RFC 10024: Negotiating Post-Quantum TLS handshakes

Kris Kwiatkowski

No Institute Given

I co-authored RFC 10024 and ran the server side of the 2019 Cloudflare-Google post-quantum experiment. RFC 10024 is nine pages long. If you strip the boilerplate, the actual specification - three code points for PQ/T hybrid key agreements - fits on about four of them. It says things like: concatenate the ML-KEM-768 encapsulation key and the X25519 share; the result is 1216 bytes. That is roughly the technical depth of the whole document.

And it matters more than the browser padlock suggests. TLS now secures machine-to-machine traffic too - API calls, mail delivery, VPNs, database links, even a server's link to its own disks, as of this year's NVMe 2.4 spec. This is infrastructure, not a browser feature.

As the IETF community, we spent several years on those four pages.

1 The part that was not cryptography

The easy part was the cryptography - the hard math was done by other people, elsewhere. NIST ran an eight-year competition and produced ML-KEM. Academia produced the security proofs for hybrid key exchange. The framework document RFC 9954 worked out how to feed two shared secrets into the TLS 1.3 key schedule so that the session stays secure if either component survives.

What was left for us was, on paper, trivial: pick the combinations, pick the byte order, pick the code points. A competent engineer could write that spec in an afternoon.

That afternoon took a few years. Let me explain why that is not a failure of the process, but the entire point of it.

Standards work looks like argument and is mostly constraint discovery. The years went to someone, somewhere on a mailing list, saying: that will not work here, and here is why - and being right.

1.1 The other migration, and why this one is different

We have run one comparable migration before: HTTP to HTTPS - and the contrast is stark.

HTTPS took a quarter of a century, and it was loud the whole way. It needed certificates, which meant money, which meant Let's Encrypt had to be invented to

make them free. It needed every site operator to act, browsers to shame stragglers with “Not secure” warnings, and users trained to look for a padlock and then untrained when it turned out to mislead them. The migration succeeded, but the hard way: by making the old thing socially and commercially untenable.

The post-quantum migration inverts all of that. No site operator has to buy anything, configure anything, or even know it happened; two endpoints that both speak a hybrid group simply use it, and everyone else upgrades whenever their software does. HTTPS was a migration of people and incentives. This is a migration of code paths - and code paths migrate far faster than incentives.

The numbers show it. HTTPS took two decades to reach half of web page loads, around 2017, and only now nears saturation. The hybrid key exchange went from zero to majority in about two years: Chrome default in April 2024, Firefox that November, Apple in late 2025, and Cloudflare past 71% of human web traffic by August 2026. AWS moved in parallel, making ML-KEM hybrids the default across KMS, ACM, Secrets Manager, and S3 in April 2026 and retiring the pre-standard Kyber code points on schedule. Twenty years versus two, for a change that is cryptographically far more radical.

That majority is measured at the edge, though, and the edge is the easy half. Browsers and CDNs are a handful of codebases with fast release cycles, and when they moved, most of the traffic moved with them. Behind them, connections from a CDN to the origin it fronts run near 12% (Cloudflare Radar) - a gap of six to one against the numbers above. None of that is a decision anyone is refusing to make; it is a server stack that has not shipped PQC yet, or has shipped it in a version nobody has deployed. That backlog is emptying now that OpenSSL 3.5 carries the hybrids by default, and the pattern is worth naming because the authentication migration will run into the same asymmetry from the start.

That speed was inherited, and the inheritance has a precise address: these groups are defined for TLS 1.3 and nothing below it. It was deliberate, and a recurring argument in the working group - deployments asked for a TLS 1.2 variant, and the answer stayed no. TLS 1.2 would have needed invasive surgery to host a KEM safely, and every hour retrofitting it would have extended the life of a protocol the group wants retired. TLS 1.3 already had the machinery (`supported_groups`, `key_share`), a key schedule the hybrid analysis could build on, and deployment majority, because the 1.2-to-1.3 migration had itself quietly succeeded a few years earlier. Quantum resistance became the carrot for finishing that one: if you want PQ, the road runs through 1.3.

So when this migration looks effortless, it is because it stands on an older one that was anything but. Boring deployments are compound interest on earlier pain.

1.2 Where it actually started

The origin of this RFC is not 2019, and not lattices. Google's CECPQ experiments had already glued X25519 to NewHope in 2016 (CECPQ1), then to NTRU-HRSS (CECPQ2) and, in a sibling branch, to SIDH-based SIKE (CECPQ2b). In 2019 the second round ran client-side in Chrome Canary, where Google had implemented it, and server-side at Cloudflare, where I ran it: both hybrids, against real users and real networks, in the first large-scale A/B test of post-quantum cryptography on the public Internet.

It was a contest between two failure modes. NTRU-HRSS is fast but big - its public key is around 1100 bytes and pushes the ClientHello past a kilobyte. SIKE was the opposite: tiny keys, around 330 bytes, comfortably inside one packet, but two orders of magnitude more CPU. Nick Sullivan's framing at NIST's second PQC conference stuck: HRSS the ostrich, big and fast; SIKE the turkey, small and slow. If the network punished size, the turkey won; if computation dominated, the ostrich did. The network answered clearly - the latency cost of the extra bytes was modest and shrinking, while SIKE's arithmetic was a fixed tax on every handshake. Lattices won, and everything since - Kyber, ML-KEM, this RFC - follows that fork.

