Let's stay encrypted

Bas Westerbaan, Cloudflare Research

RWPQC 2026, Taipei, March 8th

Post-quantum key agreement on the web

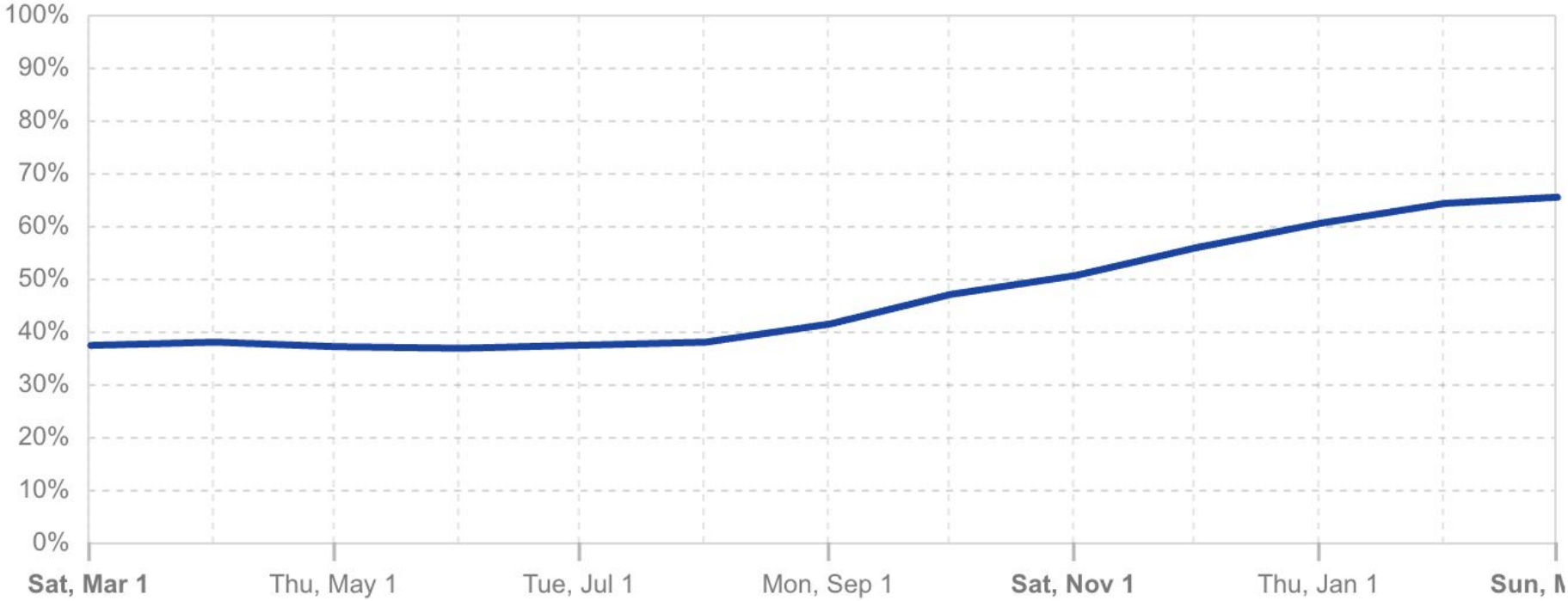
2025 was a great year for PQ key agreement

X25519MLKEM768 is enabled by default in many libraries, platforms and cloud providers in 2025, too many to list!

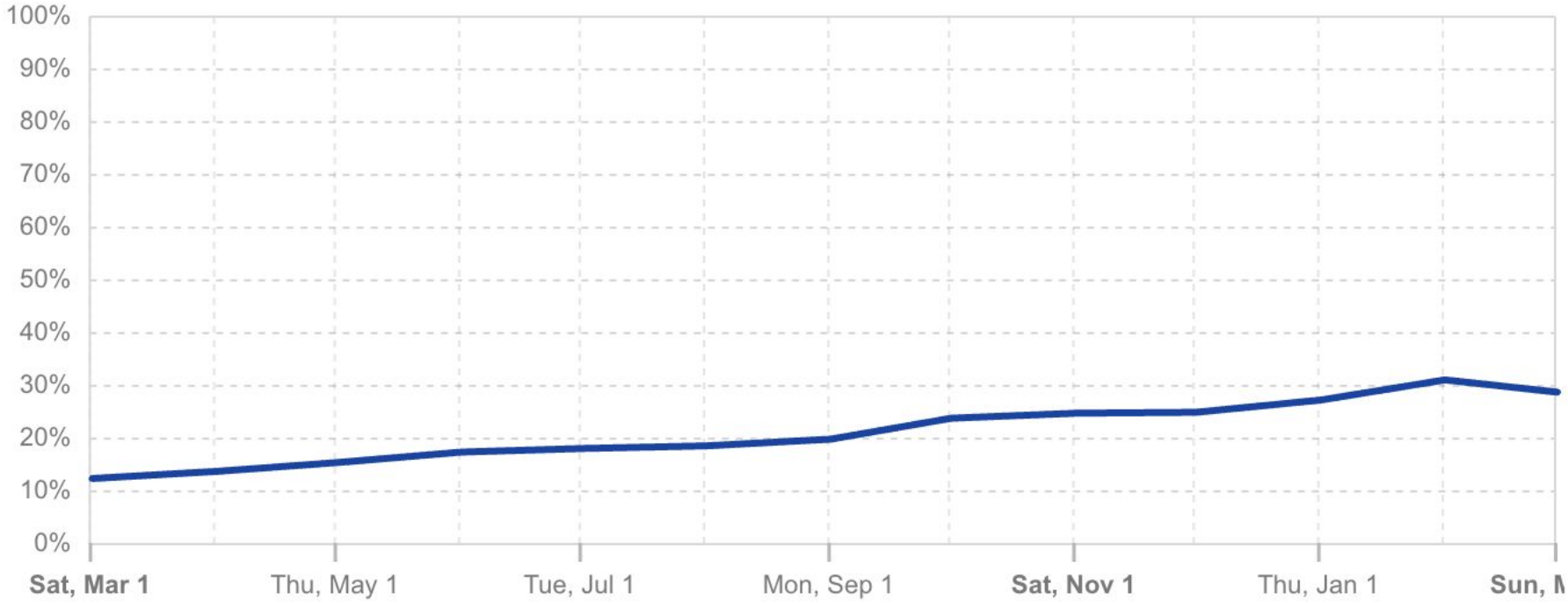
Particularly noteworthy: OpenSSL 3.5.0 and iOS / macOS 26.

