



CLLOUDFLARE®

Dawn of the Post-Quantum Internet

Dr Bas Westerbaan, Cloudflare Research
Connections 2024, Feb 8th, 2024

About Cloudflare

We run a **global network** spanning 300 cities in over 100 countries.

Started of as a **CDN** and **DDoS mitigation** company, we now offer many more services, including

- **1.1.1.1**, public DNS resolver
- **Workers**, serverless compute
- **SASE**, to protect corporate networks

We serve nearly **20% of all websites** and process 46 million HTTP requests per second.



Building a better Internet

Cloudflare cares deeply about a **private**, **secure** and **fast** Internet, helping design, and adopt, among others:

- Free SSL (2014), TLS 1.3 and QUIC
- DNS-over-HTTPS
- Private Relay / OHTTP
- Encrypted ClientHello

And, the topic today:

- Migrating to post-quantum cryptography.



This talk

Overview of the current state of migration of the **Internet / WebPKI**, and its unique challenges.

Changing the Internet / WebPKI is hard

- **Very diverse.** Many different users / stakeholders with varying (performance) constraints and update cycles.

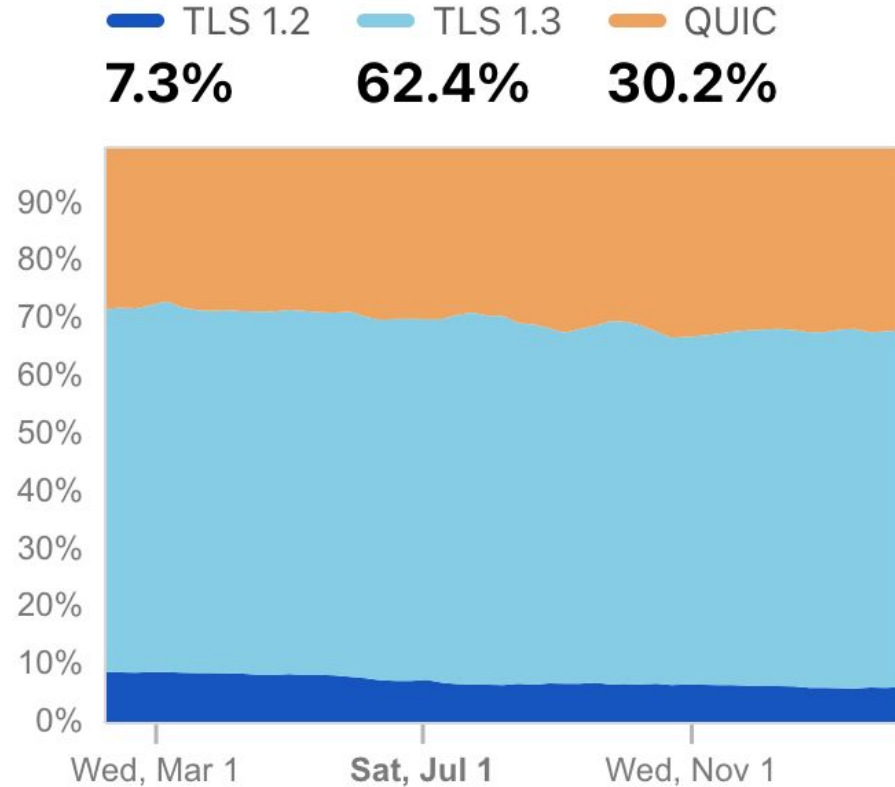
We can't assume everyone is on fiber, or uses modern CPU, can store state, or can update at all.

- **Protocol ossification.** Despite being designed to be upgradeable, any flexibility that isn't used in practice, is probably broken, because of faulty implementations.

TLS 1.3 migration

Early versions of TLS 1.3 were completely undeployable because of protocol ossification

After **six more years** of testing and adding workarounds, the final version of TLS 1.3 is a success, used by over 90% of our visitors.



[Cloudflare Radar](#)

There will be *two* post-quantum migrations.

1. Key agreement

Communication can be recorded today and decrypted in the future. We need to upgrade **as soon as possible**.

2. Signatures

Less urgent: need to be replaced **before** the arrival of cryptographically-relevant quantum computers.

Key agreement 🤝

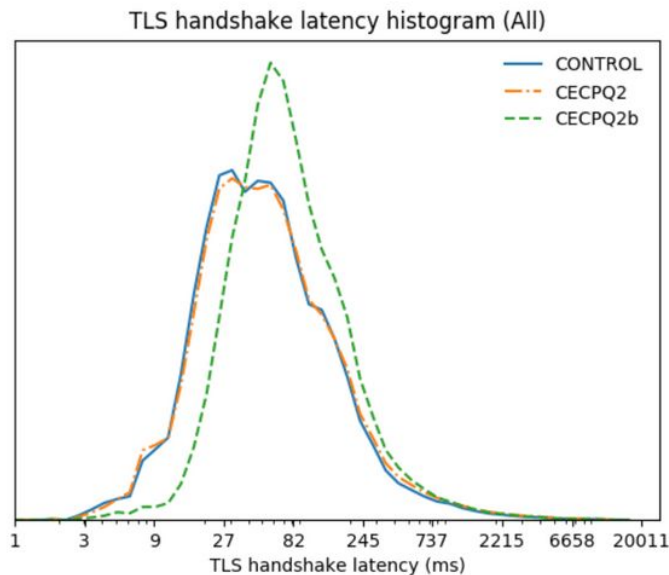
Urgent, and the *easier* one.

Feasibility study with Chrome

In 2019 we performed large-scale test of PQ kex with Chrome. Takeaways:

- Performance of lattice-based KEMs is acceptable.
- Significant amount of broken clients because of protocol ossification (*split ClientHello*.)

Google has been working with vendors to fix issues.



X25519. CECPQ2 is X25519+NTRU-HRSS (lattice) and CECPQ2b is X25519+SIKE (isogenies, broken)

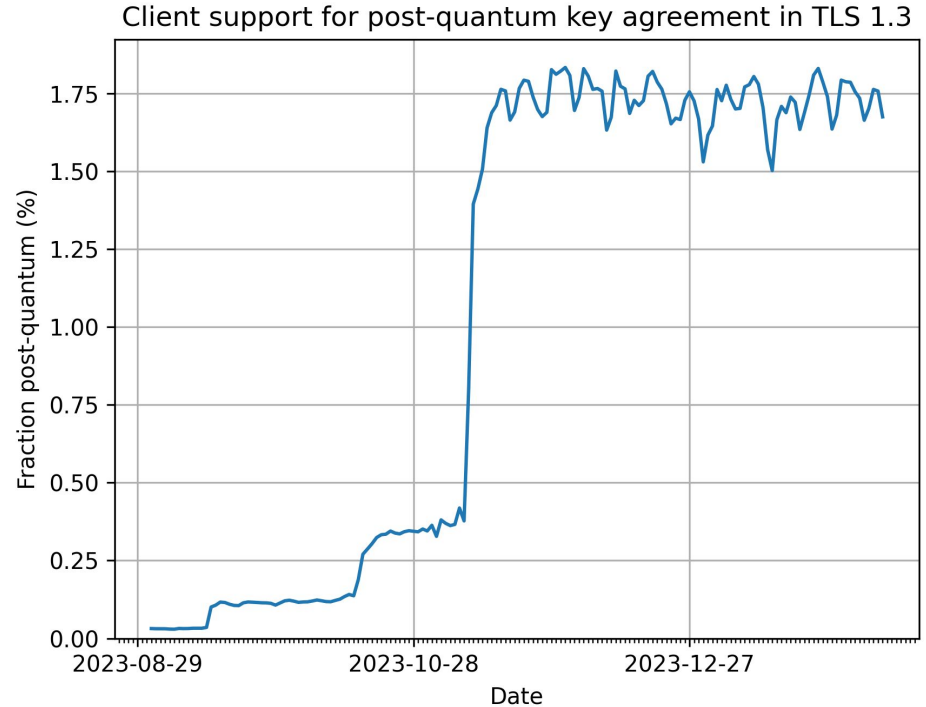
Early deployments

2022 coordinating at IETF, we enabled post-quantum key agreement (~20% Internet.)

In 2023 Google enabled server-side as well.

Browsers:

- Chrome. Enabled for 10% of all traffic.
- Firefox. Opt-in in nightly.



Promising early results

As of writing, no hard failures preventing further roll-out identified by Chrome 🙌. Our own testing has shown that there are about a hundred customers with incompatible origin servers. We're reaching out to help fix them.

It is likely that we will see **double-digit percentage** post-quantum key-agreement later this year.

Key agreement

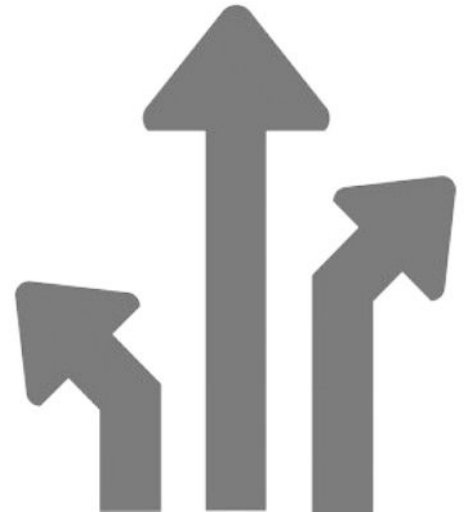
Urgent and the **easier** of the two to deploy. We're on track for ~30% client-side deployment in 2024. That took 5 years.

