CLOUDFLARE

Dawn of the Post-Quantum Internet

Dr Bas Westerbaan, Cloudflare Research

QSNS 2024, Paris, June 26th, 2024

About Cloudflare

We run a global network spanning 320 cities in over 120 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 57 million HTTP requests per second. >30% of Fortune 1000 are paying customers.

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.

I

The quantum menace

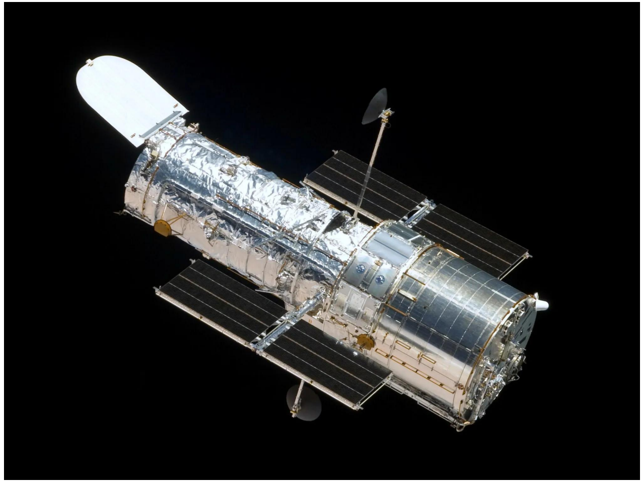
Quantum computers are great: efficient simulation of nature → new materials & medicine!

Minor inconvenience: they'll break most cryptography.

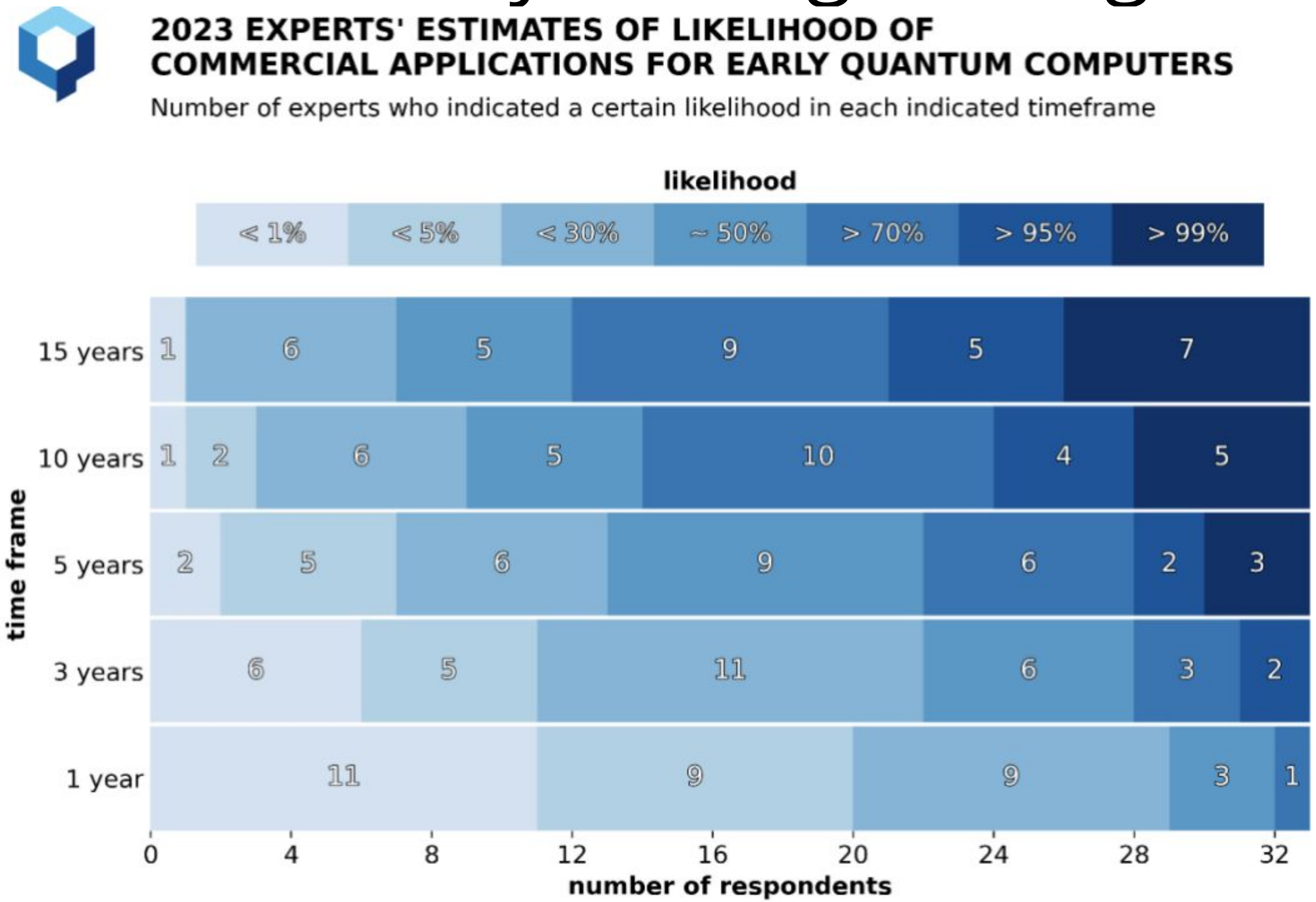
Why care now?

