

Post-Quantum Authentication for the Web

IAB pqws workshop

David Benjamin
davidben@chromium.org
September 2026

This document discusses a roadmap for post-quantum HTTPS authentication on the Web. It covers:

- The choice of post-quantum certificate construction
- The server capabilities necessary to deploy post-quantum certificates
- The challenges in providing meaningful security from the transition

Post-Quantum Certificates

Chrome has two sets of trust anchors (aka “roots”) for server certificates:

- Trust anchors from the Chrome Root Store
- User- or admin-added trust anchors, often used in enterprise deployments

The first set is shipped with Chrome, with the [Chrome Root Program](#) responsible for adding or removing trust anchors as appropriate to keep the browser secure and functional. Other browsers, OSes, etc., have analogous root programs. These independently-operated but overlapping root programs together are often loosely referred to as “the Web PKI” or “publicly-trusted roots”.

The second set is a configuration option, with either the user, enterprise administrator, etc., taking responsibility for curating these trust anchors. These are often called “private PKIs”, though this is a broad class of non-overlapping PKIs with often vastly different needs.

Chrome Root Store

For the Chrome Root Store, Chrome additionally enforces a transparency requirement, using [Certificate Transparency](#) (CT), which ensures that all accepted certificates are visible in public logs. This system has been crucial to the security of the Web PKI. Without it, we have no oversight on whether CAs are binding names to keys correctly. Cryptographic strength means nothing if one uses the attacker’s key.

This requirement brings some challenges. In the [PLANTS BoF at IETF 124](#), speakers discussed how PQ signature sizes disproportionately impact CT, in both handshake size and log size.

For these reasons, Chrome [plans to](#) deploy [Merkle Tree Certificates](#) (MTCs) for the post-quantum iteration of the Chrome Root Store. MTCs integrate transparency constructions into issuance, in a way that helps mitigate the handshake and log size impacts. The integration also addresses some other issues with CT. See [these PLANTS BoF slides](#) for details.

Although MTCs aim to meet web browsers' needs, they can still coexist with the many non-browser clients that also use Web PKI roots. MTC certificates are still X.509 certificates, just with a new signature algorithm. Basic MTC support can be added to an underlying X.509 implementation similar to [RFC 9881](#) or other new signature algorithms.

Chrome and others in the ecosystem are actively working on MTC deployment, targeting an initial bootstrapping for Q1 2027.

Private PKIs

Unlike the Web PKI, private PKIs do not participate in CT and are managed externally. Some of the above constraints don't apply, and it is ultimately up to the user or administrator to design the PKI according to their needs.

Chrome shipped support for ML-DSA private PKI roots ([RFC 9881](#)) and TLS keys in [June 2026](#). For private PKIs that wish to use MTCs, it's likely Chrome will offer some form of support for user-added MTC roots as well in the future.

Certificate Negotiation

A single TLS service often supports a wide range of clients: up-to-date browsers, older browsers, command-line utilities, mobile apps, un-updatable TV set-top boxes, payment terminals, etc. Services can often comfortably sit in the intersection of all clients. However, that intersection shrinks during transitions that change which CAs are trusted. PQ is a particularly sharp instance of this, replacing the entire trust store. No single certificate will work with both classical-only and post-quantum-only clients.

TLS is an online protocol, so the standard practice is negotiation. A TLS server supports multiple versions of some parameter (protocol version, cipher suite, etc.) and selects the one the client supports on the fly. Likewise, a TLS server can provision multiple certificates and [select between them](#).

On the Web today, certificate negotiation is underdeveloped. Web servers typically configure only one certificate. Even picking between RSA and ECDSA is relatively rare. The one algorithm transition, SHA-1 to SHA-256, was applied on such a long timescale that, by then, clients mostly already supported SHA-256. Though there were exceptions, services largely switched from a single SHA-1-signed certificate to a single SHA-256-signed certificate because there were almost no remaining SHA-1-only clients.

Post-quantum timelines ([Google](#), [Cloudflare](#), [Microsoft](#), [US](#)) are such that we cannot apply a similar strategy. Classical-only clients will continue to exist when post-quantum certificates are deployed. That necessitates building out the missing multi-certificate capabilities in TLS, server configuration pipelines, certificate provisioning, etc., that were not previously strictly required.

