

Post-Quantum

Cryptography Conference

Coping with Post-Quantum Signatures in the WebPKI

Bas Westerbaan

Research Engineer at Cloudflare



CLLOUDFLARE®

Coping with PQ Signatures in the WebPKI

Dr Bas Westerbaan, Cloudflare Research

PKI Consortium Post-Quantum Cryptography Conference, AMS, Nov 8th, 2023

This technical break-out

We start with a brief recap of the many signatures in the **WebPKI**.

Then we take measure of the current (draft) offering of **post-quantum signature schemes** and those being considered in the on-ramp.

And as main course, we cover the three coping strategies:

- Leaving out **intermediates**
- **KEMTLS**: using key agreement as authentication
- **Merkle Tree Certificates**

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	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.

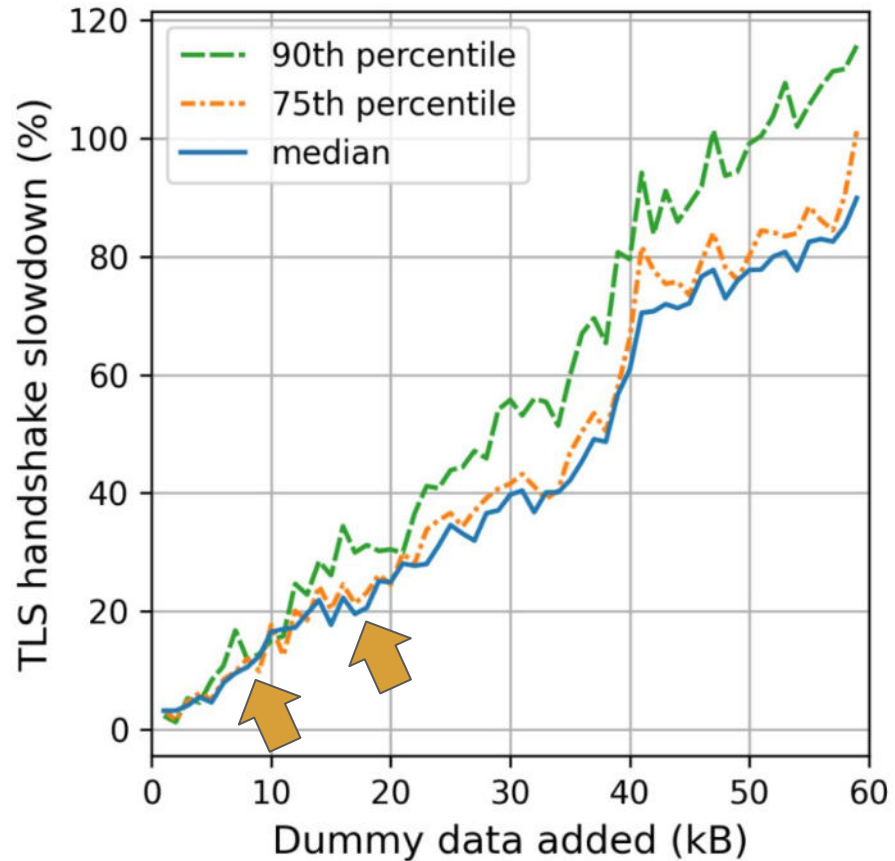
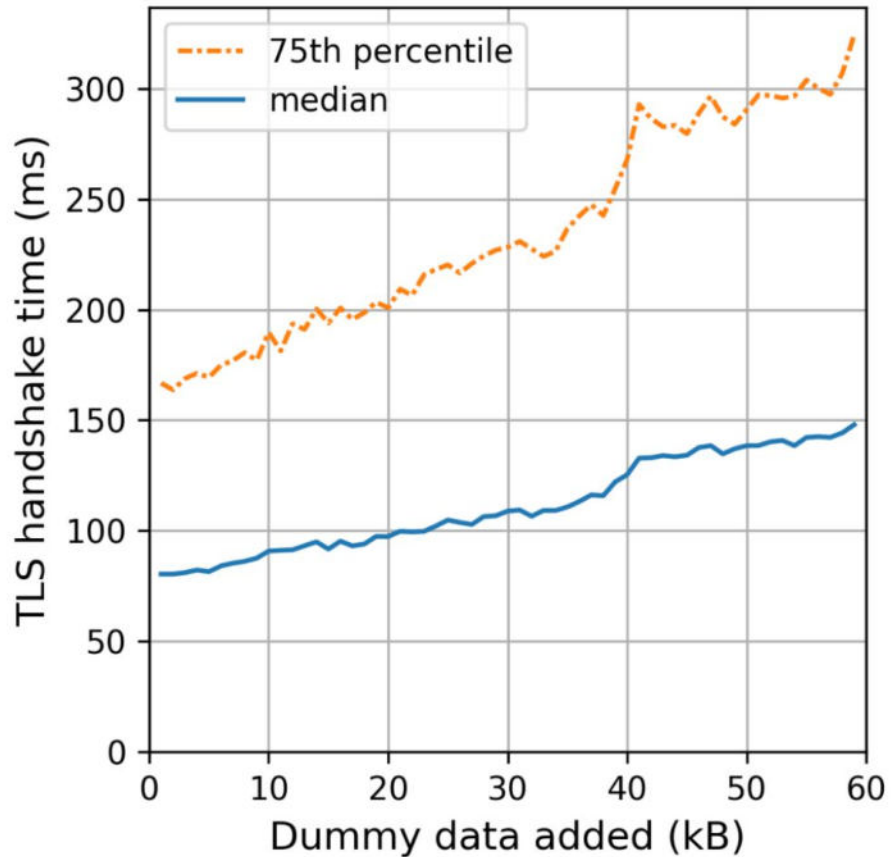
Concrete instances with on-ramp candidates

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.

Thank you, questions?

And do please reach out if you want to collaborate on testing these approaches.

Post-Quantum

Cryptography Conference



PKI
Consortium



KEYFACTOR



THALES



amsterdam
convention
bureau