These updates take time to roll out…

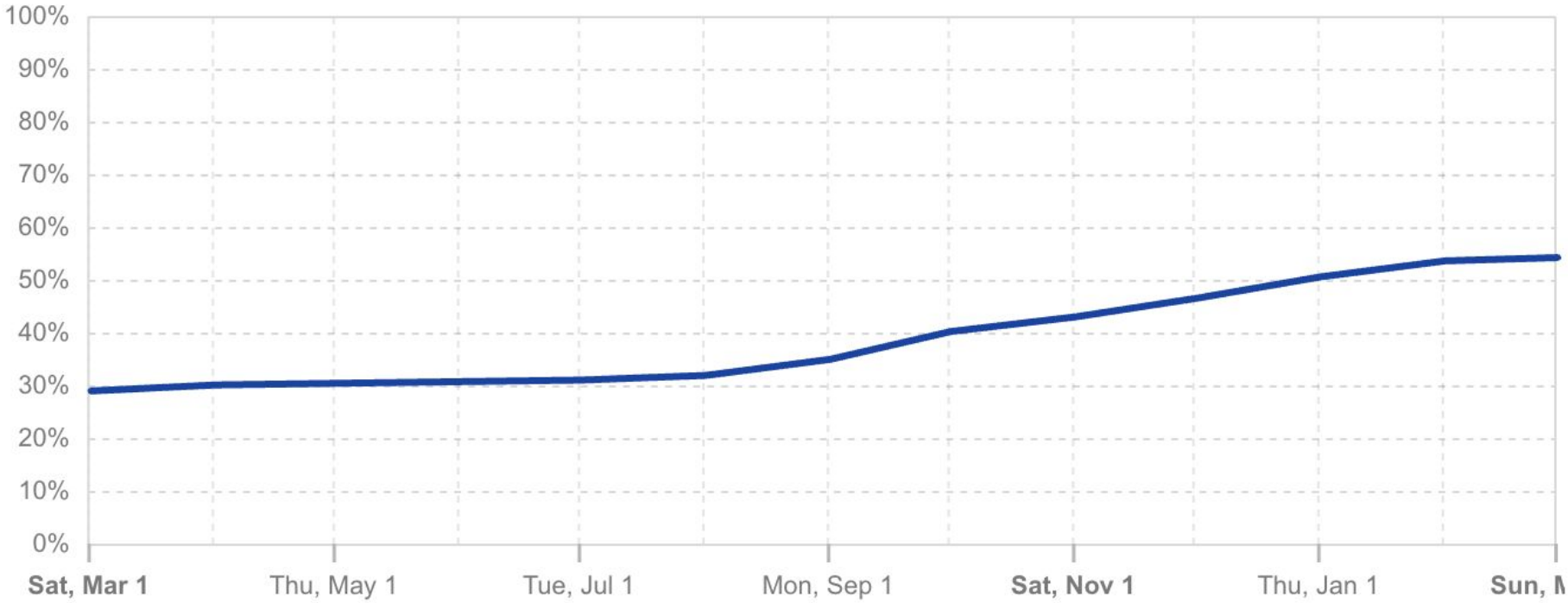
Client support excluding bots



Bots


100%
90%
80%
70%
60%
50%
40%
30%
20%
10%
0%
Sat, Mar 1
Thu, May 1
Tue, Jul 1
Mon, Sep 1
Sat, Nov 1
Thu, Jan 1

Client support overall

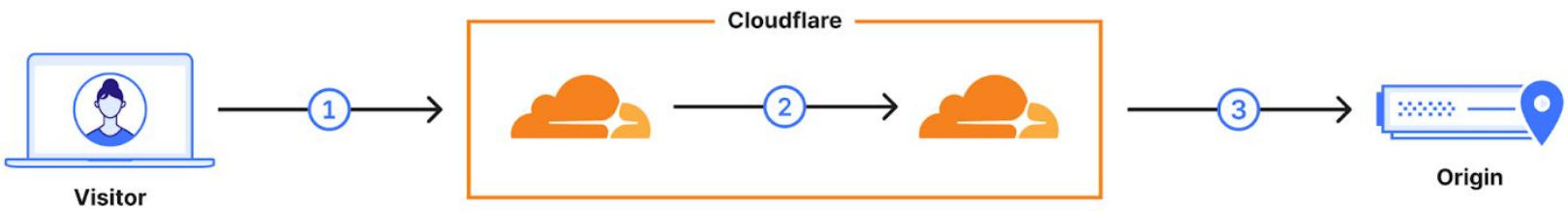


Server support

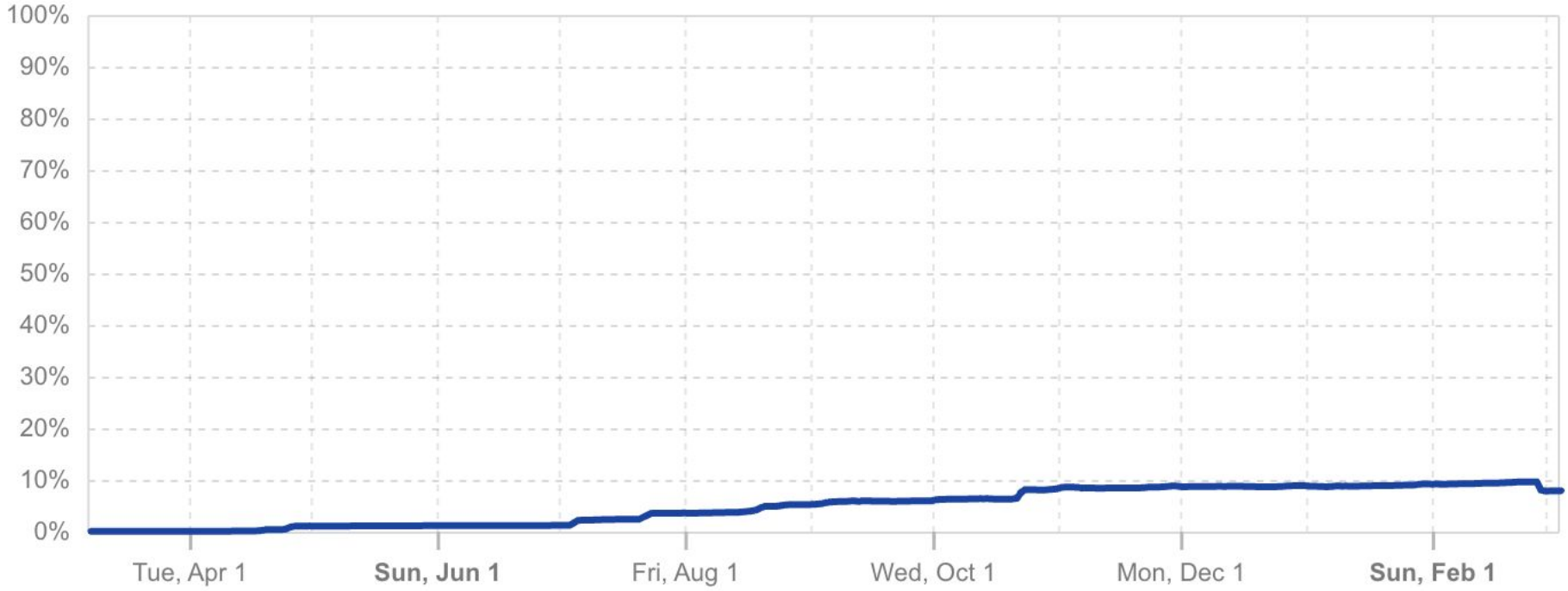
What about the servers?