1. Captured data encrypted today can be decrypted by a quantum computer in the future.

2. Transitions take time.



When? Everyone is guessing

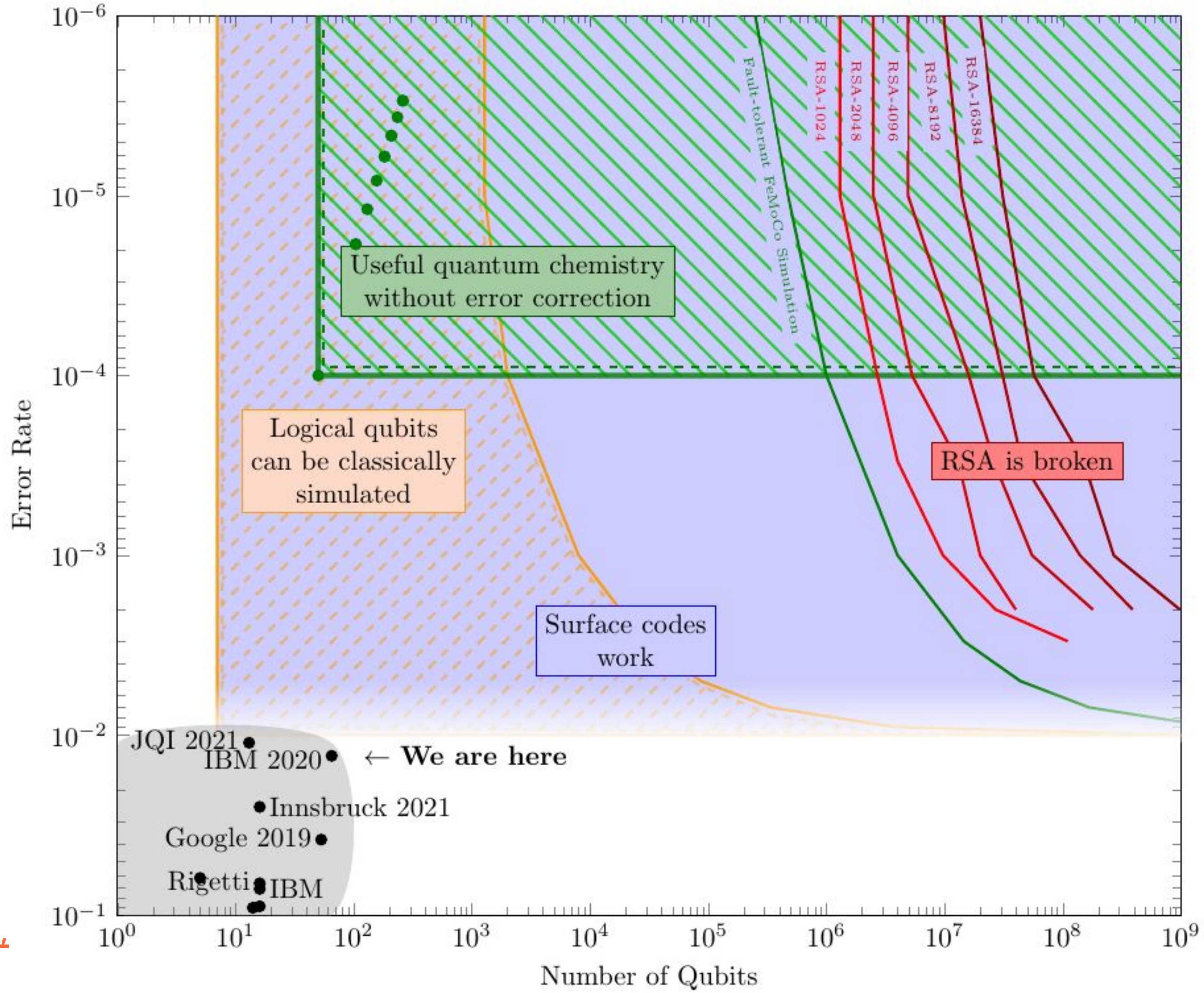


Interview with 32 experts, Mosca & Piani 2023

Don’t just count qubits!

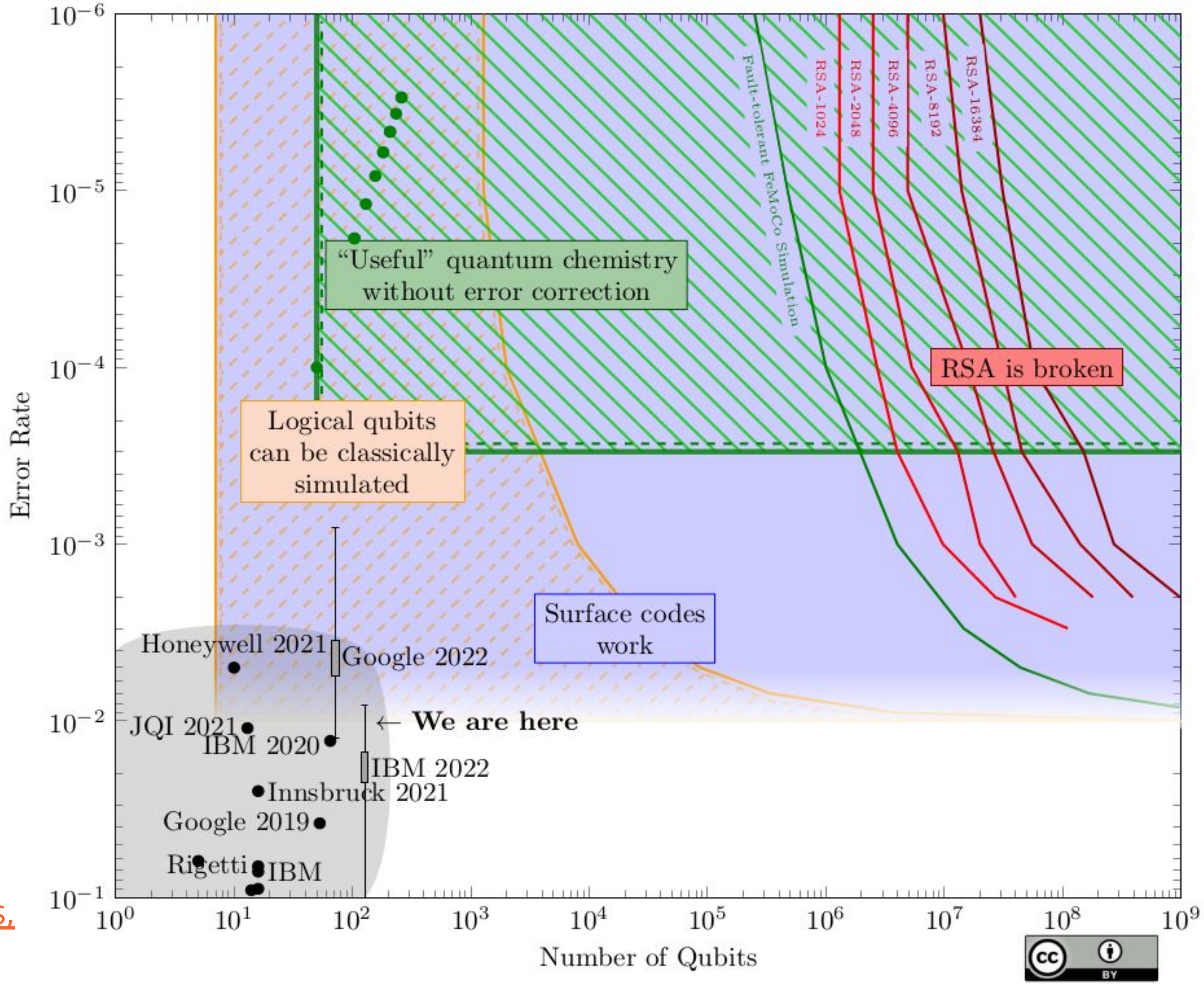
It’s about noise: Quantum computers are analog!