Signatures

Less urgent, but **much more challenging.**

#1, many more parties involved:

Cryptography library developers, browsers, certificate authorities, HSM manufacturers, CT logs, and every server admin that cobbled together a PKI script.



#2, there is **no all-round great** PQ signature

	PQ	Size (bytes)		CPU time (lower is better)	
		Public key	Signature	Signing	Verification
Ed25519	✗	32	64	1 (baseline)	1 (baseline)
RSA-2048	✗	256	256	70	0.3
Dilithium2	✓	1,312	2,420	4.8	0.5
Falcon512	✓	897	666	8*	0.5
SPHINCS ⁺ 128s	✓	32	7,856	8,000	2.8
SPHINCS ⁺ 128f	✓	32	17,088	550	7

Intermezzo: naming confusion

Original name	NIST's name	FIPS number
Dilithium	ML-DSA	204
Falcon	FN-DSA	?
SPHINCS ⁺	SLH-DSA	205
Kyber	ML-KEM	203

For the moment I will use the original names.

Online signing — Falcon's Achilles' heel

- For fast signing, Falcon requires a **floating-point unit** (FPU).
- We do not have enough experience running cryptography securely (**constant-time**) on the FPU.
- On commodity hardware, **Falcon should not be used when signature creation can be timed**, eg. TLS handshake.
- Not a problem for signature verification.



#3, there are **many** signatures on the Web

- Root on intermediate
- Intermediate on leaf
- Leaf on handshake
- Two SCTs for Certificate Transparency
- An OCSP staple

Typically **6 signatures**
and **2 public keys**
when visiting a **website**.

(And we're not even counting DNSSEC.)



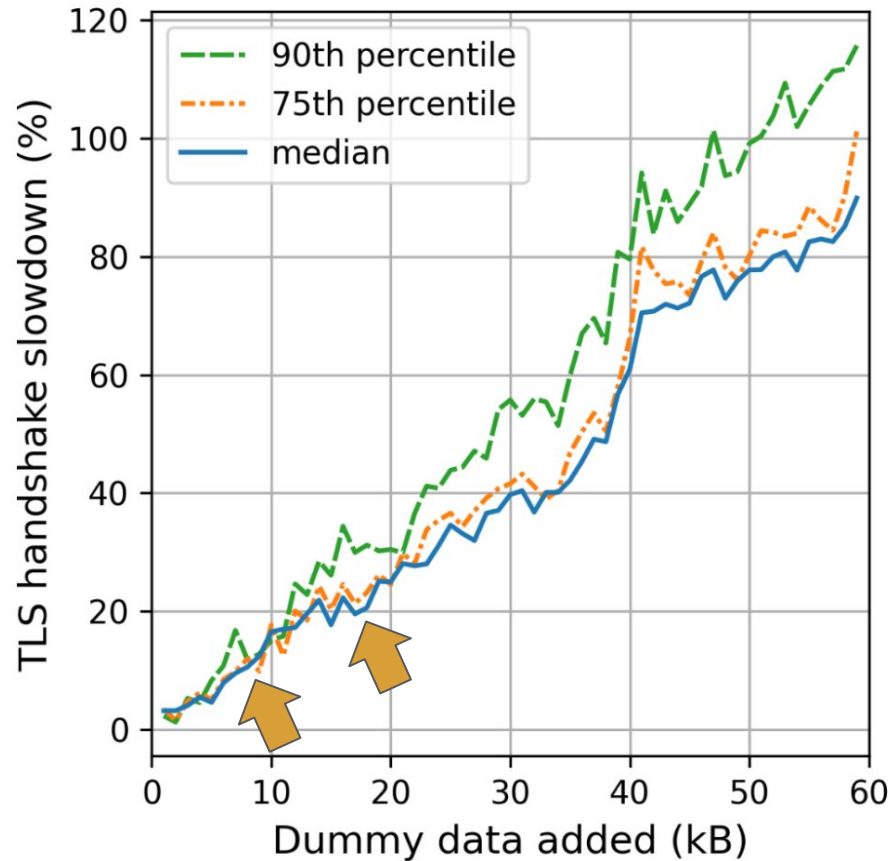
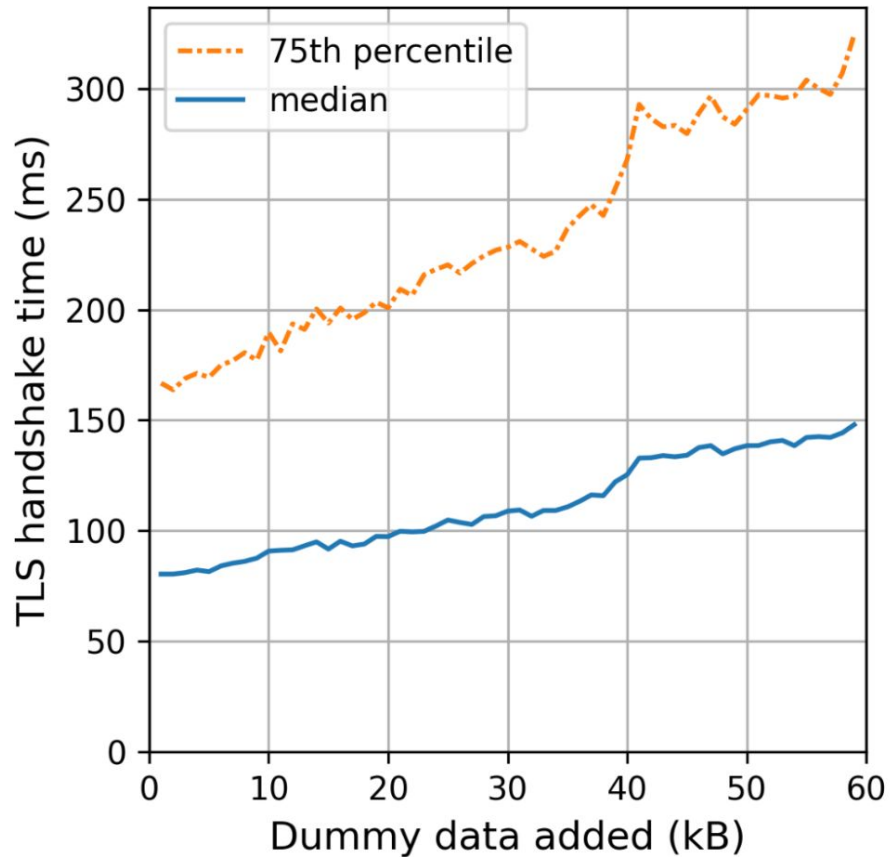
Using only Dilithium2

+17,144 bytes

Using Dilithium2 for the TLS handshake and Falcon for the rest

+7,959 bytes

Is that too much? We had a look...



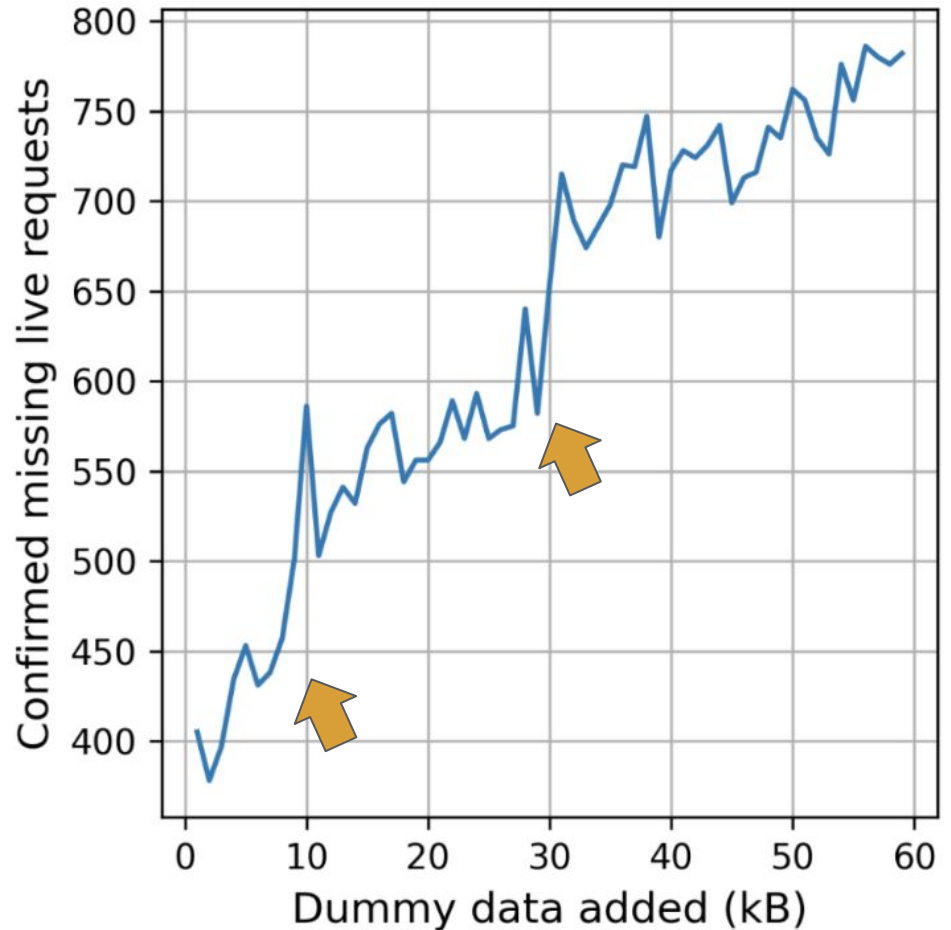
And, of course...