The TLS WG is working on the [Trust Anchor IDs](#) extension, which provides the base for robust multi-certificate support. This should set the ecosystem up for not just this PKI transition, but future ones.

MTCs build on Trust Anchor IDs, using them to name entities, negotiate trust anchors, and also negotiate the “landmarks” size optimization. The MTC draft’s aim is that TLS serving software will not need MTC-specific logic, only the generic Trust Anchor IDs machinery.

Downgrade Protection

Ultimately, the goal is to secure connections against a quantum computer. Specifically, we need:

1. **Security:** An active quantum adversary cannot intercept connections between *updated* clients and *updated* services. If we do not achieve this, PQ certs just waste bytes.
2. **Compatibility:** Almost all clients can still talk to almost all services. This target shifts as the client and server ecosystem changes. However, at all times, clients must talk to the services they support, and services must serve the clients they support, wherever they currently are.

Unfortunately, simply *adding* post-quantum certificate support to clients and servers is not sufficient. This is discussed in this [roadmap document](#) and in [SAAG at IETF 126](#) ([slides](#), [video](#)).

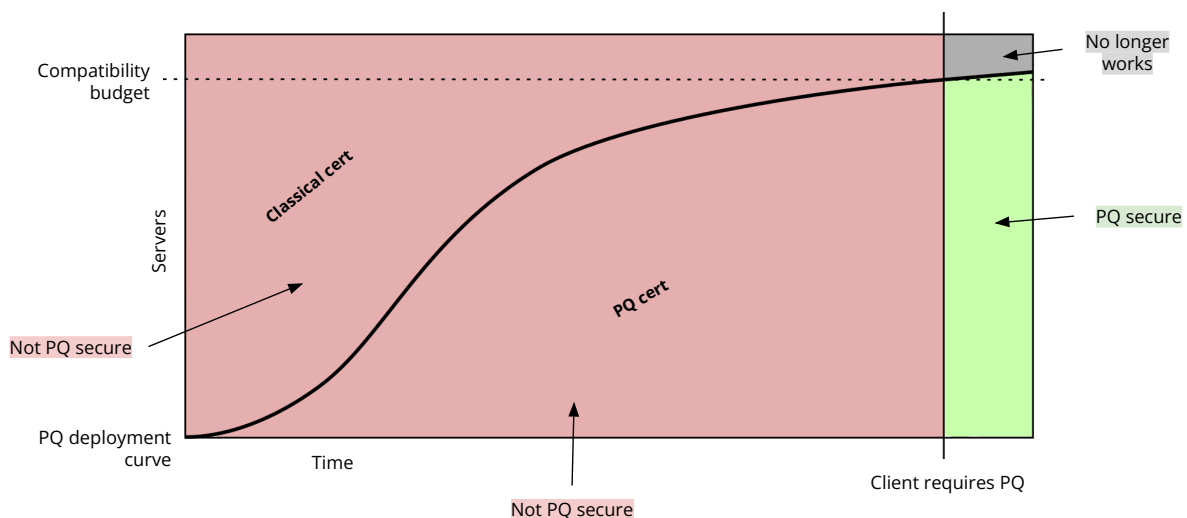
No matter what certificate the legitimate service actually deployed on its servers, the attacker chooses the certificate it presents in an attack. If the client *would have* accepted some certificate under the attacker’s control, it is vulnerable.

	Service <i>only</i> has classical certificate	Service can serve classical or PQ certificate	Service <i>only</i> has PQ certificate
Client <i>only</i> accepts classical certificate	Not PQ-secure	Not PQ-secure	Cannot connect
Client accepts PQ or classical certificate	Not PQ-secure	<i>Not PQ-secure due to downgrade</i>	<i>Not PQ-secure due to downgrade</i>
Client <i>only</i> accepts PQ certificate	Cannot connect	PQ-secure	PQ-secure

Perhaps surprisingly, authentication can be vulnerable even if the service has no classical certificates. **Security for post-quantum authentication depends entirely on client behavior.**

However, the client must hold to the second compatibility constraint. It cannot mandate PQ certificates until almost all of its peer services serve a PQ certificate.

For some clients, this is straightforward. A mobile client for one particular service can simply wait for its corresponding service to transition. But general-purpose HTTPS clients, like browsers, must [remain compatible](#) with a wide, wide range of services. The slowest 0.01% of *all* supported services dominate the timeline for client enforcement. No matter how fast any given service deploys PQC, its security goals are limited by this timeline.