2021



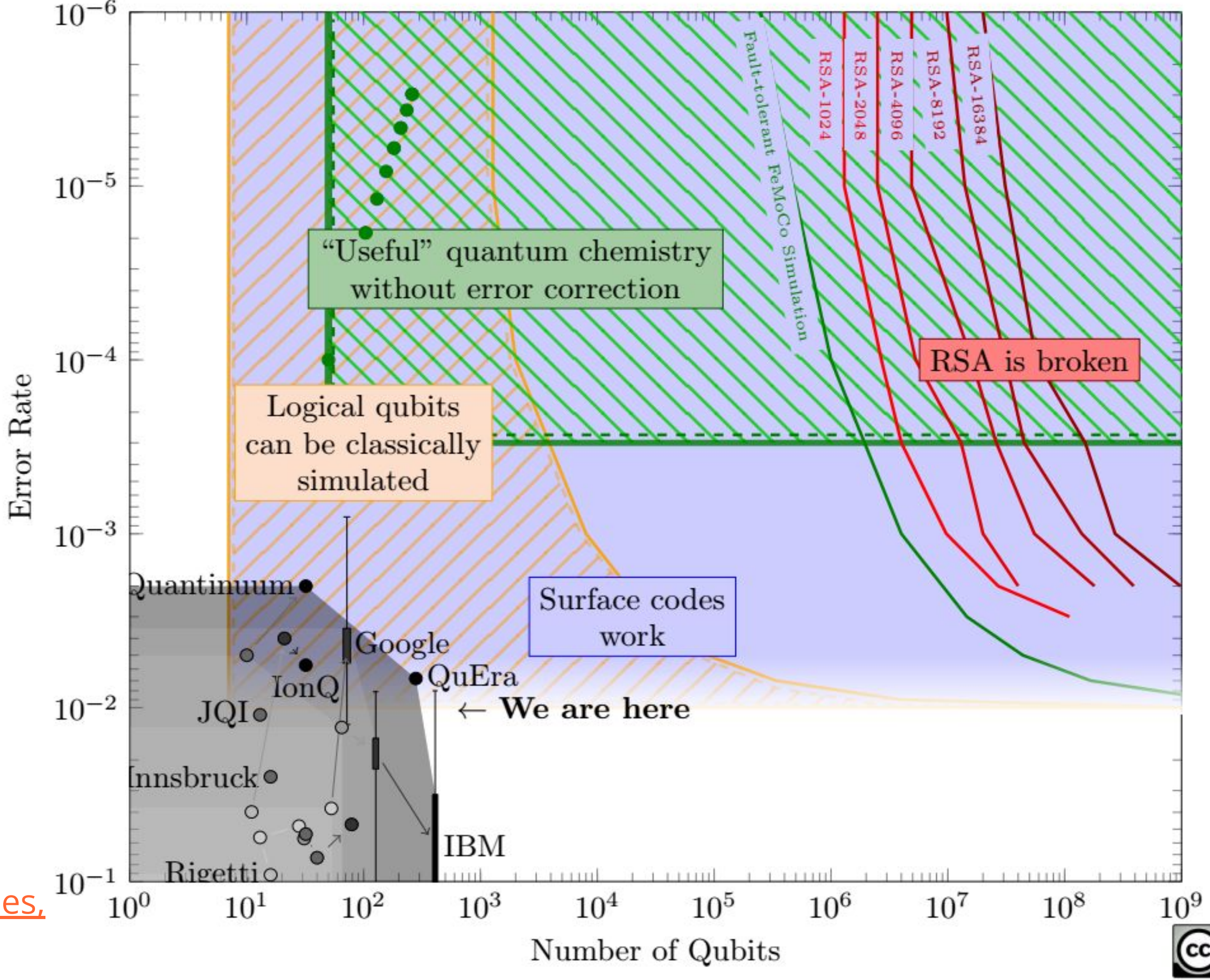
Credit: S. Jacques, U. of Waterloo.

2022



Credit: S. Jacques, U. of Waterloo.

2023



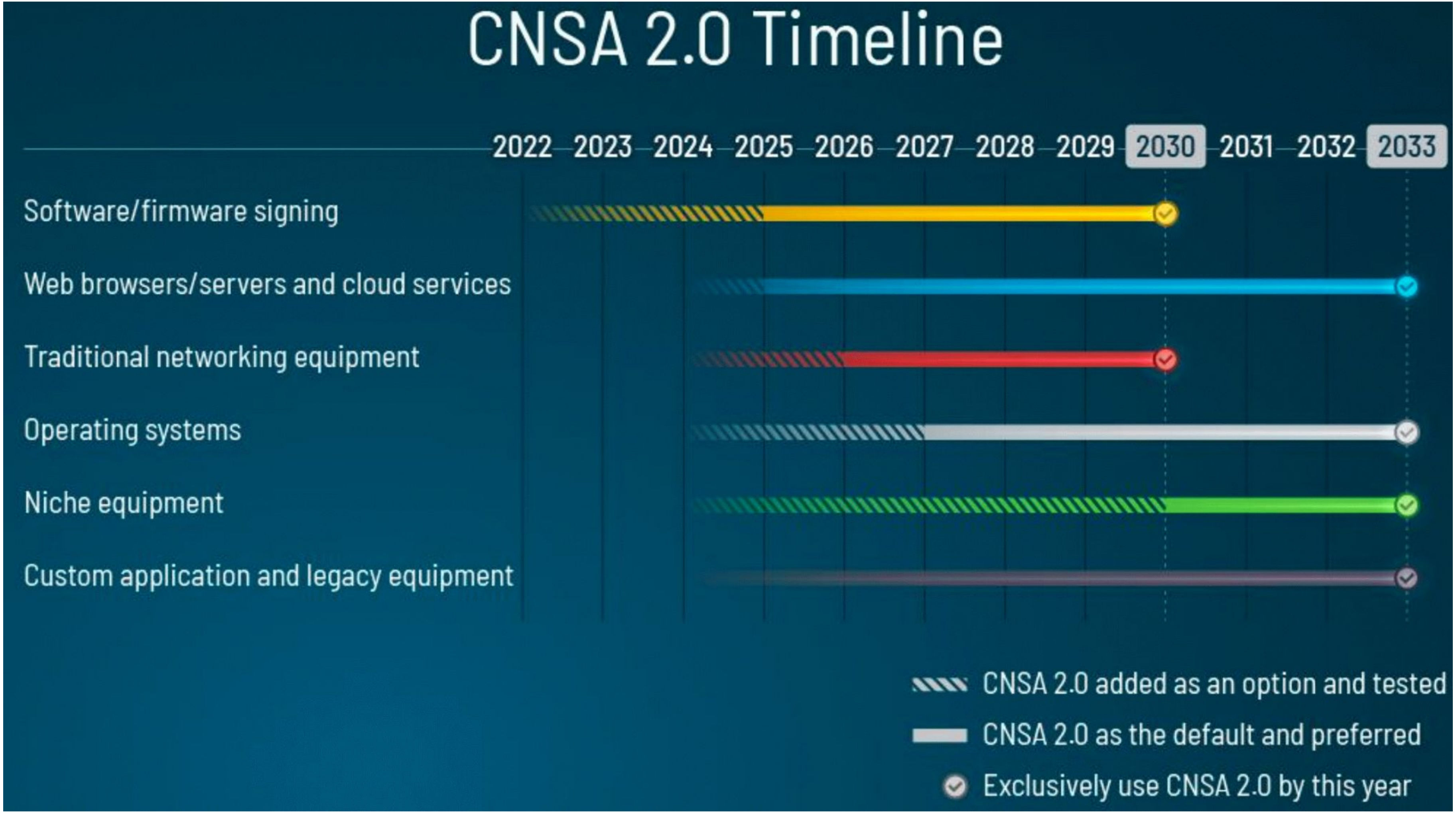
Credit: S. Jacques, U. of Waterloo.

BRIEFING ROOM

National Security Memorandum on Promoting United States Leadership in Quantum Computing While Mitigating Risks to Vulnerable Cryptographic Systems

MAY 04, 2022 • STATEMENTS AND RELEASES

To mitigate this risk, the United States must prioritize the timely and equitable transition of cryptographic systems to quantum-resistant cryptography, with the goal of mitigating as much of the quantum risk as is feasible by 2035. Currently, the Director of the National Institute of



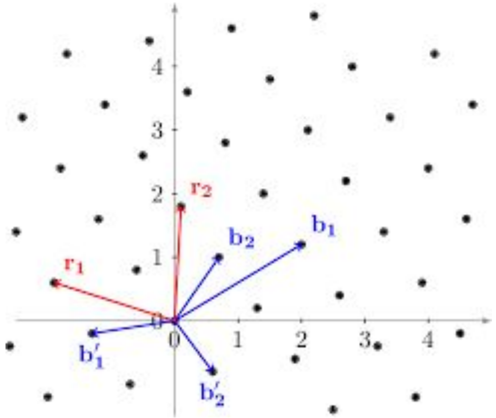
CNSA 2.0 Timeline
2022
2023
2024
2025
2026
2027
2028
2029
2030
2031
2032
2033
Software/firmware signing
Web browsers/servers and cloud services
Traditional networking equipment
Operating systems
Niche equipment
Custom application and legacy equipment
CNSA 2.0 added as an option and tested
CNSA 2.0 as the default and preferred
Exclusively use CNSA 2.0 by this year

Solution: Post-Quantum (PQ) Cryptography

NIST (SHA, AES) is running a competition since 2016.
We expect final standards late 2024.