Protocol ossification

Bump in missing requests suggests some clients or middleboxes do not like certificate chains longer than **10kB** and 30kB.

This is problematic for composite certificates.

Instead configure servers for **multiple separate certificates** and let TLS negotiate the one to send.



Not great, not terrible

It probably won't break the Web, but the performance impact will **delay adoption**.

NIST signature on-ramp

NIST took notice and has called for new signature schemes to be submitted.

See backup slides at the end.

The short of it: there are some very promising submissions, but their security is as of yet unclear.

Thus, we cannot assume that a new post-quantum signature will solve our issues.



In the meantime

There are small and larger changes possible to the protocols to **reduce the number of signatures**.



- Leave out intermediate certificates.
- Use key agreement for authentication.
- Overhaul WebPKI, eg. Merkle Tree Certificates.

These are discussed briefly in the backup slides at the end.

Signatures



Less urgent, but path is unclear. Real risk we will start migrating too late.

That's not all: the Internet isn't just TLS

There is much more cryptography out there with their own unique challenges.

- DNSSEC with its harder size constraints
- Research into post-quantum privacy enhancing techniques, eg. anonymous credentials, is in the early stages.

Thank you, questions?

References

- Follow along at the [IETF](#)
- Check out our blog, eg.:
 - [2019 TLS experiment](#) with Google
 - [Sizing-up Post-Quantum Signatures](#)
 - [Deploying Kyber worldwide](#)
- Reach out: ask-research@cloudflare.com

Backup slides

Not all signatures are equal

The TLS handshake signature is created on-the-fly (**online**) and is transmitted together with its public key.

The handshake signature benefits from balanced signing/verification time, and balanced public key/signature size.

The other signatures are **offline**, and can trade signing time for better verification time. The intermediate's signatures are sent with their corresponding public key, and the rest (SCT/OCSP staple) **without public key**.


The former benefits from balanced signature/public key size. For the latter it's beneficial to trade public key and signature sizes.

			Sizes (bytes)		CPU time (lower is better)	
		PQ	Public key	Signature	Signing	Verification
Standardised	Ed25519	✗	32	64	1 (baseline)	1 (baseline)
	RSA-2048	✗	256	256	70	0.3
Hash-based	XMSS* w=256 h=20 n=16	✓	32	608	6 ⚠	2
NIST drafts	Dilithium2	✓	1,312	2,420	4.8	0.5
	Falcon512	✓	897	666	8 ⚠	0.5
	SPHINCS ⁺ 128s	✓	32	7,856	8,000	2.8
	SPHINCS ⁺ 128f	✓	32	17,088	550	7
Sample from signatures on-ramp	MAYO_one	✓	1,168	321	11	1.3
	MAYO_two	✓	5,488	180	13	0.7
	SQISign I	✓	64	177	60,000	500
	UOV Is-pkc	✓	66,576	96	2.5	2
	HAWK512	✓	1,024	555	2	1

Concrete instances with NIST drafts

Using **Dilithium2** for everything adds 17kB.

Using **Dilithium2** for handshake and **Falcon512** for the rest, adds 8kB.

 Fast and secure Falcon512 signing is hard to implement.

Using **SPHINCS⁺-128** for everything adds 50kB. Order of magnitude worse signing time than RSA. Most conservative choice.

Stateful hash-based signatures

Using **XMSS^(MT)** with $w=256$, $n=128$, two subtrees for SCTs and intermediates, and single tree for the rest, and **Dilithium2** for handshake signature, adds 8kB.

- ⚠ $n=128$ and $w=256$ instances are not standardised.
- ⚠ We lose non-repudiation.
- ⚠ Large precomputations/storage required for efficient signing.
- ⚠ Challenging to keep state.

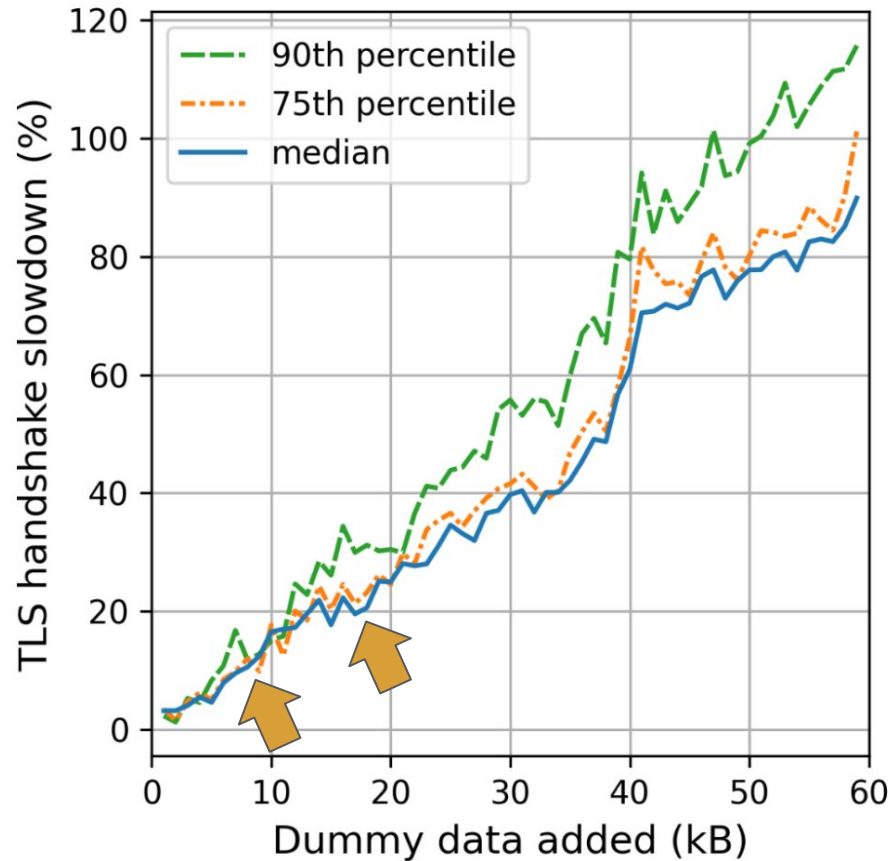
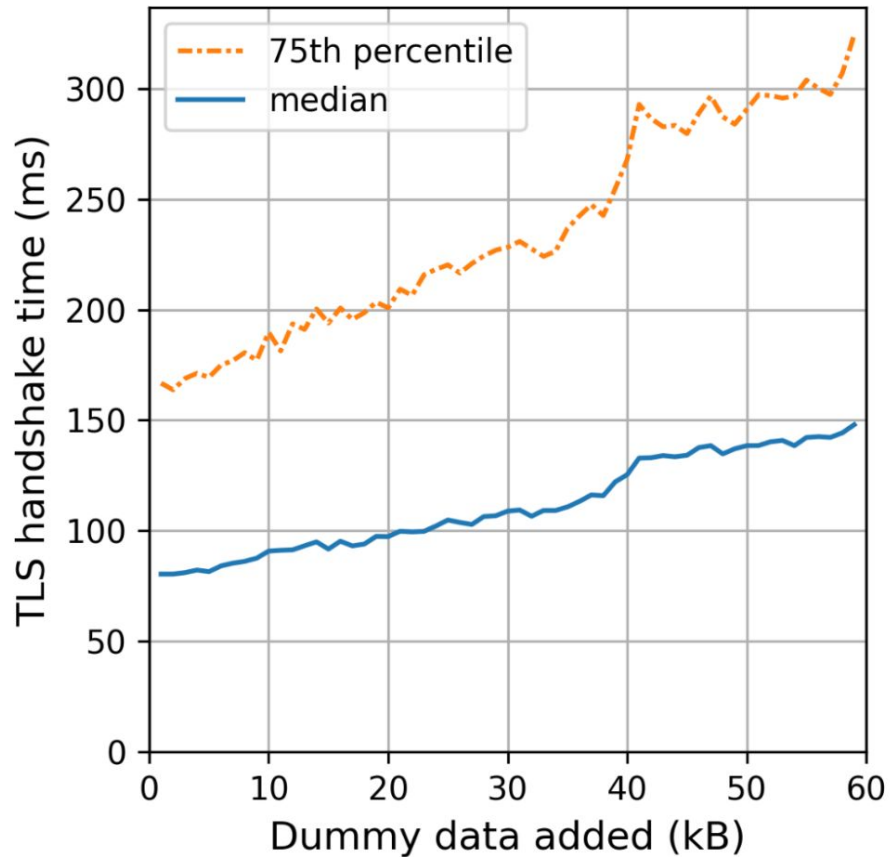
Concrete instances with on-ramp candidates

Using **MAYO** *one* for leaf/intermediate, and *two* for the rest, adds 3.3kB. Signing time between ECC/RSA. ⚠ Security uncertain.

Using **UOV** Is-pkc for root and SCTs, and **HAWK512** for the rest, adds 3.2kB. 66kB for stored UOV public keys. HAWK relies on Falcon assumptions and then some more.