If we do not design the transition with this in mind, we may not have meaningful post-quantum authentication for a decade or more.

Opt-in Individual Services

It is tempting to solve this by making some list of servers with and without PQ keys. For example, such a list could be patterned after [HTTP Strict Transport Security](#) (HSTS), [as discussed here](#).

However, while this has the advantage of being deployable quickly, HSTS-like schemes have significant drawbacks at scale. When PQ services are few, this is fine. As a sizable fraction of the Web deploys PQ certificates and desires meaningful security, this will hit a wall. Any services that cannot deploy “PQ-HSTS” will see no security benefits from their post-quantum certificates.

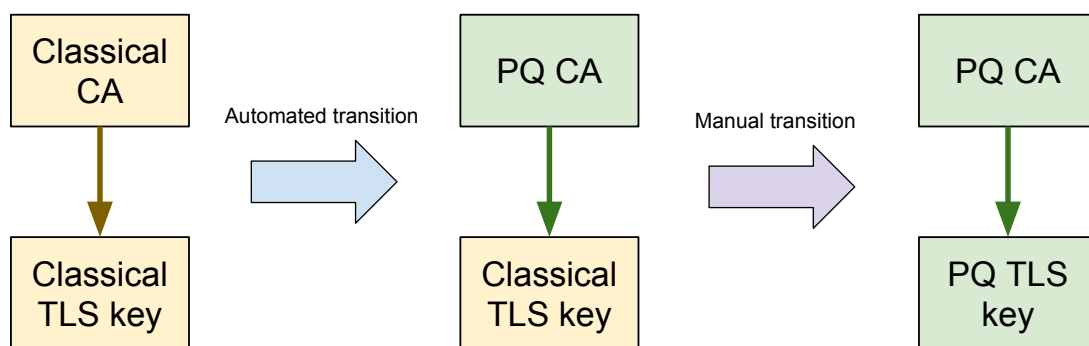
PQ-HSTS thus can only be a temporary stopgap. Any deployment is ultimately a bet that we can remove it *before* more services have transitioned than PQ-HSTS can support. For that, we need a more durable strategy.

Transition CAs First

The above deployment curve is slow because it is limited by manual work from the many, many server operators on the Web:

- Server operators must update TLS server software to support PQ TLS keys
- Server operators must generate PQ TLS keys and manage them alongside RSA/EC keys

The end-entity key is only half of the certificate. The CA signature depends on the CA key. We could look at transition in two steps: switching the CAs to PQ, then switching the TLS keys to PQ.



While this middle state, in itself, does not provide PQ security, it is an achievable and sufficient baseline for downgrade protection:

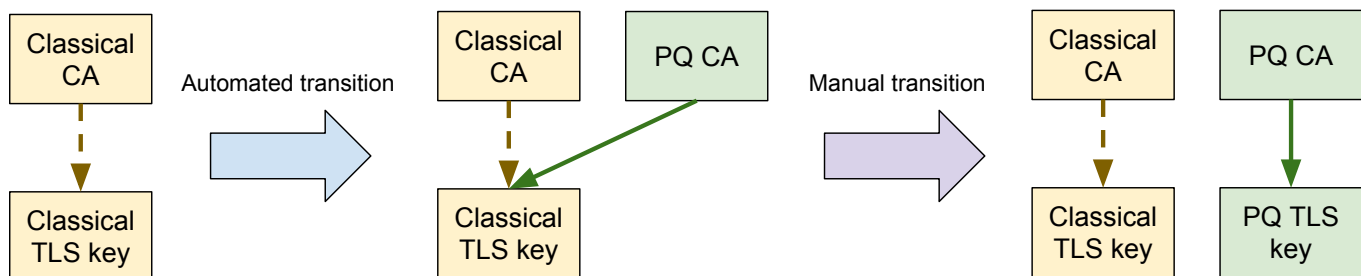
1. There are far fewer CA operators than server operators on the Web. Servers already have to regularly obtain new certificates from their CAs, which is [increasingly automated](#). Designed right, transitions within just the PKI can complete much quicker.
2. Once this quicker transition completes, clients can shift to mandating PQ CAs while still allowing classical TLS keys. By allowing classical TLS keys, legacy services in the middle state remain working. However, an attacker cannot trick the client into trusting a classical TLS key for PQ services in the final state, because it cannot forge certificates from the PQ CA.

