

TLS Preflight: smaller, faster and more private handshakes

Dennis Jackson

Mozilla · September 2026

Post-quantum (PQ) cryptography makes TLS handshakes substantially larger [1]. Merkle Tree Certificates (MTCs) [2] shrink the certificate chain. KEMTLS [3] goes further by replacing the handshake signature with a much smaller KEM ciphertext, but requires invasive changes to the TLS state machine, which have proven a barrier to deployment. Meanwhile, Encrypted Client Hello (ECH) [4] would further inflate TLS handshake size if it were naively switched to PQ primitives.

TLS Preflight aims to address the issues blocking deployment of KEMTLS and PQ ECH. It achieves size reductions similar to KEMTLS while being much easier to implement, and brings PQ security to ECH without inflating the handshake. It also lets clients make zero round-trip (0-RTT) connections on their very first contact with a server, which TLS 1.3 cannot do today, cutting latency on first connections.

Design

The server publishes an authenticated, short-lived KEM public key in DNS. Clients use it to establish a pre-shared key (PSK) before connecting, then run an ordinary TLS 1.3 PSK handshake [5].

1. The server obtains an MTC over a PQ KEM public key with a short lifetime, e.g. seven days. It publishes the key, the MTC, and configuration metadata (such as preferred 0-RTT and ECH ciphersuites) in its DNS HTTPS record [6].
2. The client fetches the HTTPS record, as it does for ECH today, and verifies the MTC.
3. The client generates a fresh PSK and encapsulates it to the server's KEM key, much as ECH uses HPKE [7] today.
4. The client sends a normal TLS ClientHello offering the PSK, with the KEM ciphertext as the PSK identity, optionally followed by 0-RTT application data.
5. If the server accepts the PSK, it completes a normal PSK handshake (no Certificate message but still including an ephemeral key exchange for forward secrecy). Otherwise it falls back to a full handshake, exactly as when a session ticket is rejected today.

To support ECH, we define a PSK-based ECH ciphersuite. Instead of carrying a separate ECH KEM ciphertext, the ClientHelloOuter encrypts the ClientHelloInner under a key derived from the offered PSK.

Benefits

Smaller handshakes

Consider a connection using neither 0-RTT nor ECH. The server is authenticated by the MTC the client fetched from DNS and by its ability to decapsulate the PSK, so it sends neither a Certificate nor a CertificateVerify message. Sizes below assume ML-KEM-768 [8] and ML-DSA-44 [9].

Where	What changes	Size
DNS record	MTC over the KEM public key: the key (1 KB) plus an inclusion proof relative to a landmark the client already trusts (1 KB)	+2 KB, cacheable
ClientHello	KEM ciphertext carrying the PSK, alongside the usual PQ key share [10]	+1 KB
Server flight	MTC (2 KB) and PQ handshake signature (2.4 KB) are no longer sent	-4.4 KB

Within the handshake itself the client sends about 1 KB more and the server about 4.4 KB less, so the handshake shrinks by 3.4 KB. Counting the DNS record too, since the MTC has moved rather than

disappeared, the total saving is 1.4 KB. But the DNS record is cacheable and shared across contexts, so for popular domains its cost amortizes to nothing.

0-RTT on the first connection

Today 0-RTT requires a session ticket from a previous connection. With TLS Preflight, a client can send 0-RTT data the first time it ever connects to a server. This helps in three situations.

- **High-latency paths.** Removing round trips before application data flows matters most when RTTs are elevated, for example on mobile networks or through multi-hop VPNs such as Apple Private Relay [11].
- **Cross-site contexts.** Session tickets link connections. If foo.com and bar.com both embed cdn.com, a browser cannot reuse a cdn.com ticket obtained while loading foo.com when it later loads bar.com. A Preflight PSK is derived from a public DNS record and is unlinkable to any previous connection, so browsers can use 0-RTT in fresh contexts.
- **Fresh connection per request.** Privacy architectures such as Oblivious HTTP [12], or Apple's Private Cloud Compute [13], open a new connection for every request to keep requests unlinkable. Each request therefore pays for a full TLS handshake, costing three or more round trips before it can be sent. With TLS Preflight the request and response complete in a single round trip.

ECH with no extra bytes and an authenticated config

- **Size.** Conventional PQ ECH would need its own KEM ciphertext in the ClientHello. With TLS Preflight and the PSK-based ECH ciphersuite, the ClientHelloInner is encrypted under the PSK already offered in the ClientHelloOuter, so PQ ECH adds no overhead to TLS Preflight.
- **Authentication.** ECH configs in DNS are unauthenticated today. With Preflight the config carries an MTC, so the client verifies that the key belongs to the server before encrypting anything to it.

Deployment

TLS Preflight requires few changes.

- **No changes to the TLS state machine.** The client derives a PSK and offers it exactly as it would a resumption PSK. Preflight can even be implemented at the application layer: the application performs the DNS lookup and MTC verification and hands the resulting PSK to an unmodified TLS library.
- **Only a few new pieces need to be standardized.** A way to carry MTCs and the server's 0-RTT configuration in HTTPS records, plus the details of how to derive a suitable PSK from the published KEM public key.
- **Composes with existing designs.** It builds on TLS 1.3 PSK mode and 0-RTT, HTTPS records, MTCs and ECH, and falls back gracefully where sites don't want to use 0-RTT or ECH.