Using **UOV** Is-pkc again, but combined with **Dilithium2**. Adds 7.4kB. Relatively conservative choice.

SQIsign only. Adds 0.5kB. Signing time >1s (not constant-time), and verification time >35ms. 🐢



Leaving out intermediates

Most browsers ship intermediates, so why bother sending them?

Leaving out intermediates

Three proposals:

- 2019, [draft-kampanakis-tls-scas](#), send flag to indicate server should only return leaf. Simple but error prone.
- 2022, [draft-ietf-tls-cert-abridge](#), replaces intermediates with identifiers from yearly updated central list from CCADB. Client sends version of latest list. Also proposes tailored compression.
- 2023, [draft-davidben-tls-trust-expr](#). Simplified: client sends which trust store it uses, and the version it has. CA adds as metadata to a certificate, in which trust store (version) it's included. Trust stores can then add intermediates as roots.

Gains leaving out intermediates: median 3kB

Scheme	Storage Footprint	p5	p50	p95
Original	0	2308	4032	5609
TLS Cert Compression	0	1619	3243	3821
Intermediate Suppression and TLS Cert Compression	0	1020	1445	3303
This Draft	65336	661	1060	1437
This Draft with opaque trained dictionary	3000	562	931	1454
Hypothetical Optimal Compression	0	377	742	1075

From Dennis Jackson's [draft-ietf-tls-cert-abridge-00](#)

KEMTLS (aka. Authkem)

Use KEM instead of signature for handshake authentication.

KEMTLS

Replacing Dilithium2 handshake signature with Kyber512 saves 2.9kB server → client, but adds 768B in the second flight client → server.

At the moment gains are modest. Interesting for embedded, to reduce code size by eliminating primitive. Client authentication with KEM requires extra roundtrip.

Large change to TLS. Subtle changes in security guarantees. We have a [formal analysis](#).

Proof-of-possession unclear. Could be done with lattice-based zero-knowledge proofs or challenge-response.

Merkle Tree Certificates

Pain-points of current WebPKI

OCSP is expensive to run, whereas majority of users don't use it, but rely on CRL instead (via eg. CRLite).

Too many signatures.

Certificate Transparency is difficult to run.

Many sharp edges: path building, punycode, constraint validation, etc.

(Domain control validation is imperfect — not addressed.)

Changing the WebPKI

With the post-quantum migration, the marginal cost of changing the WebPKI is lower than ever.

There is a huge design space, with many trade offs.

[Merkle Tree Certificates](#) (MTC) is a concrete, ambitious, but early draft. We're looking for feedback on the design and general direction.

Not a complete replacement for current WebPKI: it's an **optimisation of the common case** and falls back to X.509+CT.

Merkle Tree Certificates in short (1)

On a set time, eg. every hour, the CA publishes:

- The **batch** of **assertions** they certify. All assertions in a batch are implicitly valid for the same **window**, eg. 14 days. For each batch, the CA builds a Merkle tree on top.
- A **signature** on the roots of all currently valid batches.

Trust Services (eg. browser vendors) regularly pull the latest batches and window signatures from CAs, verify them for consistency, and only send the Merkle tree roots to the browsers.

Merkle Tree Certificates in short (2)

A **Merkle tree certificate** is an assertion together with a **Merkle authentication path** to the root of the batch.

A server would install three certificates: two Merkle tree certificates 7 days apart, and a fall back X509 certificate.

When connecting to a server, the client sends the sequence number of the latest batches it knows of each MTC CA.

If the client is sufficiently up-to-date, the server can return one of the Merkle tree certs, and otherwise will fall back to X.509.

Merkle Tree Certificates sizes

There are currently 6 billion unexpired certificates in CT.

If reissued every 7 days by one MTC CA, we'd have batches of 35 million assertions.

That amounts to authentication paths of **832 bytes**, and with a Dilithium2 public key a typical Merkle tree certificate will be **well below 2.5kB**, smaller than only the median compressed classical intermediate certificate of 3.2kB.

Wrapping up

We saw several different approaches to cope with large post-quantum signatures, from simple to ambitious.

There are still many unknowns: among others, compliance requirements; cryptanalytic breakthroughs; ecosystem ossification; stakeholder constraints; etc.

Which approach to take? I'd say it's good to have multiple pots on the stove.

```

static inline int64_t
fpr_rint(fpr x)
{
    /*
     * We do not want to use llrint() since it might be not
     * constant-time.
     *
     * Suppose that x >= 0. If x >= 2^52, then it is already an
     * integer. Otherwise, if x < 2^52, then computing x+2^52 will
     * yield a value that will be rounded to the nearest integer
     * with exactly the right rules (round-to-nearest-even).
     *
     * In order to have constant-time processing, we must do the
     * computation for both x >= 0 and x < 0 cases, and use a
     * cast to an integer to access the sign and select the proper
     * value. Such casts also allow us to find out if |x| < 2^52.
     */
    int64_t sx, tx, rp, rn, m;
    uint32_t ub;

    sx = (int64_t)(x.v - 1.0);
    tx = (int64_t)x.v;
    rp = (int64_t)(x.v + 4503599627370496.0) - 4503599627370496;
    rn = (int64_t)(x.v - 4503599627370496.0) + 4503599627370496;

    /*
     * If tx >= 2^52 or tx < -2^52, then result is tx.
     * Otherwise, if sx >= 0, then result is rp.
     * Otherwise, result is rn. We use the fact that when x is
     * close to 0 (|x| <= 0.25) then both rp and rn are correct;
     * and if x is not close to 0, then trunc(x-1.0) yields the
     * appropriate sign.
     */

    /*
     * Clamp rp to zero if tx < 0.
     * Clamp rn to zero if tx >= 0.
     */
    m = sx >> 63;
    rn &= m;
    rp &= ~m;

    /*
     * Get the 12 upper bits of tx; if they are not all zeros or
     * all ones, then tx >= 2^52 or tx < -2^52, and we clamp both
     * rp and rn to zero. Otherwise, we clamp tx to zero.
     */
    ub = (uint32_t)((uint64_t)tx >> 52);
    m = -(int64_t)((((ub + 1) & 0xFFF) - 2) >> 31);
    rp &= m;
    rn &= m;
    tx &= ~m;

    /*
     * Only one of tx, rn or rp (at most) can be non-zero at this
     * point.
     */
    return tx | rn | rp;
}

```

This function from Falcon as submitted to round 3 is not constant-time on ARMv7 as claimed.

Can you spot the error?