Type	**Original name**	**NIST's name**	**FIPS number**
Signature	Dilithium	ML-DSA	204
	Falcon	FN-DSA	?
	SPHINCS$^+$	SLH-DSA	205
KEM (kex)	Kyber	ML-KEM	203

Mostly lattice-based cryptography.



II

State of the post-quantum Internet

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.

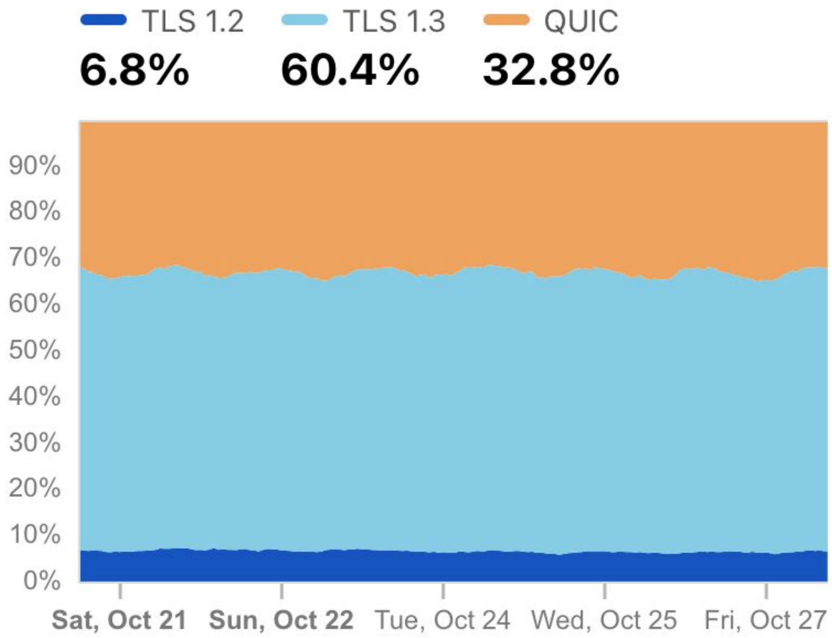
  We can't assume everyone is on fiber, or uses modern CPU, can store state, or can update at all.

- 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.

ML-KEM versus X25519

		Keyshares size (in bytes)		Ops/sec (higher is better)	
Algorithm	PQ	**Client**	**Server**	**Client**	**Server**
ML-KEM-512	✅	800	768	45,000	70,000
ML-KEM-768	✅	1,184	1,088	29,000	45,000
ML-KEM-1024	✅	1,568	1,568	20,000	30,000
X25519	❌	32	32	19,000	19,000

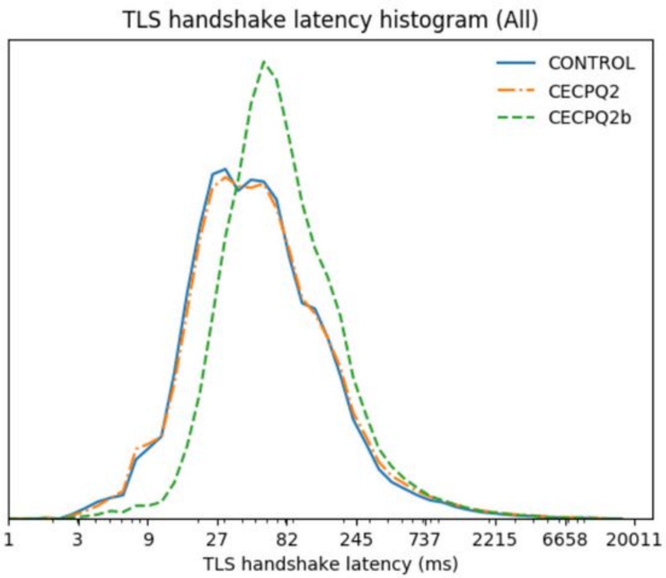
ML-KEM is faster than X25519, but more bytes on the wire.

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.

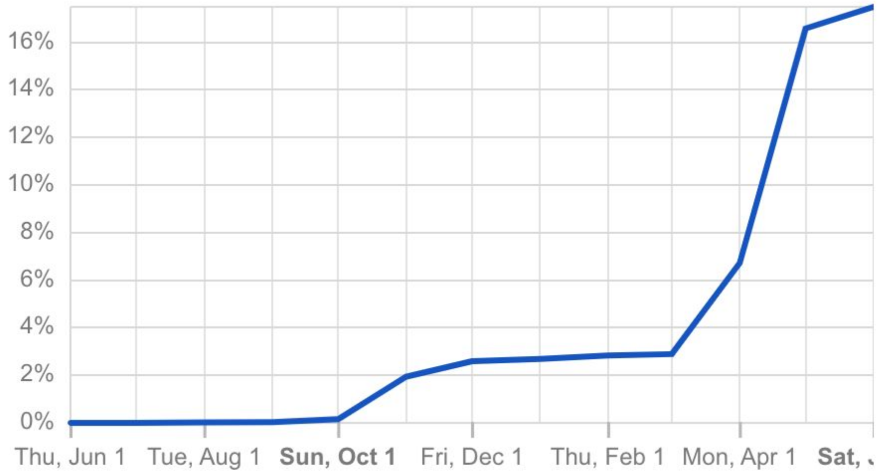


X25519. CECPQ2 is X25519+NTRU-HRSS (lattice) and CECPQ2b is X25519+SIKE (isogenies, broken)

