEM Overview



Today: ElGamal

Let $\mathbb{G}$ be a group of prime order p and g be a generator for $\mathbb{G}$. Denote by p the public parameters $(\mathbb{G}, g, p)$.

Let the secret key $\mathsf{sk} \leftarrow\!\!\$\ \mathbb{Z}_p$ be sampled uniformly at random, and let the public key be $\mathsf{pk} = g^{\mathsf{sk}}$, where pk is made public.

The ElGamal encryption scheme, with message $m \in \mathbb{G}$, works as follows:

Enc : Sample uniform $x \leftarrow\!\!\$\ \mathbb{Z}_p$ and encrypt as $X = g^x$ and $Y = \mathsf{pk}^x \cdot m$.

Dec : Decrypt the ciphertext (X, Y) to get the message $m = Y \cdot X^{-\mathsf{sk}}$.

ML-KEM KGen and Enc

$$\mathsf{sk}: \mathbf{s} \leftarrow [\beta]^m, \ \mathsf{pk}: (\mathbf{A} \leftarrow \mathbb{Z}_q^{m \times m}, \mathbf{t} = \mathbf{A}\mathbf{s} + \mathbf{e}_1), \text{ where } \mathbf{e}_1 \leftarrow [\beta]^m. \tag{6}$$

To encrypt a message $\mu \in \{0,1\}$, the encryptor chooses $\mathbf{r}, \mathbf{e}_2 \leftarrow [\beta]^m$ and $e_3 \leftarrow [\beta]$, and outputs

$$\left(\mathbf{u}^T = \mathbf{r}^T\mathbf{A} + \mathbf{e}_2^T, v = \mathbf{r}^T\mathbf{t} + e_3 + \left\lceil \frac{q}{2} \right\rfloor \mu\right). \tag{7}$$

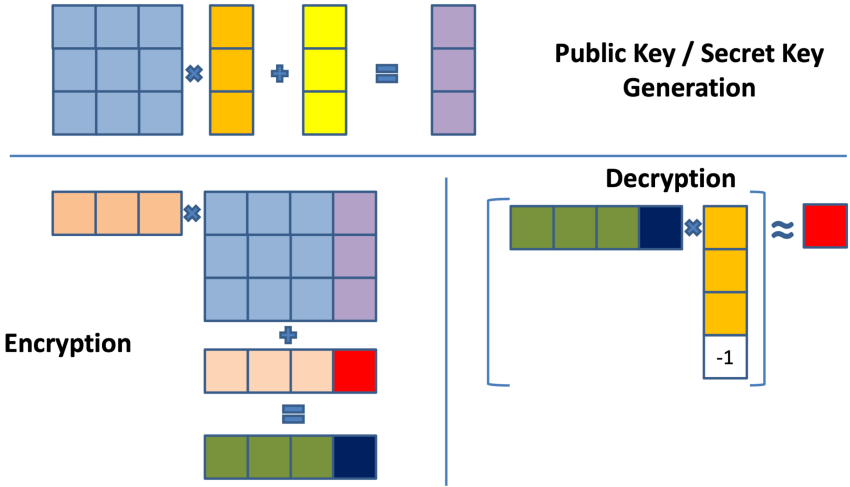
ML-KEM Dec

To decrypt, one computes $v - \mathbf{u}^{\tilde{T}}\mathbf{s}$. But rather than this cleanly giving us the message μ as in (4), we instead obtain

$$v - \mathbf{u}^T\mathbf{s} = \mathbf{r}^T(\mathbf{A}\mathbf{s} + \mathbf{e}_1) + e_3 + \frac{q}{2}\mu - \left(\mathbf{r}^T\mathbf{A} + \mathbf{e}_2^T\right)\mathbf{s} \tag{8}$$

$$= \mathbf{r}^T\mathbf{e}_1 + e_3 + \frac{q}{2}\mu - \mathbf{e}_2^T\mathbf{s} \tag{9}$$