Jan Schaumann reports 41% of Tranco top 1M websites support post-quantum key agreement in February 2026, up from 28% in March 2025.

But many servers are hidden behind a CDN…


Cloudflare
1
2
3
Visitor
Origin

Support at Cloudflare customer’s origins



This is great progress, but we can only really celebrate once X25519MLKEM768 hits >95%. That will take many years.

What to do today?

- Keep your dependencies up-to-date.
- Check your configuration doesn't accidentally disable X25519MLKEM768: many *best practices* examples do this by accident.

(An ask to TLS library maintainers: application developers shouldn't be asked to configure cryptographic algorithms, but rather just express intent.)

```
# modern configuration
SSLProtocol             -all +TLSv1.3
SSLOpenSSLConfCmd       Curves X25519:prime256v1:secp384r1
SSLHonorCipherOrder     off
SSLSessionTickets       off
```

⚠ Example config that accidentally disables X25519MLKEM768

Post-quantum certificate deployment

(nothing)

Plans for post-quantum certificate support

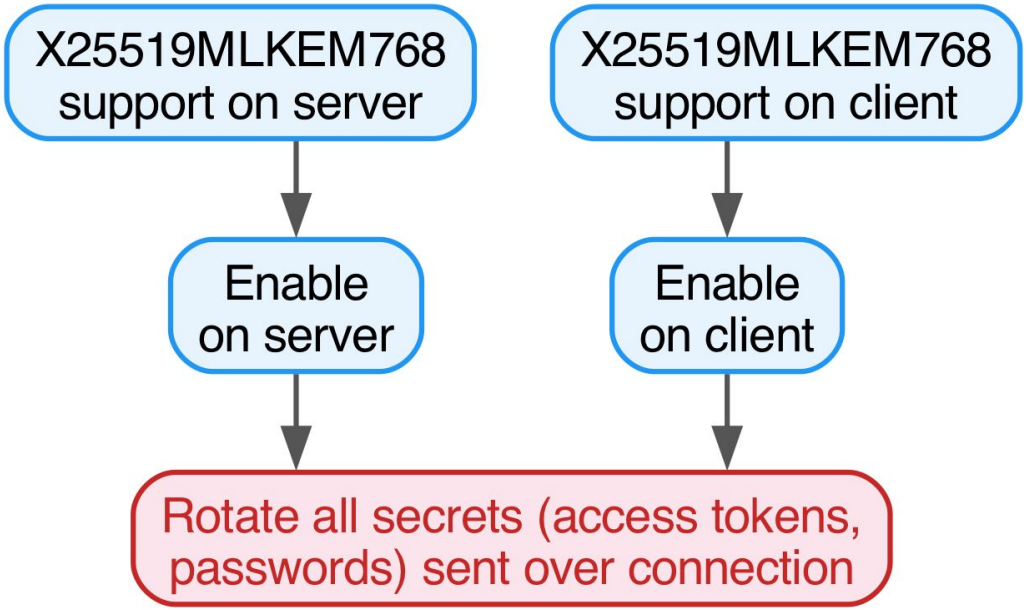
Chrome announced their plan to accept post-quantum Merkle Tree certificates 2027Q1, which are being standardized at IETF.

We're running a feasibility experiment at scale with them as we speak.

Basic client/server support is only a small lift. To gain performance benefit requires more work at client.

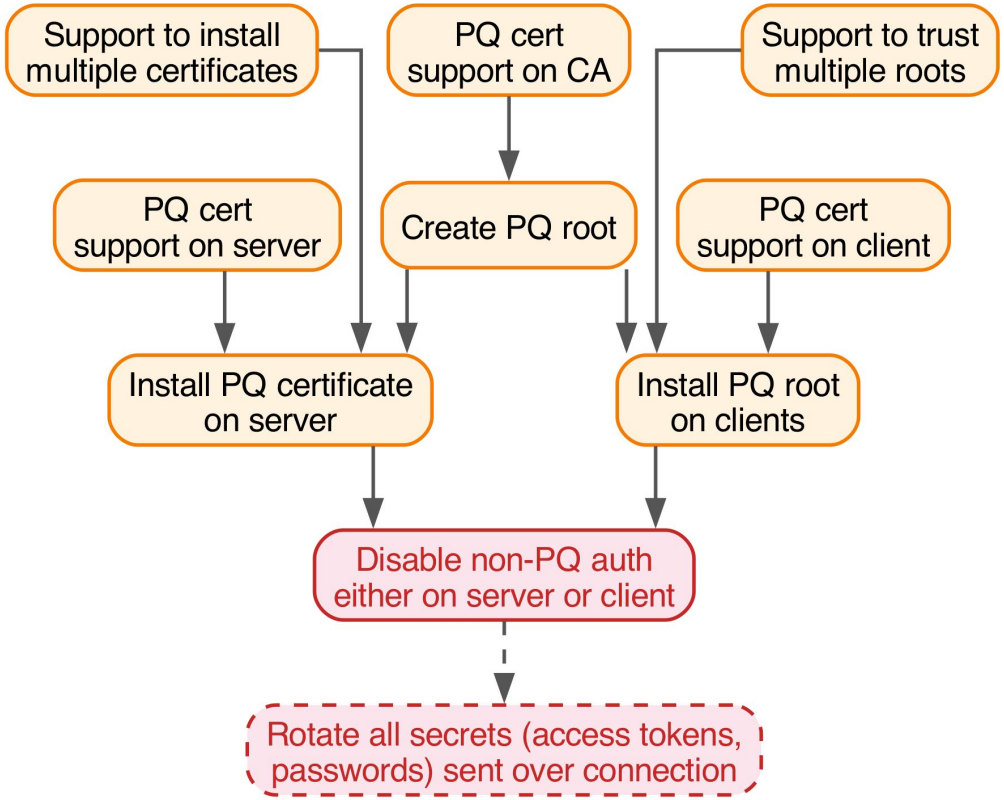
Enough spoilers: check out the RWC talk this Wednesday!

Post-quantum key agreement is “easy”



Not shown: debugging and working around protocol ossification.