Adoption

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

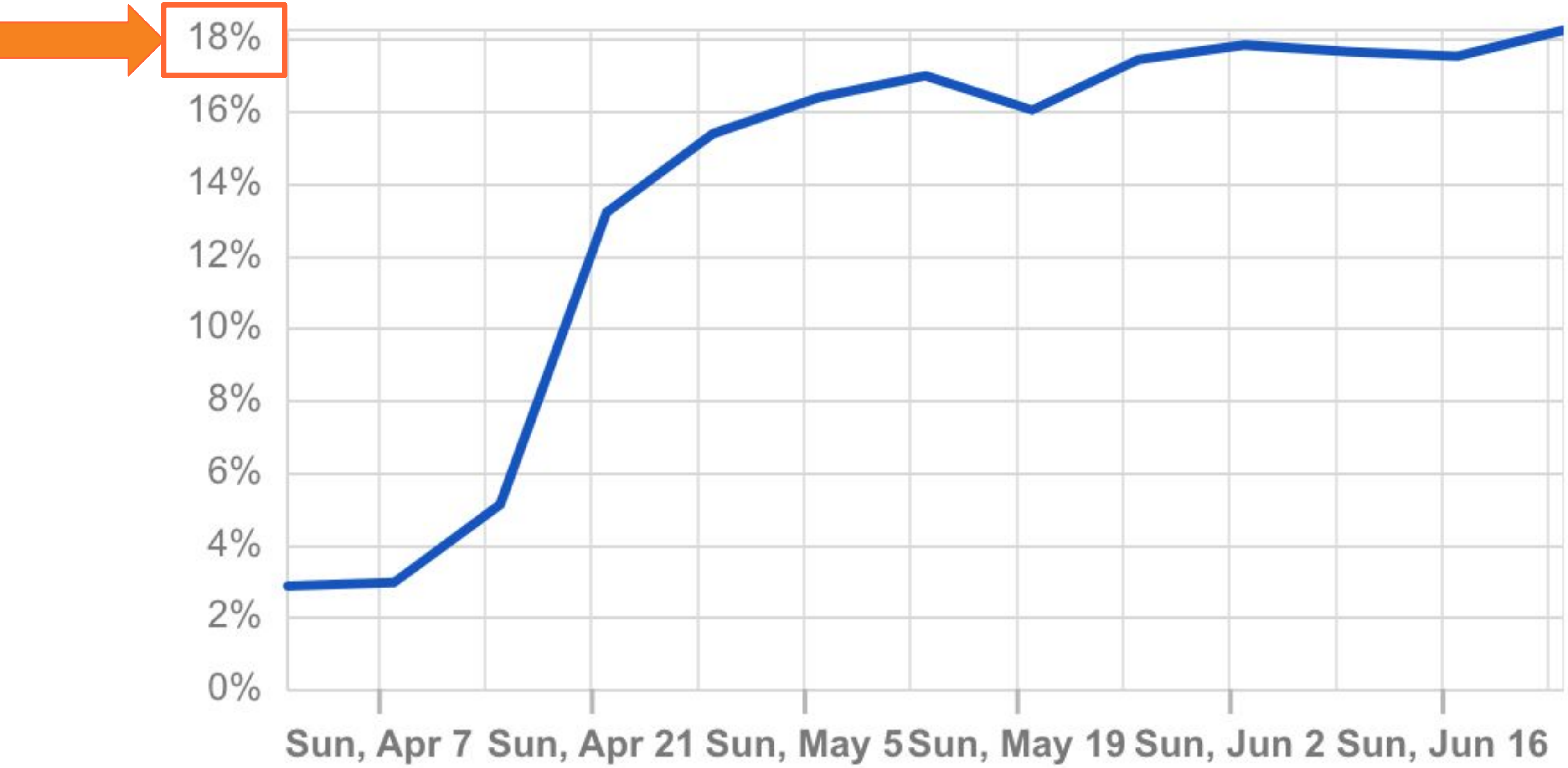
In 2023 Google enabled server-side as well.



Client PQE adoption on Cloudflare Radar

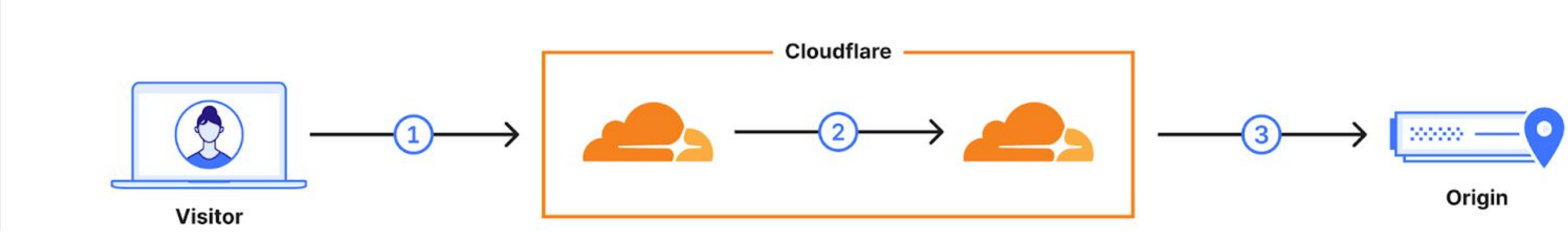
Browsers:

- Chrome & Edge: enabled by default on Desktop since April 2024.
- Firefox: small fraction; opt-in possible.



Client PQE adoption on Cloudflare Radar

Post-quantum to origins



We enabled support for PQ key agreement to origins (3).

0.5% of origins support PQ at time of writing.

0.34% incompatible when sending keyshare immediately.
We’ve reached out to customers to help remediate.

Not just a technical challenge

In 2023 we've also commenced migrating our internal connections to post-quantum key agreement.

Huge effort: every engineering team created inventory of cryptography used, risks, and planned/executed migration.

Majority of our internal connections are secured (prioritizing sensitive connections), but a long fat tail remains.

On the upside: we did not encounter any performance or compatibility issues.

Key agreement 🤝

Urgent and the easier of the two to deploy; with ~20% client adoption, the new modern baseline for the Internet .

That took 5 years.

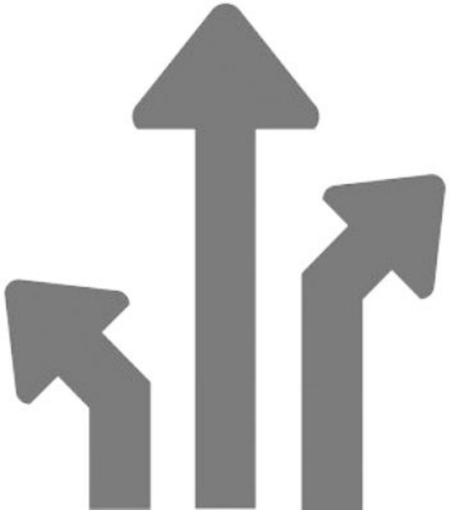
Signatures 🖋️

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.

Not just software update: also key rotation.



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

			Sizes (bytes)		CPU time (lower is better)	
		PQ	Public key	Signature	Signing	Verification
Standardized	Ed25519	❌	32	64	1 (baseline)	1 (baseline)
	RSA-2048	❌	256	256	70	0.3
NIST drafts	ML-DSA-44	✅	1,312	2,420	4.8	0.5
	FN-DSA-512	✅	897	666	8 ⚠️	0.5
	SLH-DSA-128s	✅	32	7,856	8,000	2.8
	SLH-DSA-128f	✅	32	17,088	550	7

blog.cloudflare.com/pq-2024

Online signing — Falcon's Achilles' heel

- For fast signing, FN-DSA requires a floating-point unit (FPU).
- We do not have enough experience running cryptography securely (constant-time) on the FPU.
- On commodity hardware, FN-DSA 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.)