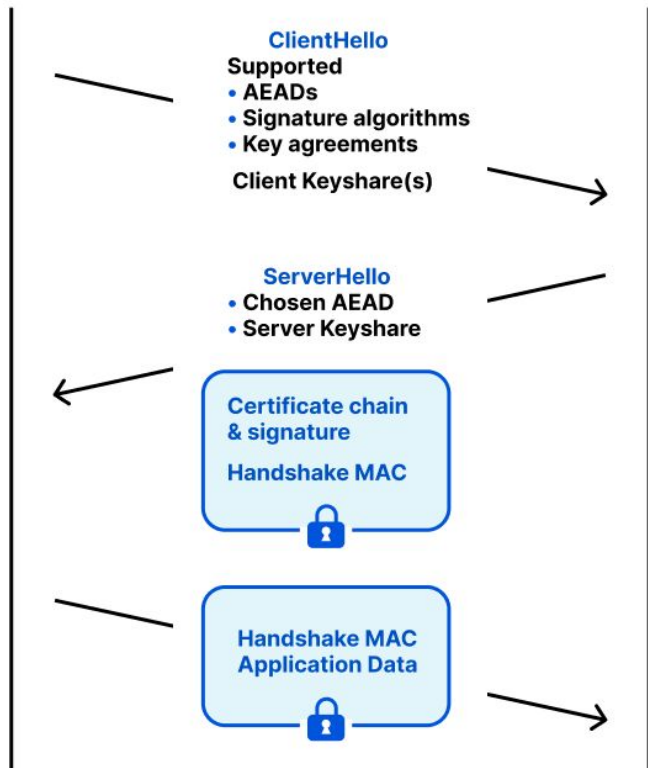
TLS 1.3 handshake



Client



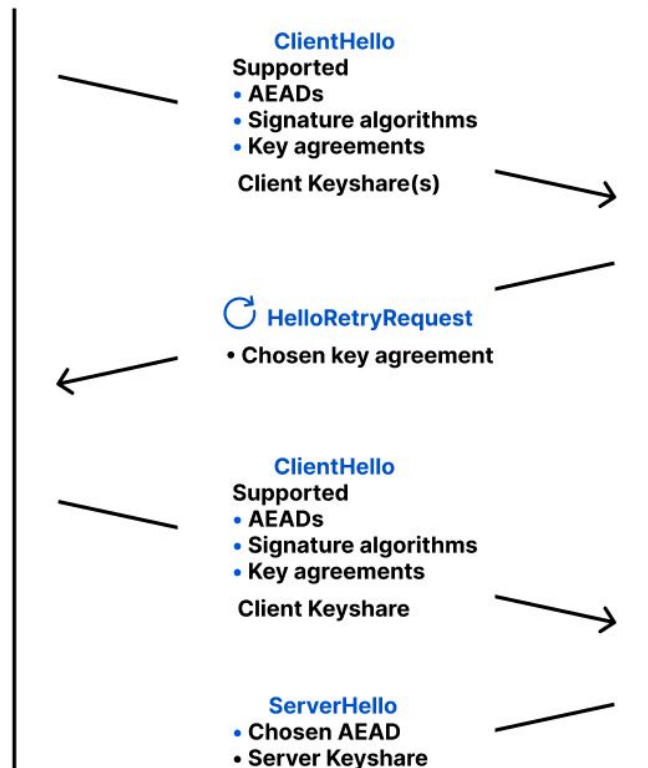
Server



Client

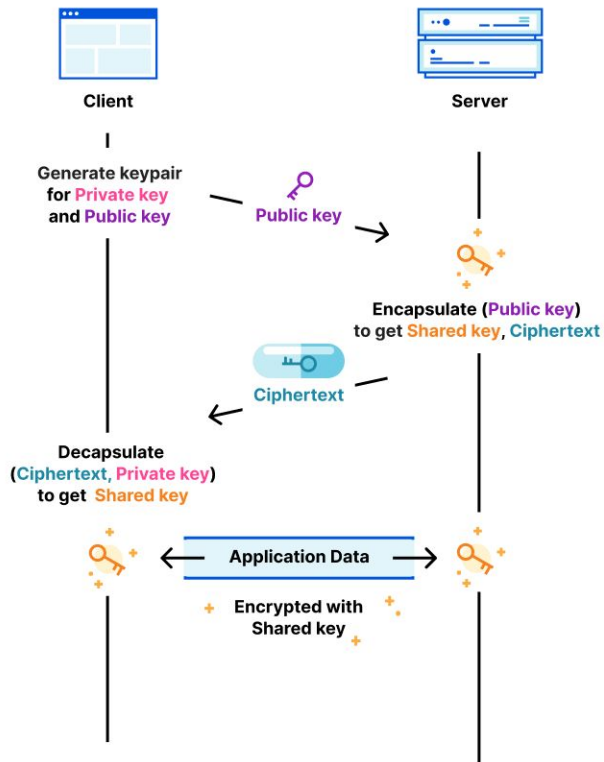


Server

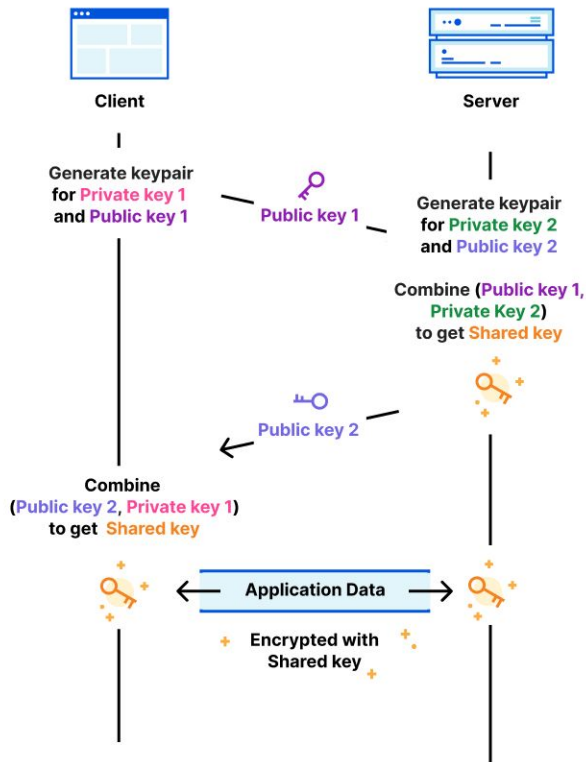


KEM versus Diffie-Hellman

Key Encapsulation Mechanism (KEM)



Diffie-Hellman (DH)



Post-Quantum Cryptography for Engineers

<https://datatracker.ietf.org/doc/draft-ietf-pquip-pqc-engineers/>

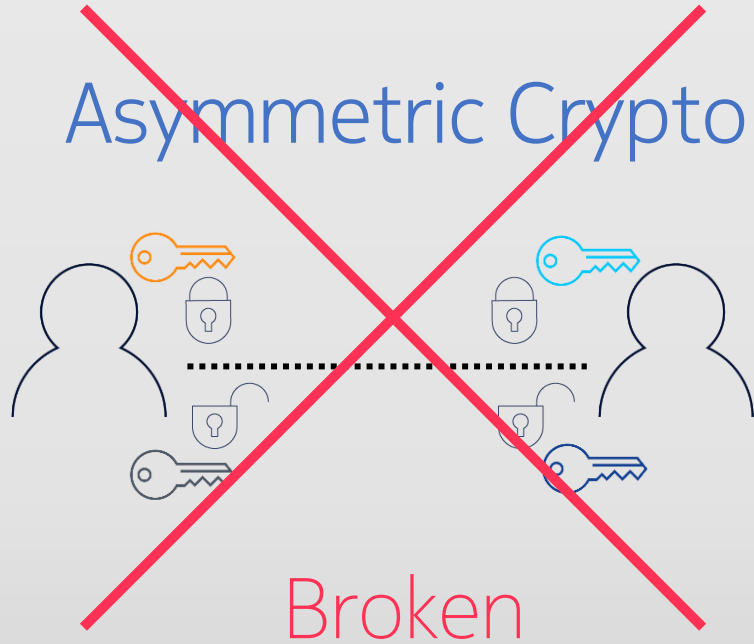
K Tirumaleswar Reddy (Nokia)

Why the draft is relevant to PQUIP?

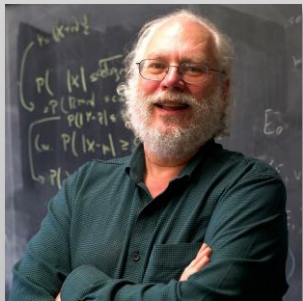
- The draft explains why engineers need to be aware of and understand post-quantum cryptography.
- It emphasizes the potential impact of Cryptographically Relevant Quantum Computers (CRQCs) on current cryptographic systems and the need to transition to post-quantum algorithms to ensure long-term security.
- Not much cryptographic math is used in explained in this draft but rather an overview of post quantum use in protocols.

Impact of Quantum Computers in Cryptography

Asymmetric Crypto

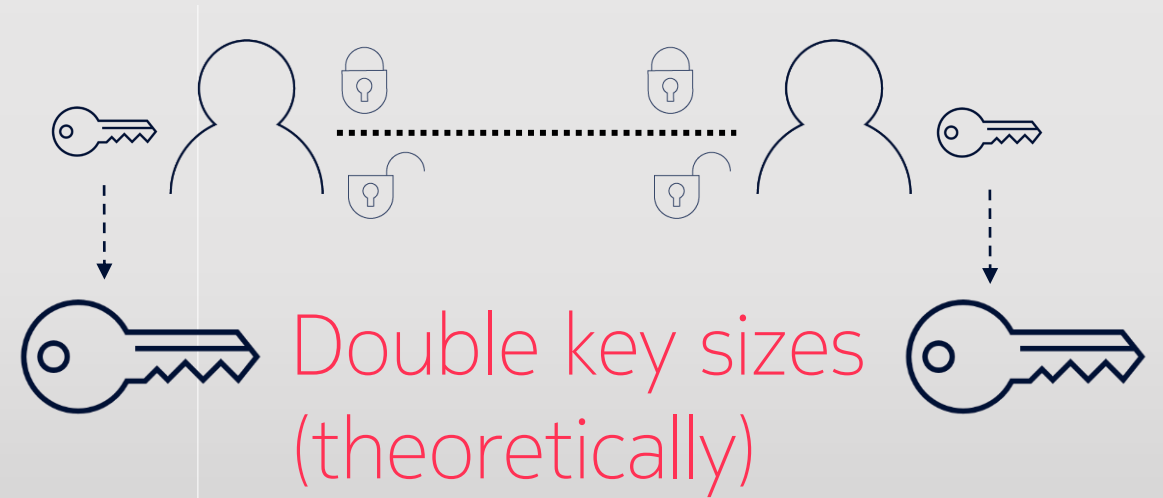


Broken



Peter Shor
Algorithm for prime factorization of large integers

Symmetric Crypto



Double key sizes
(theoretically)