Post-quantum certs are a tad more involved

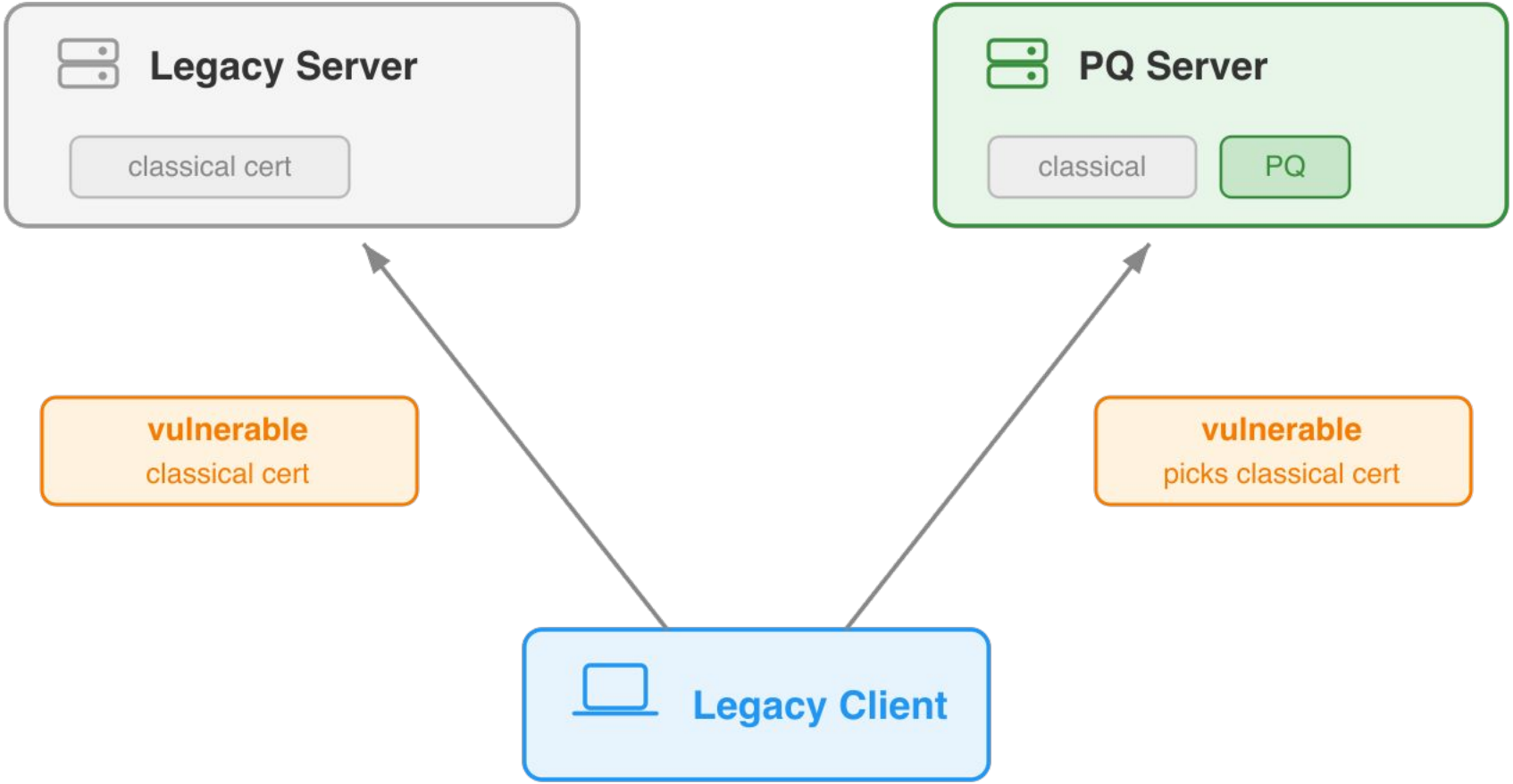


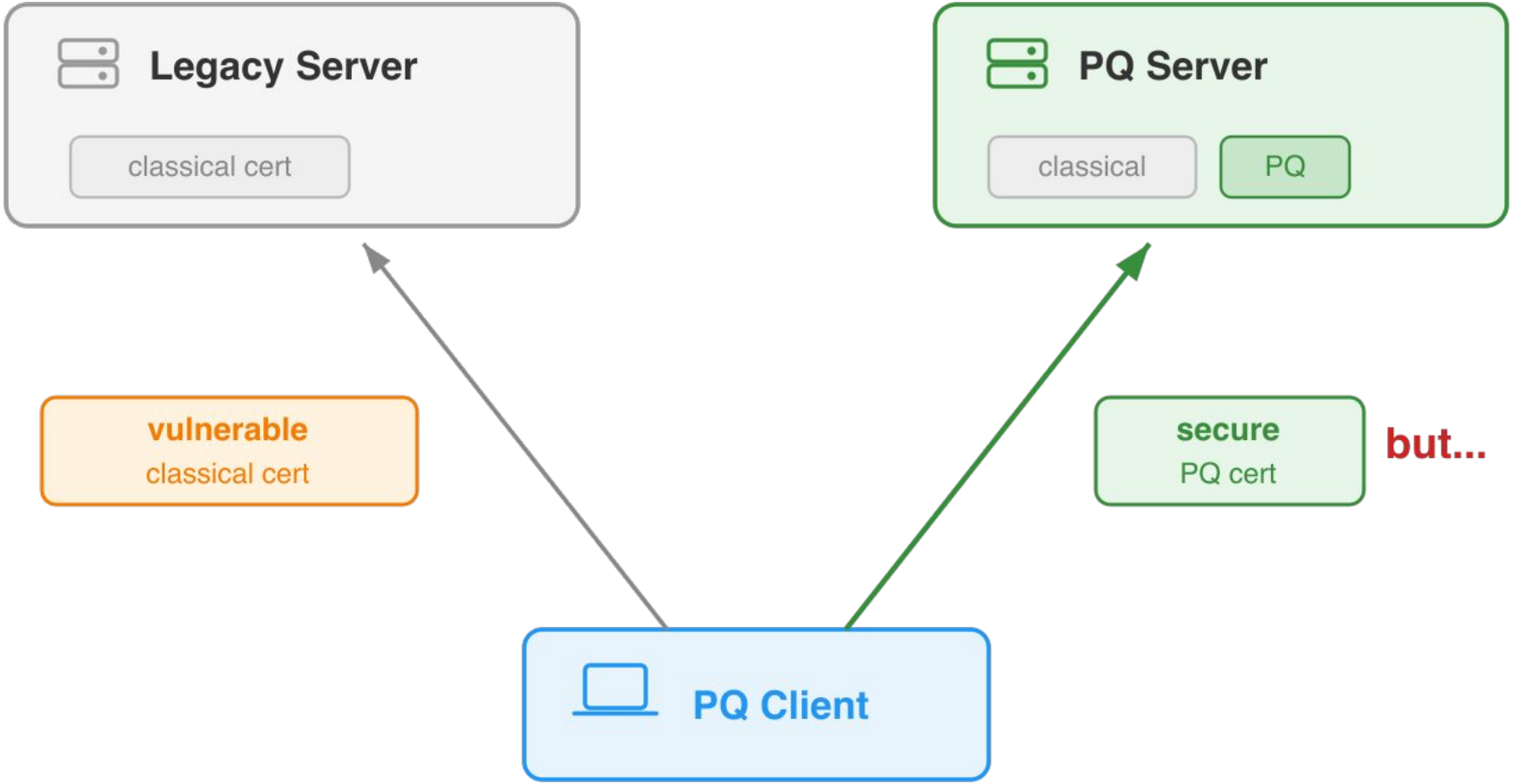
Downgrade

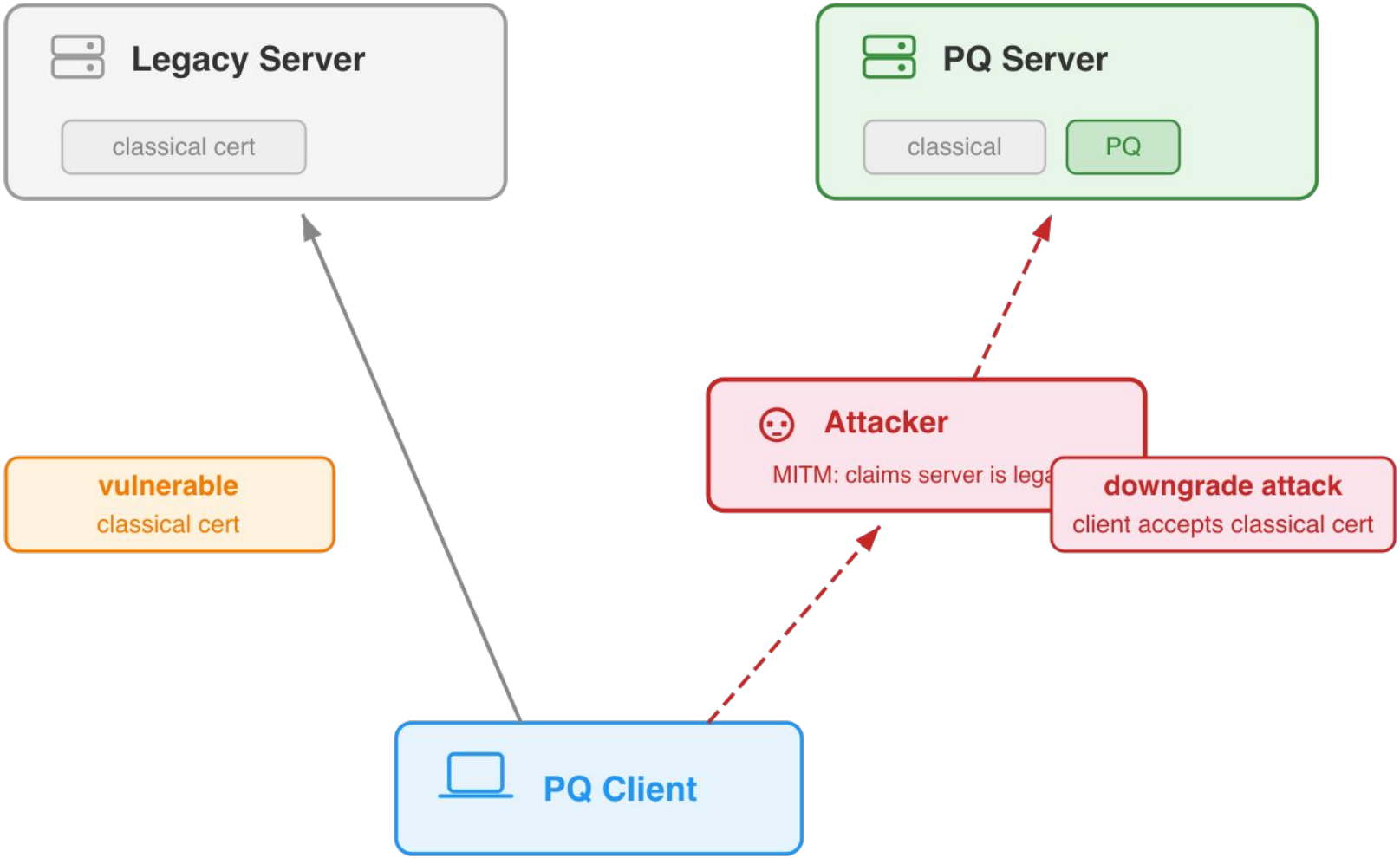
Servers and clients can’t always be updated quickly, and large systems such as the WebPKI need to continue to support quantum-vulnerable certificates:

- A legacy client only trusts a quantum vulnerable cert, but needs to talk to an upgraded server.
- An upgraded client needs to talk to a legacy server that can’t be provisioned with PQ cert.

Supporting the latter allows a downgrade: an attacker can claim the server is legacy to allow the client to accept a non-PQ cert.


Legacy Server
classical cert
PQ Server
classical
PQ
vulnerable
classical cert
vulnerable
picks classical cert
Legacy Client


Legacy Server
classical cert
PQ Server
classical
PQ
vulnerable
classical cert
secure
PQ cert
but...
PQ Client


Legacy Server
classical cert
PQ Server
classical
PQ
Attacker
MITM: claims server is lega
vulnerable
classical cert
downgrade attack
client accepts classical cert
PQ Client

Preventing downgrades with PQ-HSTS

Server can signal to a client that it will have a post-quantum certificate going forward. This is similar to how today HSTS signals that a server will support TLS.