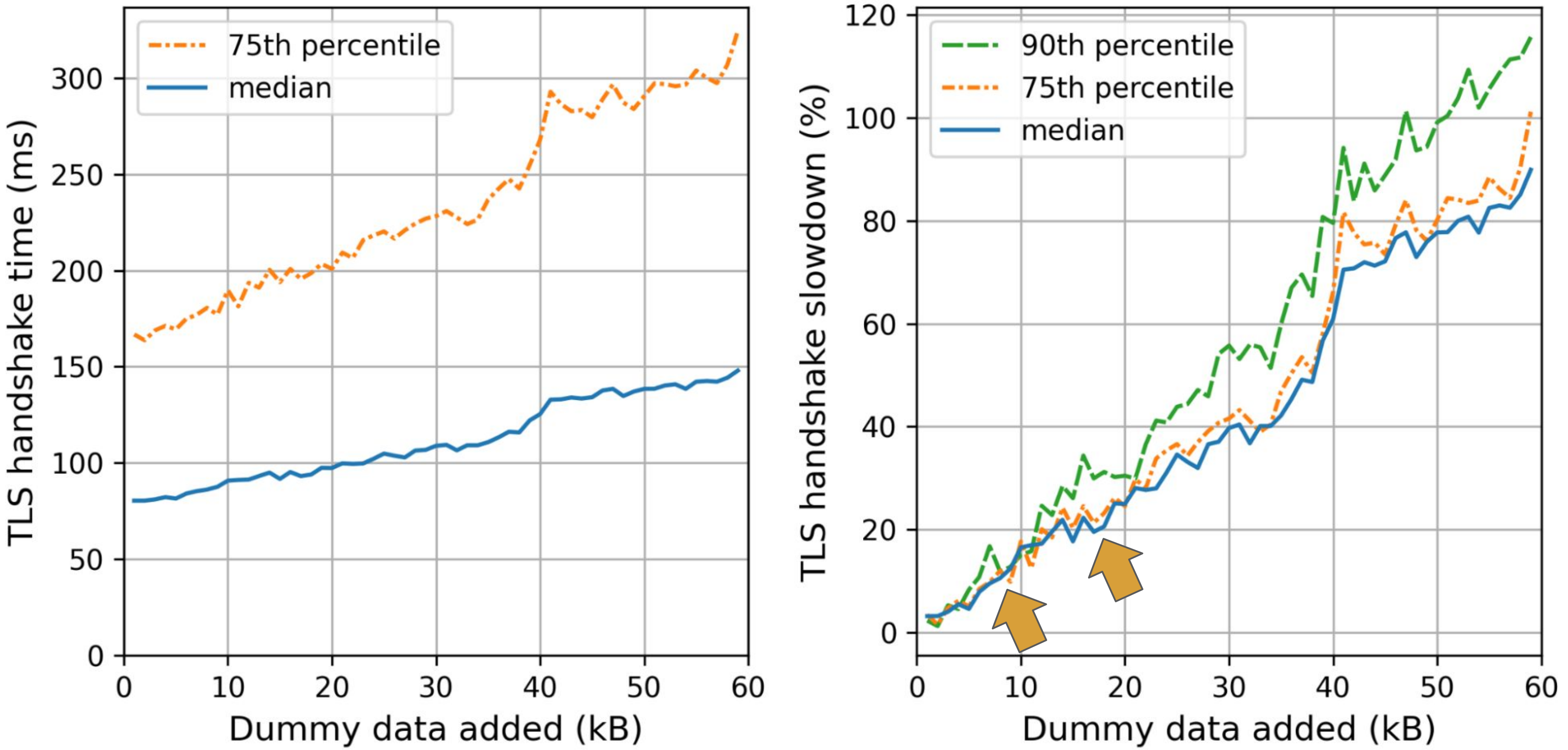
Using only ML-DSA-44

+17,144 bytes

Using ML-DSA for the TLS handshake and FN-DSA for the rest

+7,959 bytes

Is that too much? We had a look…



blog.cloudflare.com/sizing-up-post-quantum-signatures, 2021

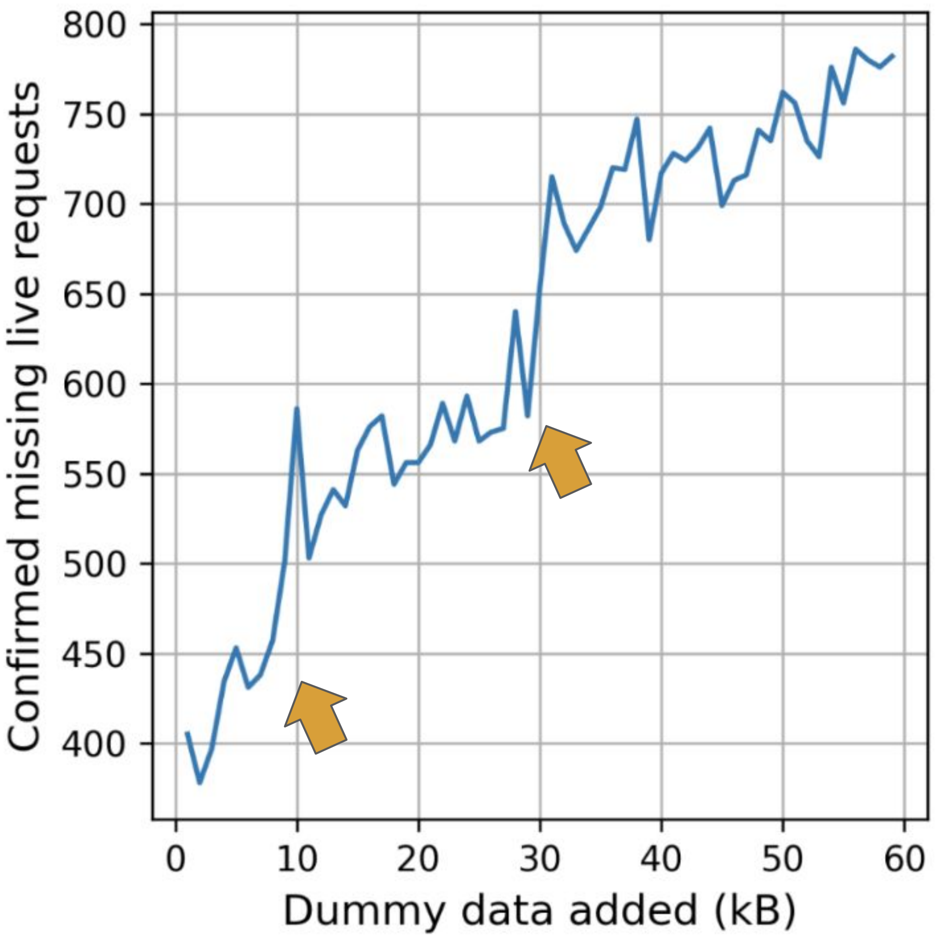
And, of course…

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.

Chrome’s take on post-quantum certificates

> Given these constraints, priorities, and risks, we think agility is more important than defining exactly what a post-quantum PKI will look like at this time. We recommend against *immediately* standardizing ML-DSA in X.509 for use in the *public* Web PKI via the CA/Browser Forum. We expect that ML-DSA, once NIST completes standardization, will play a part in a post-quantum Web PKI, but we’re focusing on agility first. This does not preclude introducing ML-DSA in X.509 as an option for private PKIs, which may be operating on more strict post-quantum timelines and have fewer constraints around certificate size, handshake latency, issuance transparency, and unmanaged endpoints.

Excerpt from Chrome’s May 2024 blog post.

NIST signature on-ramp

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

I will cover these later on.

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.

I will discuss these in more detail later on.

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 fancy cryptography, eg. privacy enhancing techniques such as anonymous credentials, is in the early stages.

Inventory of large deployments of fancy cryptography.

Questions so far?

III

Coping with post-quantum signatures

Recall: 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.

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	ML-DSA-44	✅	1,312	2,420	4.8	0.5
	FN-DSA-512	✅	897	666	8 ⚠️	0.5
	SLH-DSA-128s	✅	32	7,856	8,000	2.8
	SLH-DSA-128f	✅	32	17,088	550	7
Sample from signatures onramp	$MAYO_{one}$	✅	1,168	321	4.7	0.3
	$MAYO_{two}$	✅	5,488	180	5	0.2
	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 ML-DSA-44 for everything adds 17kB.

Using ML-DSA-44 for handshake and FN-DSA-512 for the rest, adds 8kB. ⚠️ Fast and secure FN-DSA-512 signing is hard to implement.

Using SLH-DSA-128s 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 ML-DSA-44 for handshake signature, adds 8kB.