Related Work

Deriving a first-flight key from a published server key is an old idea.

Google's original QUIC crypto protocol [14] from 2013 gave clients 0-RTT against servers whose public key they had previously cached. The design was dropped when QUIC adopted TLS 1.3. The size of PQ primitives, shorter certificate lifetimes and the introduction of ACME for certificate automation [15] now make the design attractive again.

KEMTLS-PDK [16] proposed a similar idea for KEMTLS in 2021: a client that already holds the server's KEM public key encapsulates to it in its first flight, saving bandwidth and a round trip. TLS Preflight is similar in spirit but avoids the handshake changes needed by KEMTLS.

Acknowledgements

With thanks to David Schinazi, Martin Thomson, Bas Westerbaan and Eric Rescorla for their feedback and comments.

References

- [1] B. Westerbaan, “The state of the post-quantum Internet,” *Cloudflare Blog*, Mar. 2024, [Online]. Available: <https://blog.cloudflare.com/pq-2024/>
- [2] D. Benjamin, D. O'Brien, B. Westerbaan, L. Valenta, and F. Valsorda, “Merkle Tree Certificates,” draft-ietf-plants-merkle-tree-certs-05, Jul. 2026. [Online]. Available: <https://datatracker.ietf.org/doc/draft-ietf-plants-merkle-tree-certs/>
- [3] P. Schwabe, D. Stebila, and T. Wiggers, “Post-quantum TLS without handshake signatures,” in *ACM CCS 2020*, 2020. [Online]. Available: <https://eprint.iacr.org/2020/534>
- [4] E. Rescorla, K. Oku, N. Sullivan, and C. A. Wood, “TLS Encrypted Client Hello,” RFC 9849, Mar. 2026. [Online]. Available: <https://www.rfc-editor.org/rfc/rfc9849>
- [5] E. Rescorla, “The Transport Layer Security (TLS) Protocol Version 1.3,” RFC 8446, Aug. 2018. [Online]. Available: <https://www.rfc-editor.org/rfc/rfc8446>
- [6] B. Schwartz, M. Bishop, and E. Nygren, “Service Binding and Parameter Specification via the DNS (SVCB and HTTPS Resource Records),” RFC 9460, Nov. 2023. [Online]. Available: <https://www.rfc-editor.org/rfc/rfc9460>
- [7] R. Barnes, K. Bhargavan, B. Lipp, and C. Wood, “Hybrid Public Key Encryption,” RFC 9180, Feb. 2022. [Online]. Available: <https://www.rfc-editor.org/rfc/rfc9180>
- [8] NIST, “Module-Lattice-Based Key-Encapsulation Mechanism Standard,” FIPS 203, Aug. 2024. [Online]. Available: <https://csrc.nist.gov/pubs/fips/203/final>
- [9] NIST, “Module-Lattice-Based Digital Signature Standard,” FIPS 204, Aug. 2024. [Online]. Available: <https://csrc.nist.gov/pubs/fips/204/final>
- [10] K. Kwiatkowski, P. Kampanakis, B. E. Westerbaan, and D. Stebila, “Post-Quantum Traditional (PQ/T) Hybrid Key Agreement Mechanisms for TLS 1.3,” RFC 10024, Aug. 2026. [Online]. Available: <https://www.rfc-editor.org/rfc/rfc10024>
- [11] Apple, “iCloud Private Relay Overview,” Dec. 2021. [Online]. Available: https://www.apple.com/icloud/docs/iCloud_Private_Relay_Overview_Dec2021.pdf
- [12] M. Thomson and C. A. Wood, “Oblivious HTTP,” RFC 9458, Jan. 2024. [Online]. Available: <https://www.rfc-editor.org/rfc/rfc9458>
- [13] Apple Security Engineering and Architecture, “Private Cloud Compute: A new frontier for AI privacy in the cloud,” *Apple Security Research Blog*, Jun. 2024, [Online]. Available: <https://security.apple.com/blog/private-cloud-compute/>
- [14] J. Roskind, “QUIC: Design Document and Specification Rationale,” 2013. [Online]. Available: https://docs.google.com/document/d/1RNHkx_VvKWyWg6Lr8SZ-saqsQx7rFV-ev2jRFUoVD34/edit
- [15] R. Barnes, J. Hoffman-Andrews, D. McCarney, and J. Kasten, “Automatic Certificate Management Environment (ACME),” RFC 8555, Mar. 2019. [Online]. Available: <https://www.rfc-editor.org/rfc/rfc8555>
- [16] P. Schwabe, D. Stebila, and T. Wiggers, “More efficient post-quantum KEMTLS with pre-distributed public keys,” in *ESORICS 2021*, 2021. [Online]. Available: <https://eprint.iacr.org/2021/779>