Visualization of ML-KEM

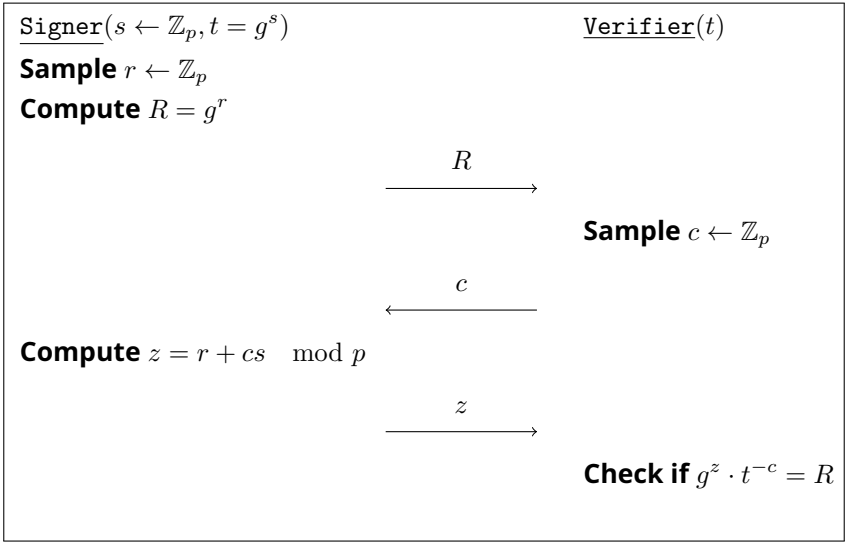


Classical vs Quantum-Safe Key-Exchange

		Size keyshares(in bytes)		Ops/sec (higher is better)	
Algorithm	PQ	**Client**	**Server**	**Client**	**Server**
Kyber512	✅	800	768	50,000	100,000
Kyber768	✅	1,184	1,088	31,000	70,000
X25519	❌	32	32	17,000	17,000

Contents

Today: Schnorr Signatures (interactive)

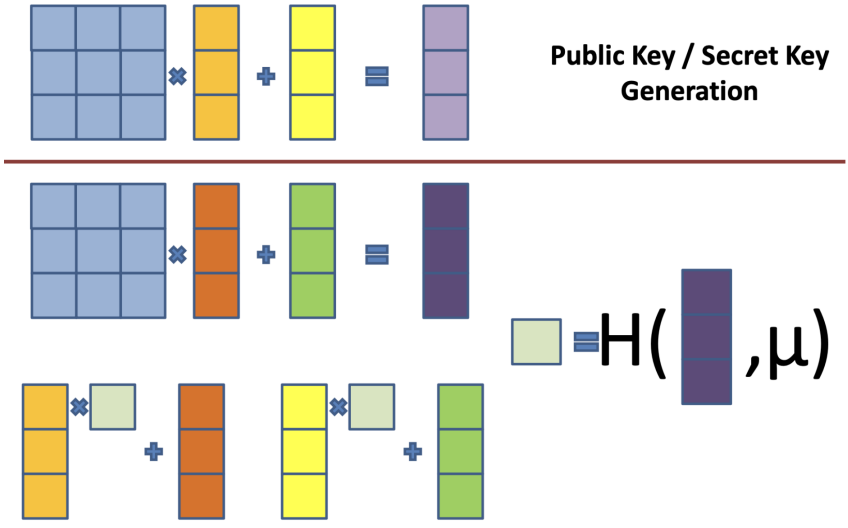


ML-DSA (interactive)

Private information: $\mathbf{s}_1 \in [\beta]^m, \mathbf{s}_2 \in [\beta]^n$
Public information: $\mathbf{A} \in \mathcal{R}_{q,f}^{n \times m}, \mathbf{t} = \mathbf{A}\mathbf{s}_1 + \mathbf{s}_2 \in \mathcal{R}_{q,f}^n$

Prover		Verifier
$\mathbf{y}_1 \leftarrow [\gamma + \bar{\beta}]^m$ $\mathbf{y}_2 \leftarrow [\gamma + \bar{\beta}]^n$, $\mathbf{w} := \mathbf{A}\mathbf{y}_1 + \mathbf{y}_2$		
	$\xrightarrow{\mathbf{w}}$	
		$c \leftarrow \mathcal{C}$
	$\xleftarrow{c}$	
$\mathbf{z}_1 := c\mathbf{s}_1 + \mathbf{y}_1$ $\mathbf{z}_2 := c\mathbf{s}_2 + \mathbf{y}_2$ if $\mathbf{z}_1 \notin [\bar{\beta}]^m$ or $\mathbf{z}_2 \notin [\bar{\beta}]^n$ then $(\mathbf{z}_1, \mathbf{z}_2) := \perp$		
	$\xrightarrow{(\mathbf{z}_1, \mathbf{z}_2)}$	
		Accept iff $\mathbf{z}_1 \in [\bar{\beta}]^m$ and $\mathbf{z}_2 \in [\bar{\beta}]^n$ and $\mathbf{A}\mathbf{z}_1 + \mathbf{z}_2 - c\mathbf{t} = \mathbf{w}$