⚠️ n=128 and w=256 instances are not standardised.

⚠️ We loose non-repudiation.

⚠️ Large precomputations/storage required for efficient signing.

⚠️ Challenging to keep state.

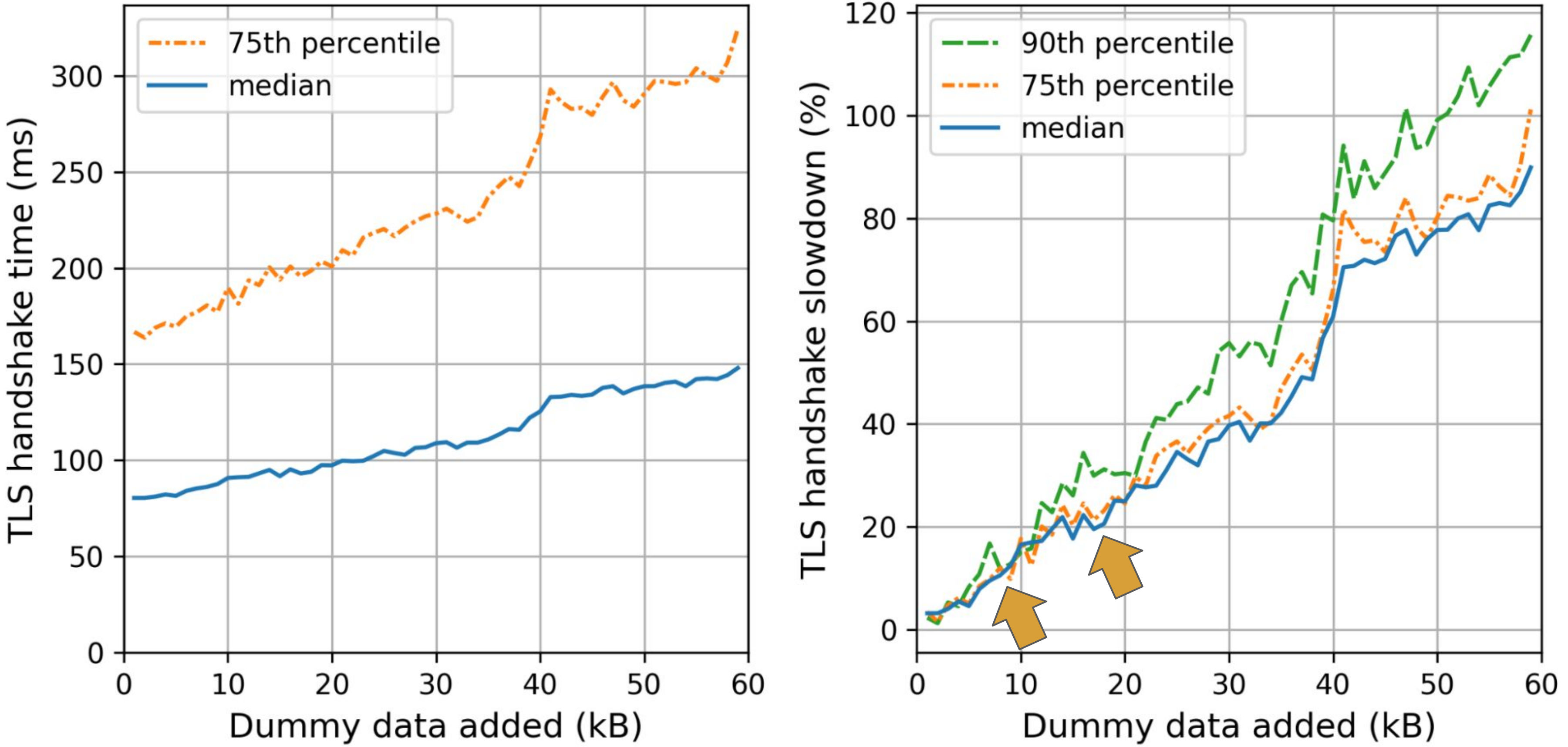
Concrete instances with onramp candidates

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

Using UOV ls-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 ls-pkc again, but combined with ML-DSA-44. Adds 7.4kB. Relatively conservative choice.

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



blog.cloudflare.com/sizing-up-post-quantum-signatures, 2021

Leaving out intermediates

Most browsers ship intermediate certificates, 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 ML-DSA-44 handshake signature with ML-KEM-512 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.

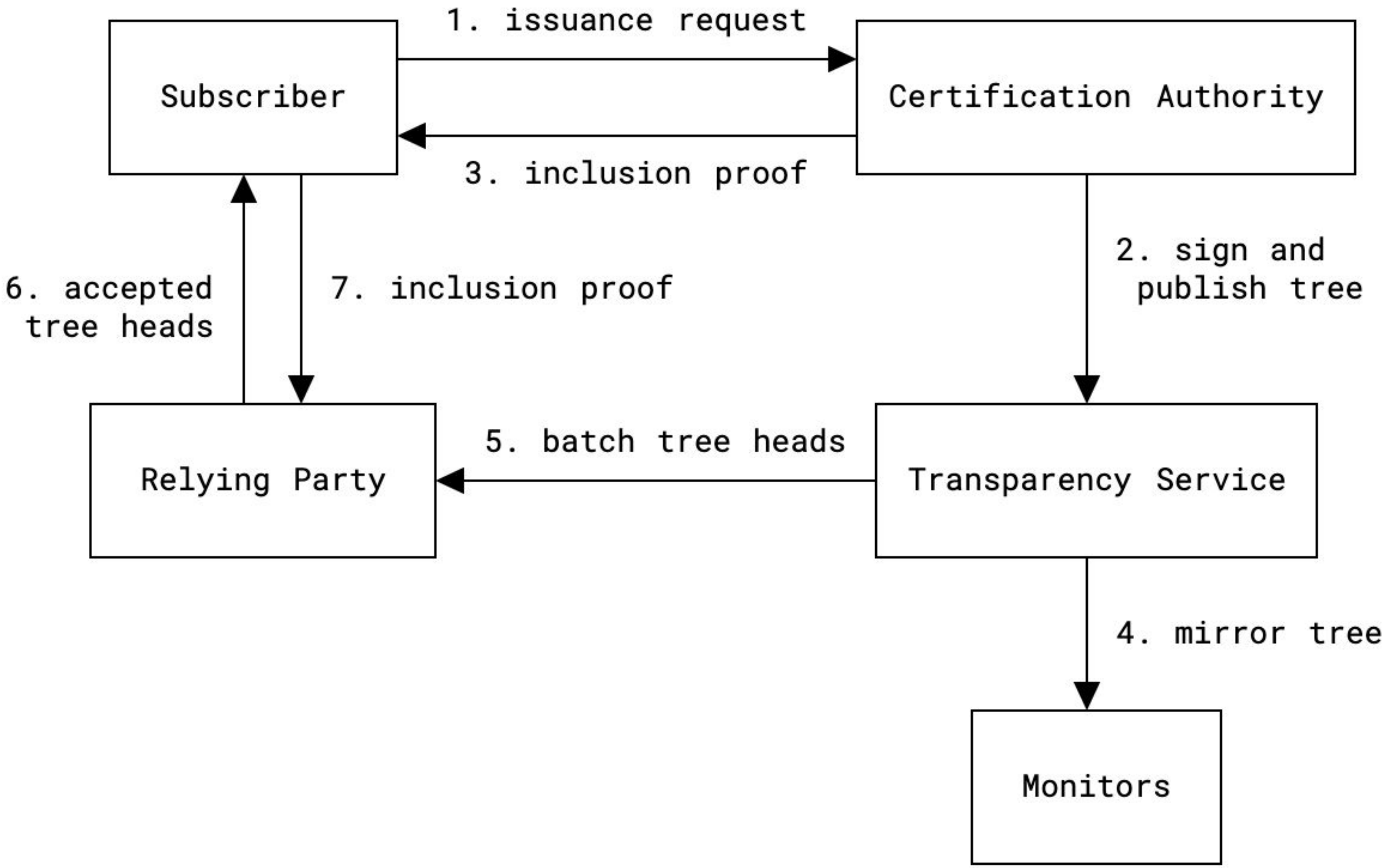
Not a complete replacement for current WebPKI: it's an optimisation of the common case and falls back to X.509+CT.