Lov Kumar Grover
shows how to search in \sqrt{N}

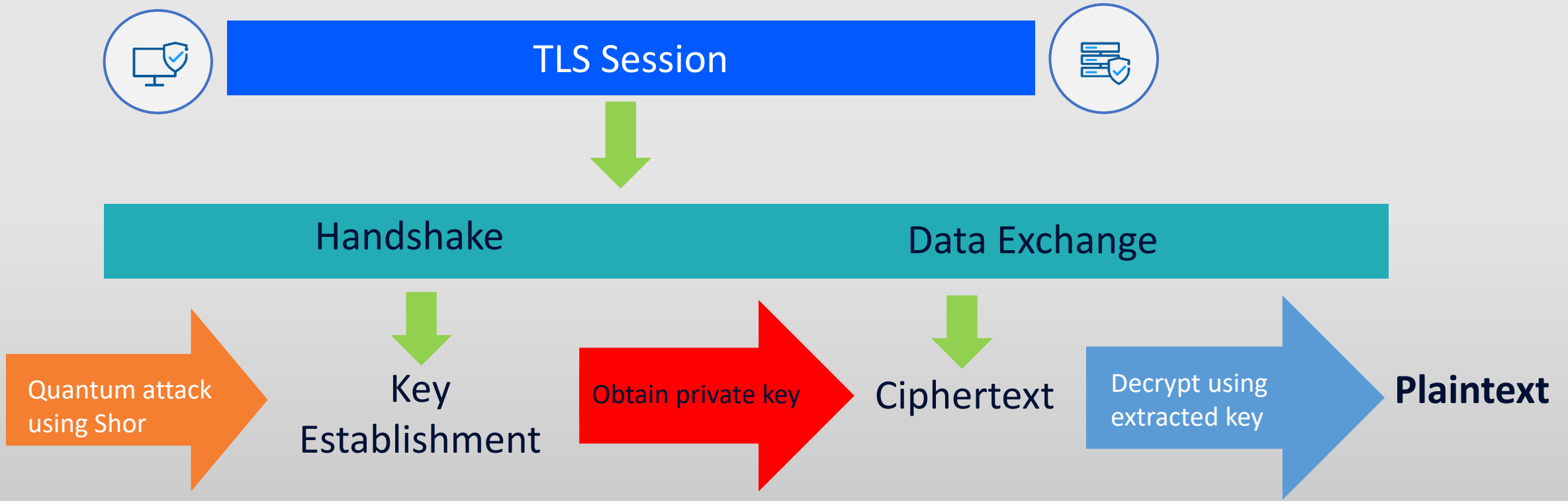
Symmetric Cryptography (Grover's algorithm)

- Grover's algorithm theoretically requires us to double the key-size (AES128 -> AES256)
- This would affect all symmetric crypto algorithms that are used currently.
- Yet, this is a misconception:
 - Grover's algorithm is highly **non parallelizable**
 - Even thousands of QCs running in parallel would offer minimal gains in breaking symmetric keys.
- NIST still standardizes AES-128 [\(link\)](#).