Visualization of ML-DSA



Schnorr vs ML-DSA

- Schnorr is based on DLOG, ML-DSA on LWE and SIS
- Schnorr has uniform challenges, ML-DSA has small
- ML-DSA aborts if the norm of z is too big
- ML-DSA require norm checks of the signature

Sizes

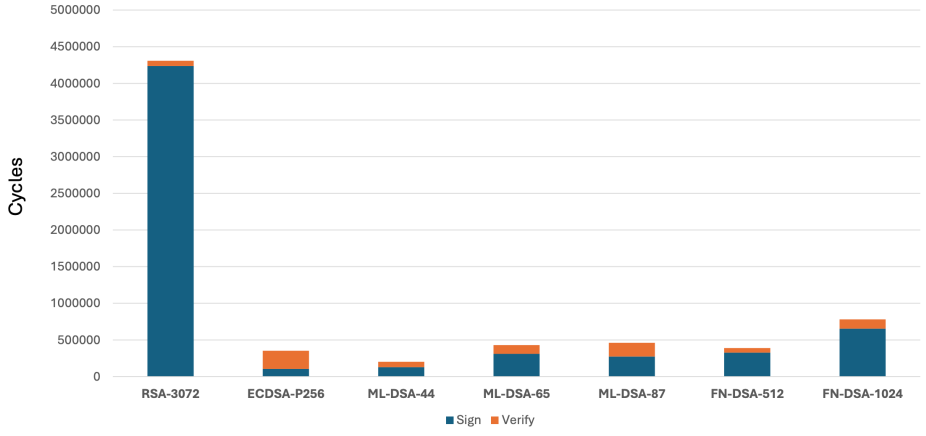
PQC Key and Signature Sizes

NIST

Scheme	Public Key (bytes)	Private Key (bytes)	Signature (bytes)	Security Level
RSA-3072	**384**	**384**	**384**	**Classical-128**
ECDSA-P256	**64**	**32**	**256**	**Classical-128**
ML-DSA-44 (Dilithium2)	**1312**	**2528**	**2420**	**PQC Category 2** *(SHA3-256)*
ML-DSA-65 (Dilithium3)	**1952**	**4000**	**3293**	**PQC Category 3** *(AES-192)*
ML-DSA-87 (Dilithium5)	**2592**	**4864**	**4595**	**PQC Category 5** *(AES-256)*
FN-DSA-512 (Falcon512)	**897**	**7553**	**666**	**PQC Category 1** *(AES-128)*
FN-DSA-1024 (Falcon1024)	**1793**	**13953**	**1280**	**PQC Category 5** *(AES-256)*

Performance

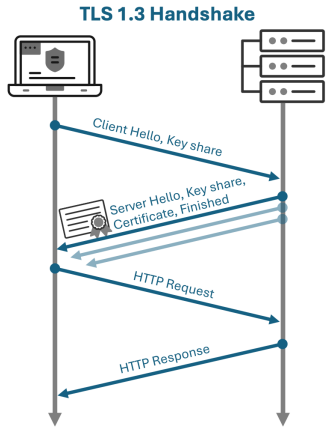
PQC Signatures– Performance



Signatures in TLS

A bit much to chew?

- TLS & WebPKI Certificate Signatures
  - *Server Certificate*: 1 public key and signature, 2 SCT signatures
  - *Intermediate CA Certificate*: 1 public key and signature
  - *TLS Handshake*: 1 signature
  - ML-DSA-44 → **14,724 bytes**
  - Current Quantum-Vulnerable → **1,248 bytes**
- ML-KEM-768 key shares
  - Client → Server: 1,184 bytes
  - Server → Client: **1,088 bytes**
- Why does this matter?
  - *TCP initial congestion window* limits the first wave of messages
  - Typical default: **~14,600 bytes**
- Without protocol/implementation changes, this could slow web connection establishment