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.

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


Subscriber
Certification Authority
1. issuance request
3. inclusion proof
2. sign and publish tree
6. accepted tree heads
7. inclusion proof
Relying Party
Transparency Service
5. batch tree heads
4. mirror tree
Monitors

Merkle Tree Certificates in short (2)

A Merke 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 fallback X.509 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 1 billion unexpired certificates in CT.

If reissued every 7 days by one MTC CA, we'd have batches of 6 million assertions.

That amounts to authentication paths of 736 bytes, and with a ML-DSA-44 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.

Try MTC for yourself: PoC MTC CA.

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.

Thank you, questions?

References

- Further reading: state of the post-quantum Internet (2024).
- Follow adoption on Cloudflare Radar.
- Check out out pq.cloudflareresearch.com for
  - technical details on our deployment;
  - pointers to software support for PQ to experiment; and
  - more references.
- Reach out: ask-research@cloudflare.com

Backup slides

Post-quantum crypto should be free, so we’re including it for free, forever

03/16/2023

Wesley Evans

Bas Westerbaan

7 min read

This post is also available in 简体中文, 日本語, Deutsch, Français and Español.

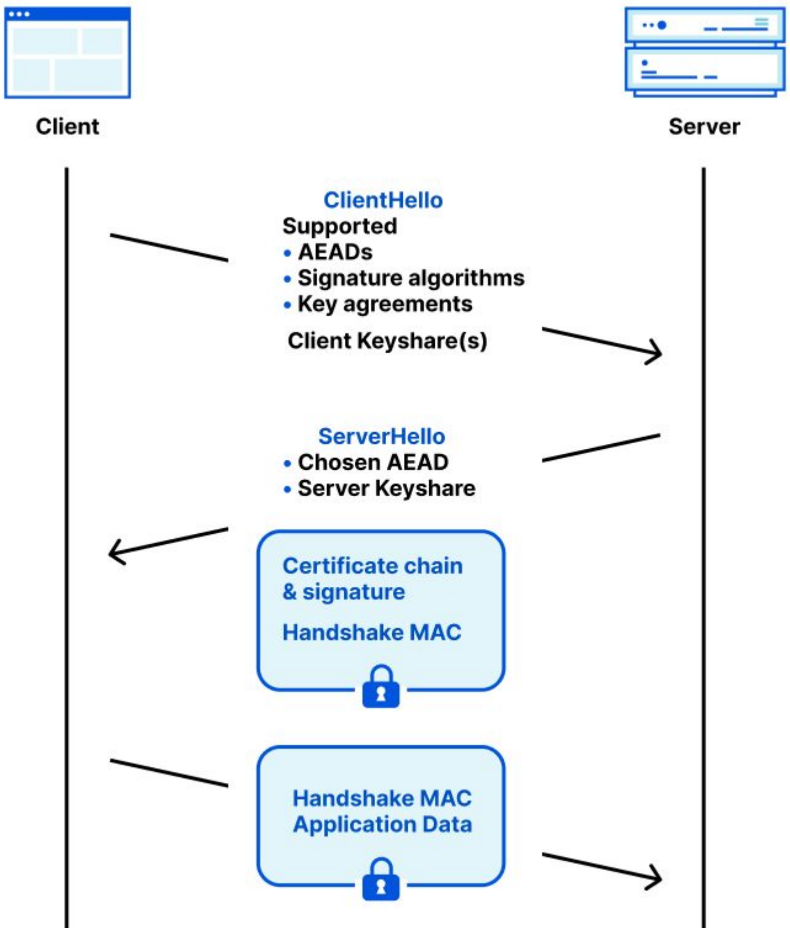
blog.cloudflare.com/post-quantum-crypto-should-be-free

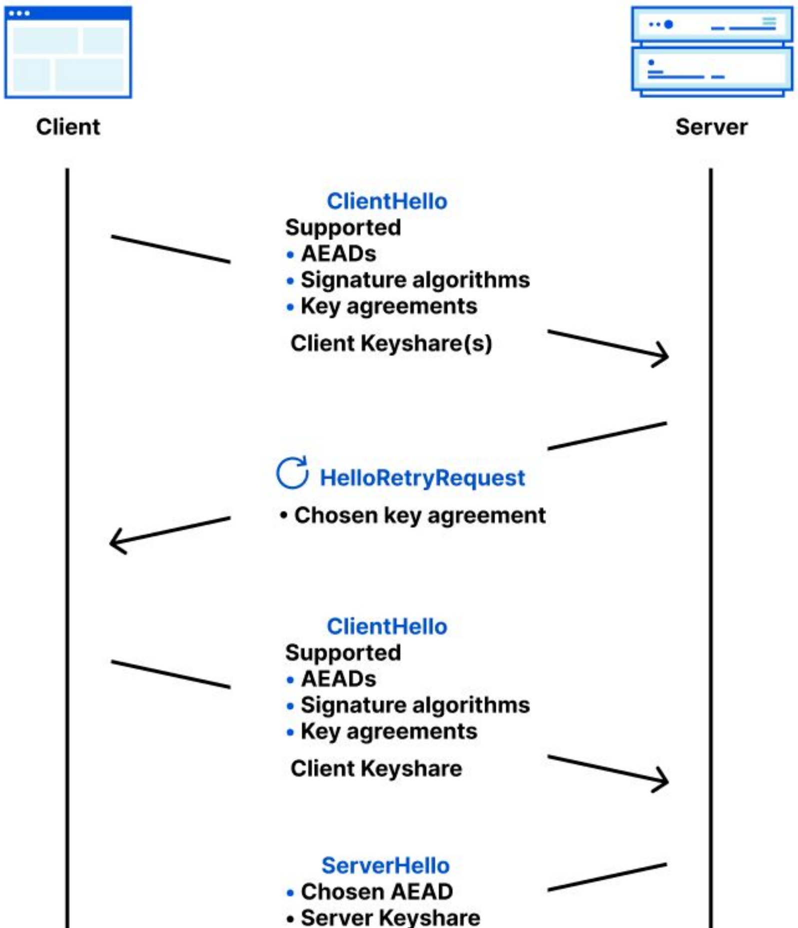
```c
static inline int64_t
fpr_rint(fpr x)
{
	/*
	 * We do not want to use llrint() since it might be not
	 * constant-time.
	 *
	 * Suppose that x >= 0. If x >= 2^52, then it is already an
	 * integer. Otherwise, if x < 2^52, then computing x+2^52 will
	 * yield a value that will be rounded to the nearest 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 FN-DSA as submitted to round 3 is not constant-time on ARMv7 as claimed.

Can you spot the error?

TLS 1.3 handshake





KEM versus Diffie–Hellman