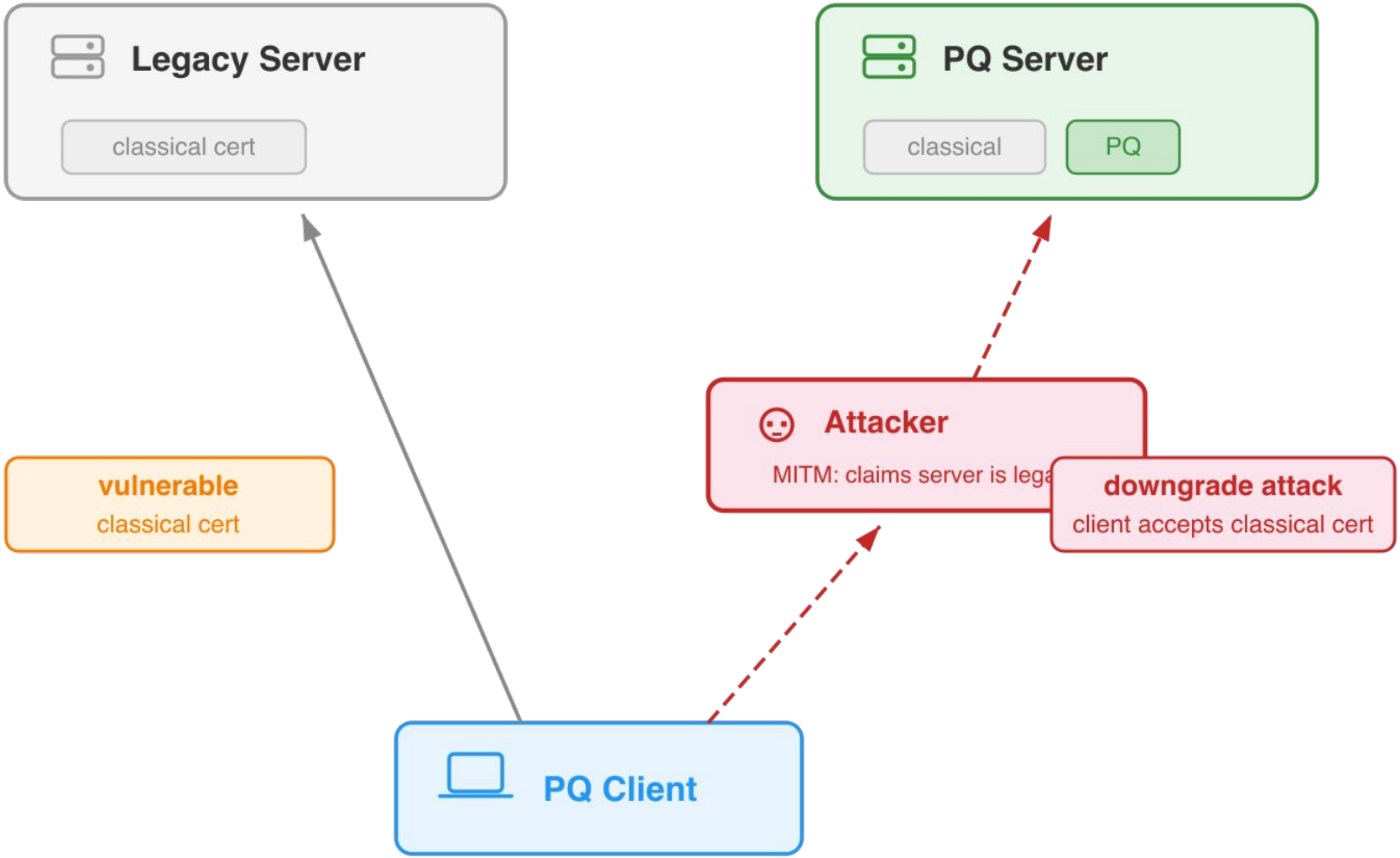
Downside is that this doesn’t work for first connections and can cause surprise breakages when switching to a provider that doesn’t support PQ certs.

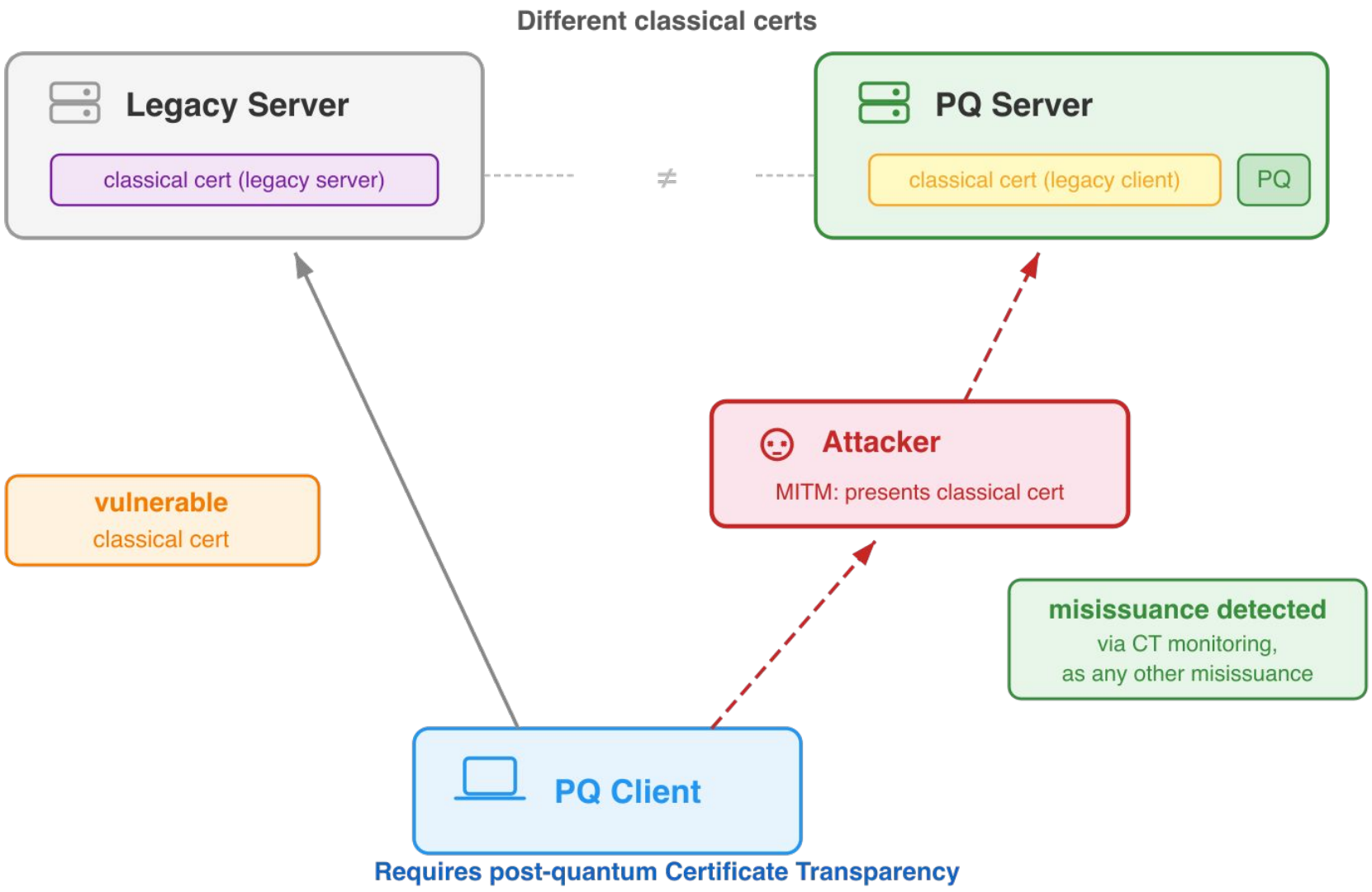
Preventing downgrades using transparency

Key idea: distinguish between the certificate chain for the legacy server and legacy client.