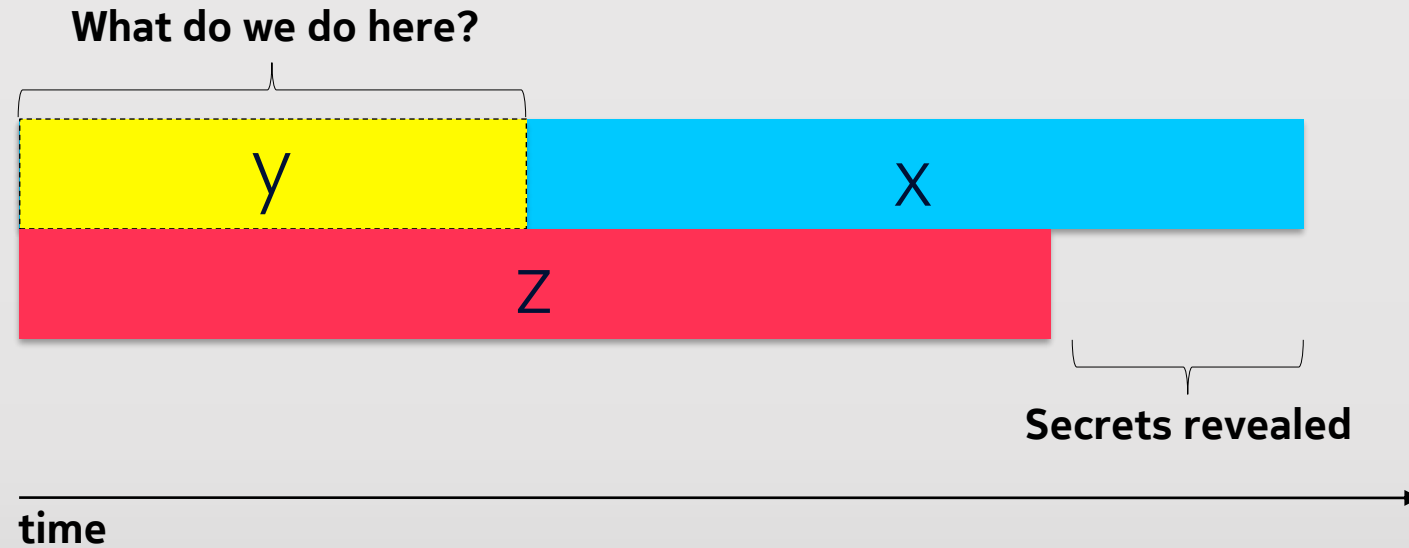
PQC standardization timeline



A Harvest Now and Decrypt Later (HNDL) Attack on TLS



Mosca's theorem of cybersecurity in the quantum era¹

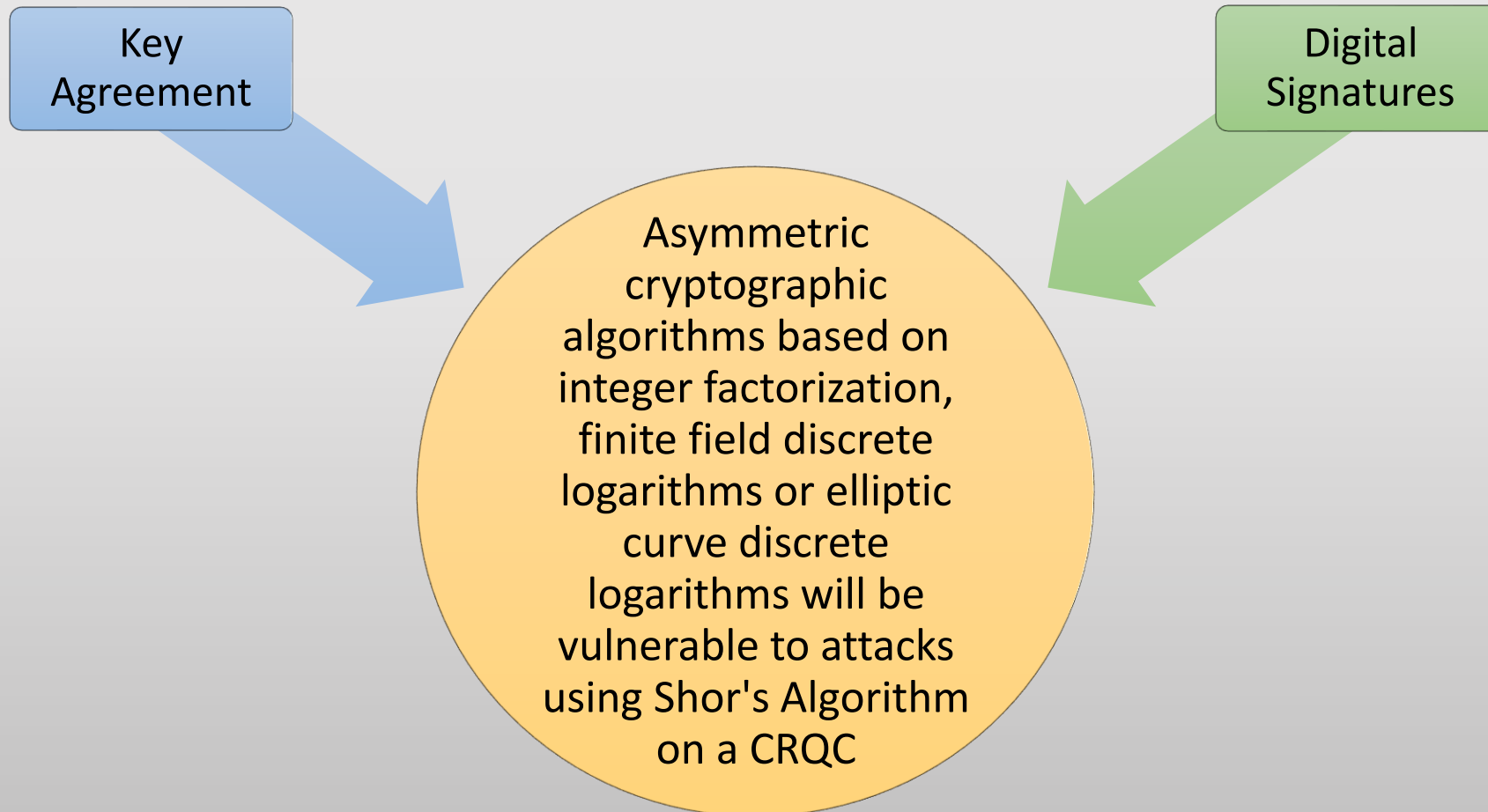


If $x + y > z$, then start worrying

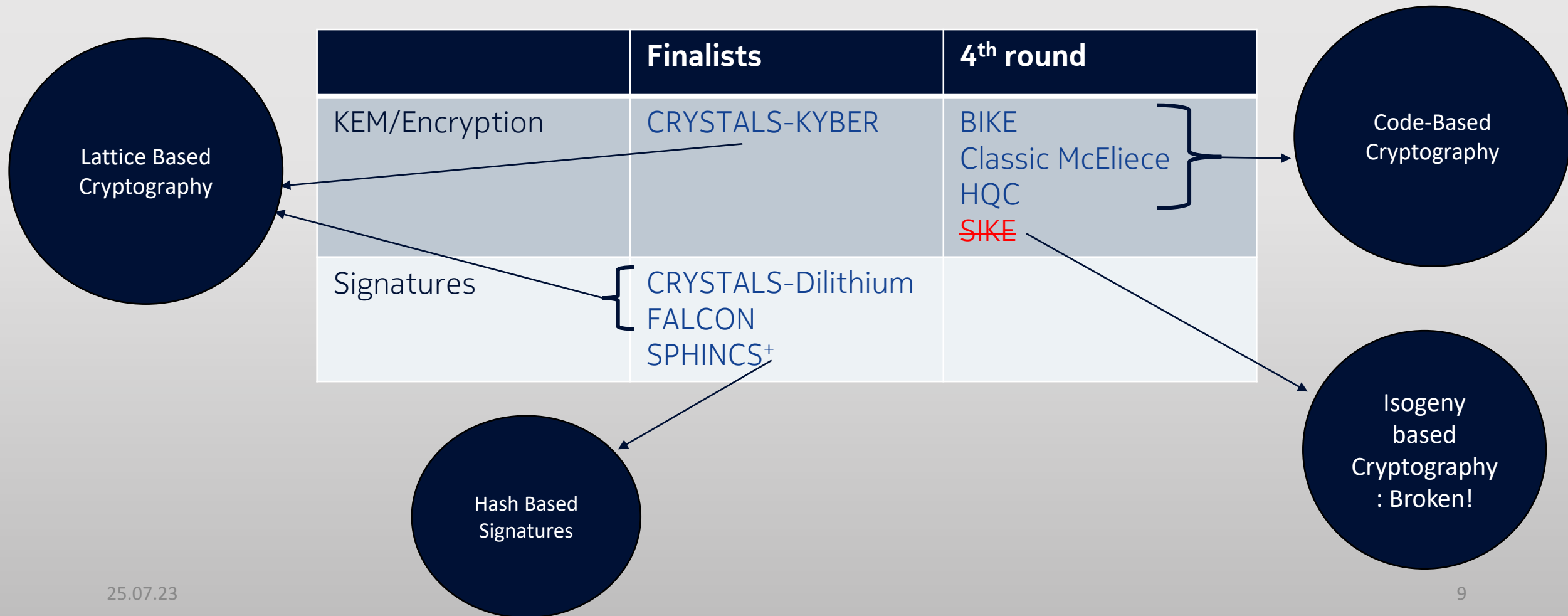
- x:** time we want to keep our systems secure
- y:** time to deploy a quantum-safe migration plan
- z:** time to build a large-scale quantum computer (2030s?)

[1] <https://eprint.iacr.org/2015/1075.pdf>

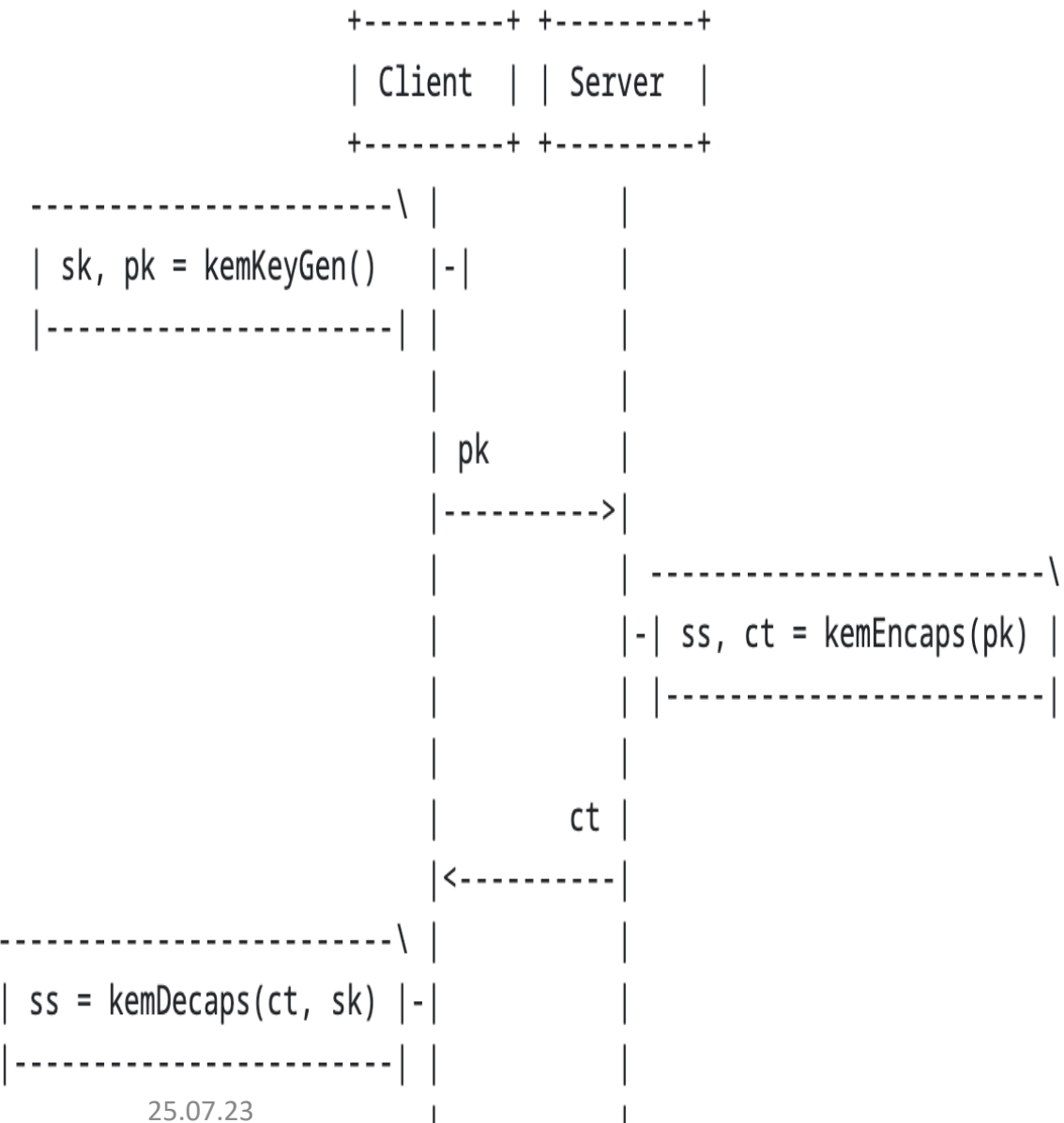
Traditional Cryptographic Primitives that Could Be Replaced by PQC



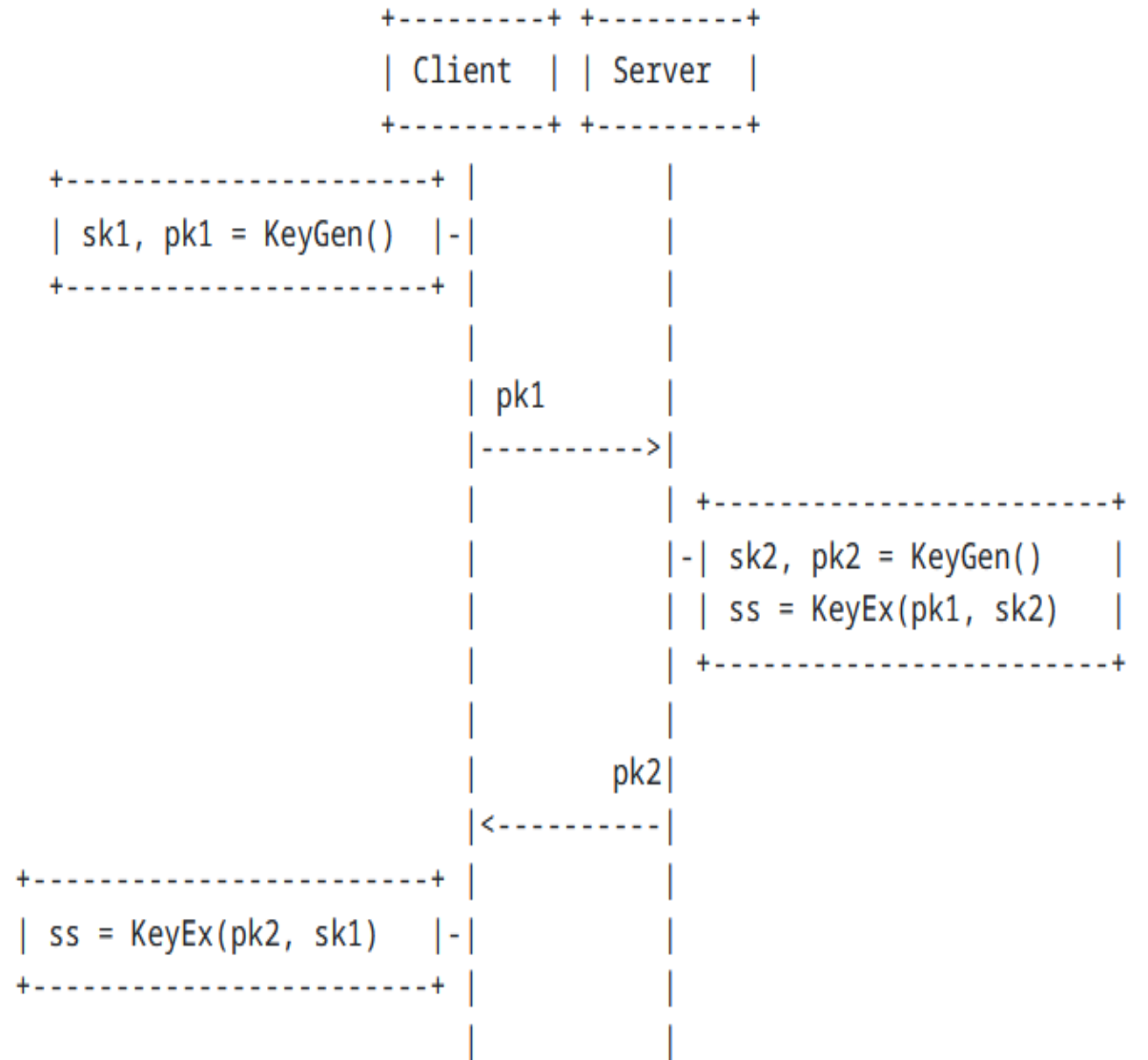
NIST Candidates Selected for Standardization/4th Round Candidates