The second point is subtle. The services in the middle state are still not PQ-secure. They must eventually take the manual transition into the third state. However, it is crucial to get the whole ecosystem to the middle state quickly, so that *other* services *already* in the final state *can* be secure. That is, the middle state is a “downgrade certificate” for legacy servers. It is a PQ-secure signal that *this* server is not yet PQ-secure. It allows the client to securely distinguish legacy servers from PQ services.

The main transition challenge is then how to make this automated step a reality.

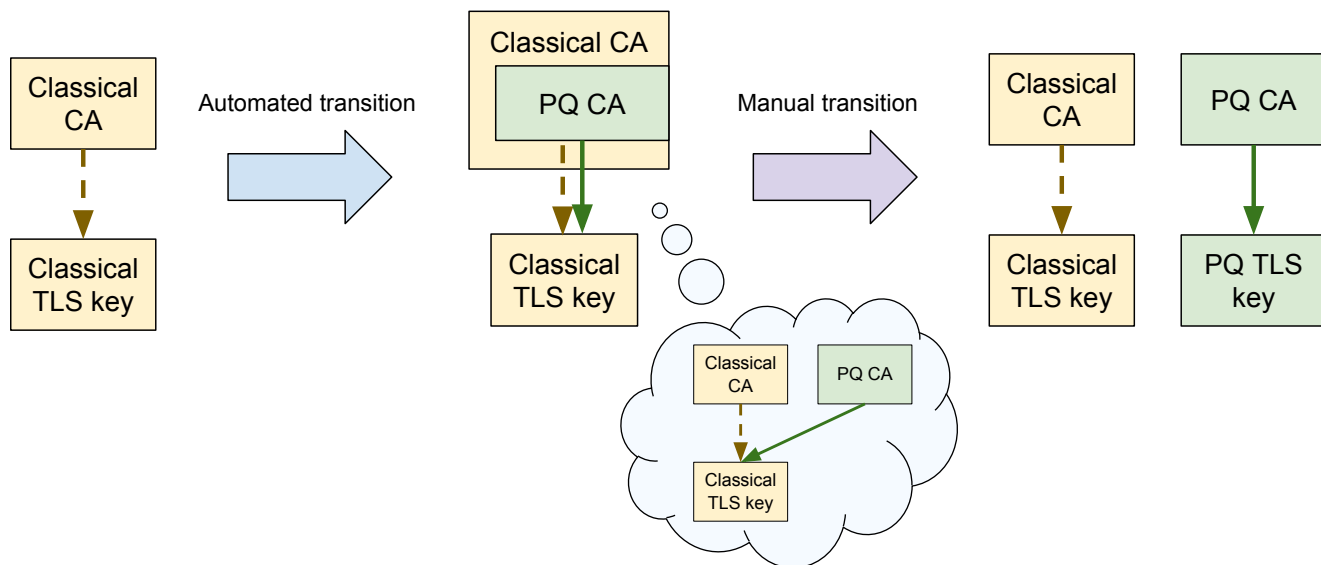
Embedding Downgrade Certificates

Just as clients must account for legacy services while they remain, services must account for legacy clients. Legacy clients will *only* accept a classical TLS key, signed by a classical CA. Including the missing certificate, we have:



This too has a problem. This middle state exists for legacy servers, but not all legacy servers are able to provision and select between certificates. Instead, we need some way to embed the downgrade certificate into the classical certificate, such that:

- A legacy server serves the certificate and passes the embedded information as-is
- A classical client ignores the information and sees a certificate signed by the classical CA
- A PQ client recognizes this construction and reconstructs some signal by the PQ CA
- The two certificates are constrained to have the same SPKI, SANs, etc., so there are no mismatches between what is verified and what is used
- Other constraints around the transition are preserved, e.g. CT log size