Contents

Notation from ML-KEM

n	Denotes the integer 256 throughout this document.
q	Denotes the prime integer $3329 = 2^8 \cdot 13 + 1$ throughout this document.
ζ	Denotes the integer 17, which is a primitive n-th root of unity modulo q.
$\mathbb{B}$	The set $\{0, 1, \ldots, 255\}$ of unsigned 8-bit integers (bytes).
$\mathbb{Q}$	The set of rational numbers.
$\mathbb{Z}$	The set of integers.
$\mathbb{Z}_m$	The ring of integers modulo m (i.e., the set $\{0, 1, \ldots, m-1\}$ equipped with the operations of addition and multiplication modulo m.)
$\mathbb{Z}_m^n$	The set of n-tuples over $\mathbb{Z}_m$ equipped with $\mathbb{Z}_m$-module structure. As a data type, this is the set of length-n arrays whose entries are in $\mathbb{Z}_m$.
R_q	The ring $\mathbb{Z}_q[X]/(X^n + 1)$ consisting of polynomials of the form $f = f_0 + f_1 X + \cdots + f_{255} X^{255}$, where $f_j \in \mathbb{Z}_q$ for all j. The ring operations are addition and multiplication modulo $X^n + 1$.

ML-KEM Key Generation

Algorithm 13 K-PKE.KeyGen(d)

Uses randomness to generate an encryption key and a corresponding decryption key.

Input: randomness $d \in \mathbb{B}^{32}$.
Output: encryption key $\text{ek}_{\text{PKE}} \in \mathbb{B}^{384k+32}$.
Output: decryption key $\text{dk}_{\text{PKE}} \in \mathbb{B}^{384k}$.

$(\rho, \sigma) \leftarrow G(d \| k)$ ▷ expand 32+1 bytes to two pseudorandom 32-byte seeds[1]
$N \leftarrow 0$
for $(i \leftarrow 0; i < k; i$++$)$ ▷ generate matrix $\hat{\mathbf{A}} \in (\mathbb{Z}_q^{256})^{k \times k}$
  for $(j \leftarrow 0; j < k; j$++$)$
    $\hat{\mathbf{A}}[i, j] \leftarrow \text{SampleNTT}(\rho \| j \| i)$ ▷ j and i are bytes 33 and 34 of the input
  end for
end for
for $(i \leftarrow 0; i < k; i$++$)$ ▷ generate $\mathbf{s} \in (\mathbb{Z}_q^{256})^k$
  $\mathbf{s}[i] \leftarrow \text{SamplePolyCBD}_{\eta_1}(\text{PRF}_{\eta_1}(\sigma, N))$ ▷ $\mathbf{s}[i] \in \mathbb{Z}_q^{256}$ sampled from CBD
  $N \leftarrow N + 1$
end for
for $(i \leftarrow 0; i < k; i$++$)$ ▷ generate $\mathbf{e} \in (\mathbb{Z}_q^{256})^k$
  $\mathbf{e}[i] \leftarrow \text{SamplePolyCBD}_{\eta_1}(\text{PRF}_{\eta_1}(\sigma, N))$ ▷ $\mathbf{e}[i] \in \mathbb{Z}_q^{256}$ sampled from CBD
  $N \leftarrow N + 1$
end for
$\hat{\mathbf{s}} \leftarrow \text{NTT}(\mathbf{s})$ ▷ run NTT k times (once for each coordinate of $\mathbf{s}$)
$\hat{\mathbf{e}} \leftarrow \text{NTT}(\mathbf{e})$ ▷ run NTT k times
$\hat{\mathbf{t}} \leftarrow \hat{\mathbf{A}} \circ \hat{\mathbf{s}} + \hat{\mathbf{e}}$ ▷ noisy linear system in NTT domain
$\text{ek}_{\text{PKE}} \leftarrow \text{ByteEncode}_{12}(\hat{\mathbf{t}}) \| \rho$ ▷ run ByteEncode_{12} k times, then append $\hat{\mathbf{A}}$-seed
$\text{dk}_{\text{PKE}} \leftarrow \text{ByteEncode}_{12}(\hat{\mathbf{s}})$ ▷ run ByteEncode_{12} k times
return $(\text{ek}_{\text{PKE}}, \text{dk}_{\text{PKE}})$

ML-DSA Sign Part 1

Algorithm 7 ML-DSA.Sign_internal(sk, M', rnd)

Deterministic algorithm to generate a signature for a formatted message M'.

Input: Private key $sk \in \mathbb{B}^{32+32+64+32\cdot((\ell+k)\cdot\text{bitlen }(2\eta)+dk)}$, formatted message $M' \in \{0,1\}^*$, and per message randomness or dummy variable $rnd \in \mathbb{B}^{32}$.
Output: Signature $\sigma \in \mathbb{B}^{\lambda/4+\ell\cdot 32\cdot(1+\text{bitlen }(\gamma_1-1))+\omega+k}$.