Then we can detect issuance of a *downgrade certificate* for legacy servers, using Certificate Transparency.

This of course requires the transparency to be post-quantum, like with Merkle Tree Certificates. See also Chrome's authentication roadmap.


Legacy Server
classical cert
PQ Server
classical
PQ
Attacker
MITM: claims server is lega
vulnerable
classical cert
downgrade attack
client accepts classical cert
PQ Client


Different classical certs
Legacy Server
classical cert (legacy server)
≠
PQ Server
classical cert (legacy client)
PQ
Attacker
MITM: presents classical cert
vulnerable
classical cert
misissuance detected
via CT monitoring,
as any other misissuance
PQ Client
Requires post-quantum Certificate Transparency

Software support for PQ certs is limited

What can we do today already?

- Update software dependencies.
- Make sure you can install two certificates server-side and CAs client-side: just try RSA and ECC.
- Automate certificate issuance and secret rotation
- If you’re worried about protocol ossification, install a certificate chain with a lot of dummy certificates in the chain to simulate large post-quantum chains.

“We’re lucky, the library we use supports post-quantum certificates!”

Great, which algorithm?

Which algorithm?

- MLDSA44-RSA2048-PSS
- MLDSA44-RSA2048-PKCS15
- MLDSA44-Ed25519
- MLDSA44-ECDSA-P256
- MLDSA65-RSA3072-PSS
- MLDSA65-RSA3072-PKCS15
- MLDSA65-RSA4096-PSS
- MLDSA65-RSA4096-PKCS15
- MLDSA65-ECDSA-P256
- MLDSA65-ECDSA-P384
- MLDSA65-ECDSA-brainpoolP256r1
- MLDSA65-Ed25519
- MLDSA87-ECDSA-P384
- MLDSA87-ECDSA-brainpoolP384r1
- MLDSA87-Ed448
- MLDSA87-RSA3072-PSS
- MLDSA87-RSA4096-PSS
- MLDSA87-ECDSA-P521
- MLDSA44
- MLDSA65
- MLDSA87

- slh-dsa-sha2-128s
- slh-dsa-sha2-128f
- slh-dsa-sha2-192s
- slh-dsa-sha2-192f
- slh-dsa-sha2-256s
- slh-dsa-sha2-256f
- slh-dsa-shake-128s
- slh-dsa-shake-128f
- slh-dsa-shake-192s
- slh-dsa-shake-192f
- slh-dsa-shake-256s
- slh-dsa-shake-256f
- hash-slh-dsa-sha2-128s
- hash-slh-dsa-sha2-128f
- hash-slh-dsa-sha2-192s
- hash-slh-dsa-sha2-192f
- hash-slh-dsa-sha2-256s
- hash-slh-dsa-sha2-256f
- hash-slh-dsa-shake-128s
- hash-slh-dsa-shake-128f
- hash-slh-dsa-shake-192s
- hash-slh-dsa-shake-192f
- hash-slh-dsa-shake-256s
- hash-slh-dsa-shake-256f

My thought process (1)

- MLDSA44-RSA2048-PSS
- MLDSA44-RSA2048-PKCS15
- MLDSA44-Ed25519
- MLDSA44-ECDSA-P256
- MLDSA65-RSA3072-PSS
- MLDSA65-RSA3072-PKCS15
- MLDSA65-RSA4096-PSS
- MLDSA65-RSA4096-PKCS15
- MLDSA65-ECDSA-P256
- MLDSA65-ECDSA-P384
- MLDSA65-ECDSA-brainpoolP256r1
- MLDSA65-Ed25519
- MLDSA87-ECDSA-P384
- MLDSA87-ECDSA-brainpoolP384r1
- MLDSA87-Ed448
- MLDSA87-RSA3072-PSS
- MLDSA87-RSA4096-PSS
- MLDSA87-ECDSA-P521
- MLDSA44
- MLDSA65
- MLDSA87

- ~~slh-dsa-sha2-128s~~
- ~~slh-dsa-sha2-128f~~
- ~~slh-dsa-sha2-192s~~
- ~~slh-dsa-sha2-192f~~
- ~~slh-dsa-sha2-256s~~
- ~~slh-dsa-sha2-256f~~
- ~~slh-dsa-shake-128s~~
- ~~slh-dsa-shake-128f~~
- ~~slh-dsa-shake-192s~~
- ~~slh-dsa-shake-192f~~
- ~~slh-dsa-shake-256s~~
- ~~slh-dsa-shake-256f~~
- ~~hash-slh-dsa-sha2-128s~~
- ~~hash-slh-dsa-sha2-128f~~
- ~~hash-slh-dsa-sha2-192s~~
- ~~hash-slh-dsa-sha2-192f~~
- ~~hash-slh-dsa-sha2-256s~~
- ~~hash-slh-dsa-sha2-256f~~
- ~~hash-slh-dsa-shake-128s~~
- ~~hash-slh-dsa-shake-128f~~
- ~~hash-slh-dsa-shake-192s~~
- ~~hash-slh-dsa-shake-192f~~
- ~~hash-slh-dsa-shake-256s~~
- ~~hash-slh-dsa-shake-256f~~

We need ML-KEM to be secure today, so relying on ML-DSA doesn’t add much of a security assumption.

My thought process (2)

- MLDSA44-RSA2048-PSS
- MLDSA44-RSA2048-PKCS15
- MLDSA44-Ed25519
- MLDSA44-ECDSA-P256
- ~~**MLDSA65-RSA3072-PSS**~~
- ~~**MLDSA65-RSA3072-PKCS15**~~
- ~~**MLDSA65-RSA4096-PSS**~~
- ~~**MLDSA65-RSA4096-PKCS15**~~
- ~~**MLDSA65-ECDSA-P256**~~
- ~~**MLDSA65-ECDSA-P384**~~
- ~~**MLDSA65-ECDSA-brainpoolP256r1**~~
- ~~**MLDSA65-Ed25519**~~
- ~~**MLDSA87-ECDSA-P384**~~
- ~~**MLDSA87-ECDSA-brainpoolP384r1**~~
- ~~**MLDSA87-Ed448**~~
- ~~**MLDSA87-RSA3072-PSS**~~
- ~~**MLDSA87-RSA4096-PSS**~~
- ~~**MLDSA87-ECDSA-P521**~~
- MLDSA44
- ~~**MLDSA65**~~
- ~~**MLDSA87**~~