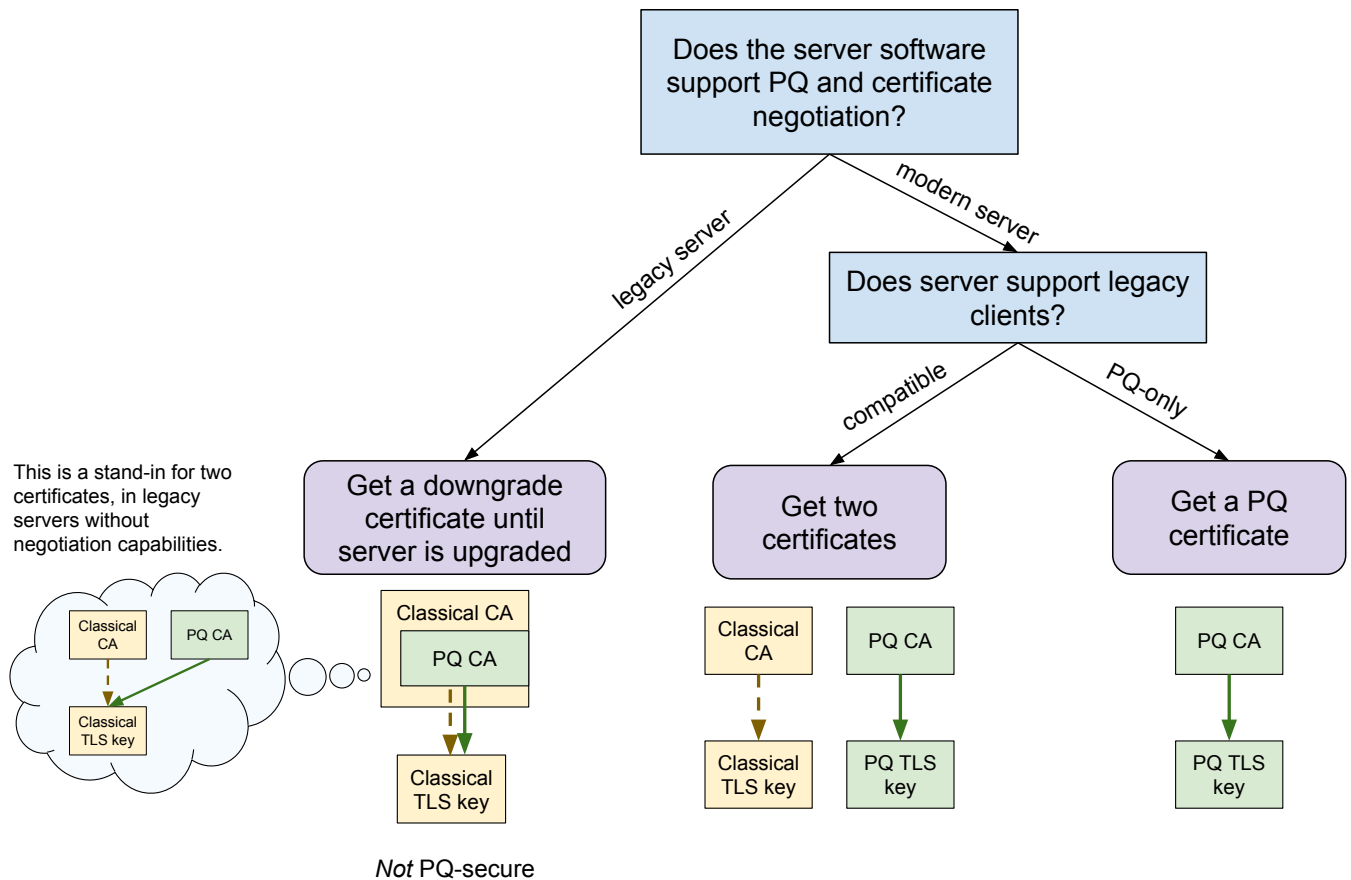


This embedding is *only* a temporary transitional measure for legacy servers. Updated servers are assumed to support multiple certificates. This also means, when the transition is complete, this format can be removed. Subsequent PKI transitions can rely on multi-certificate support directly.

Transition Target

For clients, while we cannot hope to drop classical TLS keys, dropping classical CAs may be within reach. Fortunately, this is sufficient to provide security for PQ servers, while retaining compatibility with legacy servers.

For servers, the target will be this decision tree. Anyone in the leftmost bucket will be encouraged to move to the middle bucket as soon as possible. But, provided they have a downgrade certificate that embeds a signature from a PQ CA, they will not impede the security of PQ-capable servers.



After many years, when every service has moved to either the middle or right bucket, clients can finally drop support for classical TLS keys entirely. This will likely be a *far* future milestone, thus the importance of downgrade protection before then.