$(\rho, K, tr, \mathbf{s}_1, \mathbf{s}_2, \mathbf{t}_0) \leftarrow$ skDecode(sk)
$\hat{\mathbf{s}}_1 \leftarrow$ NTT$(\mathbf{s}_1)$
$\hat{\mathbf{s}}_2 \leftarrow$ NTT$(\mathbf{s}_2)$
$\hat{\mathbf{t}}_0 \leftarrow$ NTT$(\mathbf{t}_0)$
$\hat{\mathbf{A}} \leftarrow$ ExpandA(ρ) ▷ $\mathbf{A}$ is generated and stored in NTT representation as $\hat{\mathbf{A}}$
$\mu \leftarrow$ H(BytesToBits$(tr)||M', 64)$ ▷ message representative that may optionally be computed in a different cryptographic module
$\rho'' \leftarrow$ H$(K||rnd||\mu, 64)$ ▷ compute private random seed
$\kappa \leftarrow 0$ ▷ initialize counter κ
$(\mathbf{z}, \mathbf{h}) \leftarrow \perp$
while $(\mathbf{z}, \mathbf{h}) = \perp$ **do** ▷ rejection sampling loop
    $\mathbf{y} \in R_q^\ell \leftarrow$ ExpandMask(ρ'', κ)
    $\mathbf{w} \leftarrow$ NTT$^{-1}(\hat{\mathbf{A}} \circ$ NTT$(\mathbf{y}))$
    $\mathbf{w}_1 \leftarrow$ HighBits$(\mathbf{w})$ ▷ signer's commitment
        ▷ HighBits is applied componentwise (see explanatory text in Section 7.4)
    $\tilde{c} \leftarrow$ H$(\mu||$w1Encode$(\mathbf{w}_1), \lambda/4)$ ▷ commitment hash
    $c \in R_q \leftarrow$ SampleInBall$(\tilde{c})$ ▷ verifier's challenge
    $\hat{c} \leftarrow$ NTT(c)
    $\langle\langle c\mathbf{s}_1\rangle\rangle \leftarrow$ NTT$^{-1}(\hat{c} \circ \hat{\mathbf{s}}_1)$
    $\langle\langle c\mathbf{s}_2\rangle\rangle \leftarrow$ NTT$^{-1}(\hat{c} \circ \hat{\mathbf{s}}_2)$

ML-DSA Sign Part 2

20: $\mathbf{z} \leftarrow \mathbf{y} + \langle\langle c\mathbf{s}_1 \rangle\rangle$ ▷ signer's response
21: $\mathbf{r}_0 \leftarrow \text{LowBits}(\mathbf{w} - \langle\langle c\mathbf{s}_2 \rangle\rangle)$
22: ▷ LowBits is applied componentwise (see explanatory text in Section 7.4)
23: **if** $||\mathbf{z}||_\infty \geq \gamma_1 - \beta$ **or** $||\mathbf{r}_0||_\infty \geq \gamma_2 - \beta$ **then** $(\mathbf{z}, \mathbf{h}) \leftarrow \perp$ ▷ validity checks
24: **else**
25: $\langle\langle c\mathbf{t}_0 \rangle\rangle \leftarrow \text{NTT}^{-1}(\hat{c} \circ \hat{\mathbf{t}}_0)$
26: $\mathbf{h} \leftarrow \text{MakeHint}(-\langle\langle c\mathbf{t}_0 \rangle\rangle, \mathbf{w} - \langle\langle c\mathbf{s}_2 \rangle\rangle + \langle\langle c\mathbf{t}_0 \rangle\rangle)$ ▷ Signer's hint
27: ▷ MakeHint is applied componentwise (see explanatory text in Section 7.4)
28: **if** $||\langle\langle c\mathbf{t}_0 \rangle\rangle||_\infty \geq \gamma_2$ **or** the number of 1's in $\mathbf{h}$ is greater than ω, **then** $(\mathbf{z}, \mathbf{h}) \leftarrow \perp$
29: **end if**
30: **end if**
31: $\kappa \leftarrow \kappa + \ell$ ▷ increment counter
32: **end while**
33: $\sigma \leftarrow \text{sigEncode}(\tilde{c}, \mathbf{z} \bmod^{\pm} q, \mathbf{h})$
34: **return** σ

Implementations

- Reference implementations in C/AVX2 at `https://pq-crystals.org`
- Eudational implementations in Python:
  - `https://github.com/GiacomoPope/kyber-py`
  - `https://github.com/GiacomoPope/dilithium-py`
- Open Quantum Safe project: `https://openquantumsafe.org`

Questions?