KEM



DH KEX



KEM based AKE



Security property for KEM and Signatures

IND-CCA2

- IND-CCA2 (Indistinguishability under adaptive Chosen-Ciphertext Attack) is an advanced security notion for encryption schemes. It ensures the confidentiality of the plaintext, resistance against chosen-ciphertext attacks, and prevents the adversary from forging new ciphertexts.
- Kyber, BIKE, Classic McEliece provide IND-CCA2 security

EUFCMA

- EUFCMA (Existential Unforgeability under Chosen Message Attack) [GMR88] is a security notion for digital signature schemes. It guarantees that an adversary, even with access to a signing oracle, cannot forge a valid signature for an arbitrary message. EUFCMA provides strong protection against forgery attacks, ensuring the integrity and authenticity of digital signatures by preventing unauthorized modifications or fraudulent signatures.
- Dilithium, Falcon and Sphincs+ provide EUFCMA security.

Details of XMSS and LMS

- PQC: XMSS (RFC8391) and LMS (RFC8554) are stateful hash-based signature schemes.
- Reusing a secret key state compromises cryptographic security guarantees.
- Signing a potentially large but fixed number of messages
- The number of signing operations depends upon the size of the tree.
- Increasing the number of layers reduces key generation time exponentially and signing time linearly at the cost of increasing the signature size linearly.

Hash-then-Sign vs Sign-then-Hash

- Hash-then-Sign: Fixed size digest of the message is signed.
- Rely on the collision-resistance of the hash function.
- Reduces the size of signed messages.
- Protocols like TLS 1.3 and DNSSEC use the Hash-then-Sign paradigm.
- PQC Signature schemes internally apply hash functions.

PQ/T Hybrid Confidentiality

- Protect from protect from "Harvest Now, Decrypt Later" attack
- Concatenate hybrid key agreement scheme
 - Hybrid key exchange in TLS 1.3 <https://datatracker.ietf.org/doc/draft-ietf-tls-hybrid-design/>
 - It provides hybrid confidentiality but does not address hybrid authentication
 - the client's key-share contains two component public keys, one for a post-quantum algorithm and one for a traditional algorithm (ECDH ephemeral key-share)
 - For the server's share, concatenation of ct (ciphertext) and ephemeral key-share (ECDH)
 - hybrid secret by concatenating the two shared secrets
- Cascade hybrid key agreement scheme
 - Multiple Key Exchanges in the Internet Key Exchange Protocol Version 2 (IKEv2) <https://datatracker.ietf.org/doc/rfc9370/>
 - Allows the negotiation of one or more PQC algorithms to exchange data, in addition to the existing (EC)DH key exchange data.

PQ/T Hybrid Authentication

- Protect from on-path attacker using CRQC
- Authentication through a PQ/T hybrid scheme or a PQ/T hybrid protocol, as long as at least one component algorithm remains secure to provide the intended security level.
- The frequency and duration of system upgrades and the time when CRQCs will become widely available need to be weighed in to determine **whether and when to support the PQ/T Hybrid Authentication property.**
- Discussions in LAMPS WG to use PQ/T Hybrid Certificate

Security levels (PQC Algorithms: NIST)

PQ Security Level	AES/SHA3 hardness	PQC Algorithm
1	Find optimal key in AES-128	Kyber512, Falcon512, Sphincs+SHA256 128f/s
2	Find optimal collision in SHA3-256	Dilithium2
3	Find optimal key in AES-192	Kyber768, Dilithium3, Sphincs+SHA256 192f/s
4	Find optimal collision in SHA3-384	No algorithm tested at this level
5	Find optimal key in AES-256	Kyber1024, Falcon1024, Dilithium5, Sphincs+SHA256 256f/s

Key takeaway: Users can leverage the required algorithm based on the security level based on their use case. The security is defined as a function of resources required to break AES and SHA3 algorithms, i.e., optimal key recovery for AES and optimal collision attacks for SHA3.

Key & ciphertext/signatures of PQC Algorithms on different security levels

PQ Security Level	Algorithm	Public key size (in bytes)	Private key size (in bytes)	Ciphertext/Signature size (in bytes)
1	Kyber512	800	1632	768
1	Falcon512	897	1281	666
2	Dilithium2	1312	2528	2420
3	Kyber768	1184	2400	1088
5	Falcon1024	1793	2305	1280
5	Kyber1024	1568	3168	1588

SPHINCS+ and its many variants (Simple only)

SPHINCS+ algorithm security levels for different categories i.e., (f) for fast verification and (s) for compactness/smaller. Both SHA256 and SHAKE-256 parametrisation output the same signature sizes, so both have been included.

PQ Security Level	Algorithm	Public key size (in bytes)	Private key size (in bytes)	Signature size (in bytes)
1	SPHINCS+--{SHA2,SHAKE}-128f	32	64	17088
1	SPHINCS+--{SHA2,SHAKE}-128s	32	64	7856
3	SPHINCS+--{SHA2,SHAKE}-192f	48	96	35664
3	SPHINCS+--{SHA2,SHAKE}-192s	48	96	16224
5	SPHINCS+--{SHA2,SHAKE}-256f	64	128	49856
5	SPHINCS+--{SHA2,SHAKE}-256s	64	128	29792

Falcon vs Dilithium vs Sphincs+

- Dilithium is known for its relatively fast signature generation, while Falcon can provide more efficient signature verification.
- Falcon also has lower key and signature sizes as compared to Dilithium.
- SPHINCS+ offers smaller key sizes, larger signature sizes, slower signature generation, and slower verification when compared to Dilithium and Falcon.

Challenges in Falcon's Signing Operations

- Falcon's signing operations require constant-time, 64-bit floating point operations to avoid catastrophic side channel vulnerabilities. Doing this correctly (which is also platform-dependent to an extreme degree) is very difficult, as NIST's report noted.
- Providing a masked implementation of Falcon also seems impossible, per the authors at the RWPQC 2023 symposium earlier this year.

PQC vs Traditional KEMs/KEs

PQ Security Level	Algorithm	Public key size (in bytes)	Private key size (in bytes)	Ciphertext size (in bytes)
Traditional	P256_HKDF_SHA256	65	32	65
Traditional	P521_HKDF_SHA512	133	66	133
Traditional	X25519_HKDF_SHA256	32	32	32
1	Kyber512	800	1632	768
3	Kyber768	1184	2400	1088
5	Kyber1024	1568	3168	1588

PQC vs Traditional Signatures

PQ Security Level	Algorithm	Public key size (in bytes)	Private key size (in bytes)	Signature size (in bytes)
Traditional	RSA2048	256	256	256
Traditional	P256	64	32	64
1	Falcon512	897	1281	666
2	Dilithium2	1312	2528	768
3	Dilithium3	1952	4000	3293
5	Falcon1024	1793	2305	1280

Security Considerations - Cryptanalysis

- Classical cryptanalysis exploits weaknesses in algorithm design, mathematical vulnerabilities, or implementation flaws, whereas quantum cryptanalysis harnesses the power of CRQCs to solve specific mathematical problems more efficiently.
- Both pose threats to the security of cryptographic algorithms, including those used in PQC
- Developing and adopting new cryptographic algorithms resilient against these threats is crucial for ensuring long-term security in the face of advancing cryptanalysis techniques

Security Considerations - Cryptographic Agility

- Cryptographic agility is relevant for both classical and quantum cryptanalysis as it enables organizations to adapt to emerging threats, adopt stronger algorithms, comply with standards, and plan for long-term security in the face of evolving cryptanalytic techniques and the advent of CRQCs.
- Several PQC schemes are available that need to be tested; cryptography experts around the world are pushing for the best possible solutions, and the first standards that will ease the introduction of PQC are being prepared

Hybrid Key Exchange : Bridging the Gap Between Post-Quantum and Traditional Cryptography

- Post-quantum algorithms selected for standardization are relatively new and they have not been subject to the same depth of study as traditional algorithms.
- In addition, certain deployments may need to retain traditional algorithms due to regulatory constraints, for example FIPS compliance.
- Hybrid key exchange enables potential security against "Harvest Now, Decrypt Later" attack while not fully abandoning traditional cryptosystems.

Contributing to this document

- Comments and Suggestions are welcome
- The document is being collaborated on: [tiredy2/pqc-for-engineers \(github.com\)](https://github.com/tiredy2/pqc-for-engineers)
- E-mail archive: [pqc \(ietf.org\)](https://www.ietf.org/pqc)