- ~~slh-dsa-sha2-128s~~
- ~~slh-dsa-sha2-128f~~
- ~~slh-dsa-sha2-192s~~
- ~~slh-dsa-sha2-192f~~
- ~~slh-dsa-sha2-256s~~
- ~~slh-dsa-sha2-256f~~
- ~~slh-dsa-shake-128s~~
- ~~slh-dsa-shake-128f~~
- ~~slh-dsa-shake-192s~~
- ~~slh-dsa-shake-192f~~
- ~~slh-dsa-shake-256s~~
- ~~slh-dsa-shake-256f~~
- ~~hash-slh-dsa-sha2-128s~~
- ~~hash-slh-dsa-sha2-128f~~
- ~~hash-slh-dsa-sha2-192s~~
- ~~hash-slh-dsa-sha2-192f~~
- ~~hash-slh-dsa-sha2-256s~~
- ~~hash-slh-dsa-sha2-256f~~
- ~~hash-slh-dsa-shake-128s~~
- ~~hash-slh-dsa-shake-128f~~
- ~~hash-slh-dsa-shake-192s~~
- ~~hash-slh-dsa-shake-192f~~
- ~~hash-slh-dsa-shake-256s~~
- ~~hash-slh-dsa-shake-256f~~

Ciphertext needs to be secure forever. Certificates can be rolled when they’re weakened. ML-DSA-44 has a very comfortable margin.

My thought process (3)

- ~~**MLDSA44-RSA2048-PSS**~~
- ~~**MLDSA44-RSA2048-PKCS15**~~
- MLDSA44-Ed25519
- MLDSA44-ECDSA-P256
- ~~MLDSA65-RSA3072-PSS~~
- ~~MLDSA65-RSA3072-PKCS15~~
- ~~MLDSA65-RSA4096-PSS~~
- ~~MLDSA65-RSA4096-PKCS15~~
- ~~MLDSA65-ECDSA-P256~~
- ~~MLDSA65-ECDSA-P384~~
- ~~MLDSA65-ECDSA-brainpoolP256r1~~
- ~~MLDSA65-Ed25519~~
- ~~MLDSA87-ECDSA-P384~~
- ~~MLDSA87-ECDSA-brainpoolP384r1~~
- ~~MLDSA87-Ed448~~
- ~~MLDSA87-RSA3072-PSS~~
- ~~MLDSA87-RSA4096-PSS~~
- ~~MLDSA87-ECDSA-P521~~
- MLDSA44
- ~~MLDSA65~~
- ~~MLDSA87~~

- ~~slh-dsa-sha2-128s~~
- ~~slh-dsa-sha2-128f~~
- ~~slh-dsa-sha2-192s~~
- ~~slh-dsa-sha2-192f~~
- ~~slh-dsa-sha2-256s~~
- ~~slh-dsa-sha2-256f~~
- ~~slh-dsa-shake-128s~~
- ~~slh-dsa-shake-128f~~
- ~~slh-dsa-shake-192s~~
- ~~slh-dsa-shake-192f~~
- ~~slh-dsa-shake-256s~~
- ~~slh-dsa-shake-256f~~
- ~~hash-slh-dsa-sha2-128s~~
- ~~hash-slh-dsa-sha2-128f~~
- ~~hash-slh-dsa-sha2-192s~~
- ~~hash-slh-dsa-sha2-192f~~
- ~~hash-slh-dsa-sha2-256s~~
- ~~hash-slh-dsa-sha2-256f~~
- ~~hash-slh-dsa-shake-128s~~
- ~~hash-slh-dsa-shake-128f~~
- ~~hash-slh-dsa-shake-192s~~
- ~~hash-slh-dsa-shake-192f~~
- ~~hash-slh-dsa-shake-256s~~
- ~~hash-slh-dsa-shake-256f~~

RSA's only remaining benefit is fast verification time, which is negated by combining with ML-DSA-44's, which has verification times on the order of EC.

My thought process (4)

- ~~MLDSA44-RSA2048-PSS~~
- ~~MLDSA44-RSA2048-PKCS15~~
- ~~**MLDSA44-Ed25519**~~
- MLDSA44-ECDSA-P256
- ~~MLDSA65-RSA3072-PSS~~
- ~~MLDSA65-RSA3072-PKCS15~~
- ~~MLDSA65-RSA4096-PSS~~
- ~~MLDSA65-RSA4096-PKCS15~~
- ~~MLDSA65-ECDSA-P256~~
- ~~MLDSA65-ECDSA-P384~~
- ~~MLDSA65-ECDSA-brainpoolP256r1~~
- ~~MLDSA65-Ed25519~~
- ~~MLDSA87-ECDSA-P384~~
- ~~MLDSA87-ECDSA-brainpoolP384r1~~
- ~~MLDSA87-Ed448~~
- ~~MLDSA87-RSA3072-PSS~~
- ~~MLDSA87-RSA4096-PSS~~
- ~~MLDSA87-ECDSA-P521~~
- MLDSA44
- ~~MLDSA65~~
- ~~MLDSA87~~

- ~~slh-dsa-sha2-128s~~
- ~~slh-dsa-sha2-128f~~
- ~~slh-dsa-sha2-192s~~
- ~~slh-dsa-sha2-192f~~
- ~~slh-dsa-sha2-256s~~
- ~~slh-dsa-sha2-256f~~
- ~~slh-dsa-shake-128s~~
- ~~slh-dsa-shake-128f~~
- ~~slh-dsa-shake-192s~~
- ~~slh-dsa-shake-192f~~
- ~~slh-dsa-shake-256s~~
- ~~slh-dsa-shake-256f~~
- ~~hash-slh-dsa-sha2-128s~~
- ~~hash-slh-dsa-sha2-128f~~
- ~~hash-slh-dsa-sha2-192s~~
- ~~hash-slh-dsa-sha2-192f~~
- ~~hash-slh-dsa-sha2-256s~~
- ~~hash-slh-dsa-sha2-256f~~
- ~~hash-slh-dsa-shake-128s~~
- ~~hash-slh-dsa-shake-128f~~
- ~~hash-slh-dsa-shake-192s~~
- ~~hash-slh-dsa-shake-192f~~
- ~~hash-slh-dsa-shake-256s~~
- ~~hash-slh-dsa-shake-256f~~

Even though Ed25519 has advantages, it's regrettably not used in the WebPKI today. Let's not impose unnecessary extra work.

My thought process (5)

That leaves MLDSA44-ECDSA-P256 and ML-DSA-44.

I prefer the hybrid, as it can be deployed without *having to convince anyone*.

Unfortunately it is clear there is no consensus that this particular hybrid should be the go-to like X25519MLKEM768 is for key agreement.

It hurts to say, but for maximum compatibility, as it stands, ML-DSA-44 is the safest bet.

Closing thoughts

Thank you, questions?

ask-research@cloudflare.com