The experiment also surfaced the failure mode that would haunt the next five years: a measurable sliver of connections died not from cryptography but because middleboxes and embedded stacks had hard-coded the assumption that a ClientHello fits in one packet - the kind of hardware that gets replaced when it physically catches fire, not when a spec changes. Post-quantum shares broke that assumption forever, and you only learn the per-vendor prevalence of that breakage by shipping.

Then, in 2022, the punchline: Castryck and Decru broke SIKE completely - classical computer, about an hour on one core, after years of NIST scrutiny. Every CECPQ2b session ever served suddenly had a dead post-quantum half - and every one was still confidential, because X25519 had been standing next to it the whole time. The failure case wasn't hypothetical; it had already run in production, by accident, in advance. Hybrid is why nobody was hurt.

1.3 What the long threads were actually about

If you have never followed an IETF working group, the mailing-list archives of the TLS WG and pqc-forum are an education. Threads with hundreds of messages, resurrected three times under different subject lines, where the same two people make the same two arguments for six weeks while everyone else learns to skim.

From the outside this looks like dysfunction. From the inside, the threads *are* the engineering: every message is a stakeholder revealing a constraint you did not know existed. A browser vendor wants to ship exactly one hybrid group, so it had better be the right one. An HSM vendor's certification cycle is measured in years, not releases. A bank's "just turn it on" collides with a compliance regime

written before Shor's algorithm had a Wikipedia page. And a CDN points out that key-exchange cost is not academic: an extra kilobyte and a few hundred microseconds per handshake, times billions of connections a day, is real CPU, real bandwidth, and a real line on a real bill - the difference between a group that ships and one that stays a checkbox.

None of that is in any paper. All of it is load-bearing.

1.4 My favorite example: the byte order that encodes policy

One detail in the RFC looks like a typo and is not. In X25519MLKEM768 the shares are concatenated ML-KEM first, X25519 second - the reverse of the name, and of the convention in RFC 9954. The document calls this, dryly, "historical reasons."

The history is the interesting part. FIPS key derivation, as specified today in SP 800-56Cr2, approves combining two concatenated secrets only if the *first* one comes from an approved algorithm - so the position of 32 bytes inside a buffer decides which implementation has to be FIPS-certified. Put ML-KEM first and an X25519 library that will never be certified can ride along in a compliant deployment; for the P-256 and P-384 groups the order flips, because there the ECDH side is the certified one. There is no cryptographic content in any of it - both orders are equally secure - the constraint is entirely regulatory.

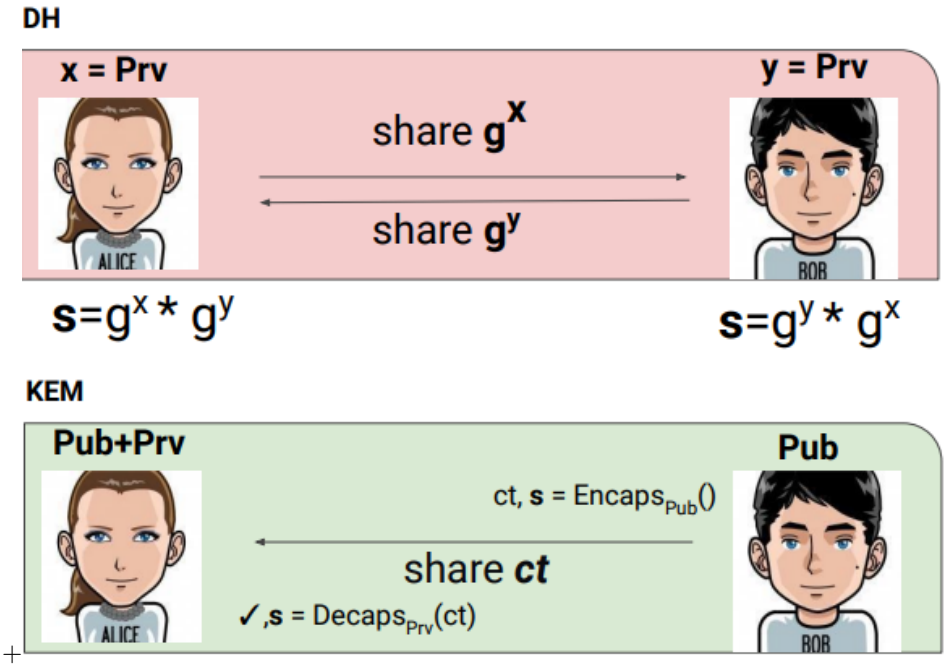
And the regulation is already moving: NIST has signaled the ordering requirement will be dropped in the next revision of SP 800-56C, and the newer SP 800-227 guidance already assumes it. But the byte order can't follow. By the time the rule started to change, X25519MLKEM768 was already deployed across browsers and CDNs, and the concatenation order is part of the wire format - change it and every existing client and server computes a different shared secret and the handshake fails. So the order is frozen: pinned to a rule that is on its way out, kept in place by the very deployment success it enabled. A fossil, formed in real time and now load-bearing. That is standards work in one artifact - a byte order as a snapshot of regulatory reality at the moment it ossified.

2 The technical part - how it actually works

Everything above was about how the decisions got made. This part is the mechanism itself: what goes on the wire, why both halves are there, what happens when the peer is unknown, and the two lines of config that turn it on.

2.1 Both parts are load-bearing

It is worth being precise about why the hybrid needs both halves, because the two components are not redundant copies of the same guarantee - they fail in disjoint ways.

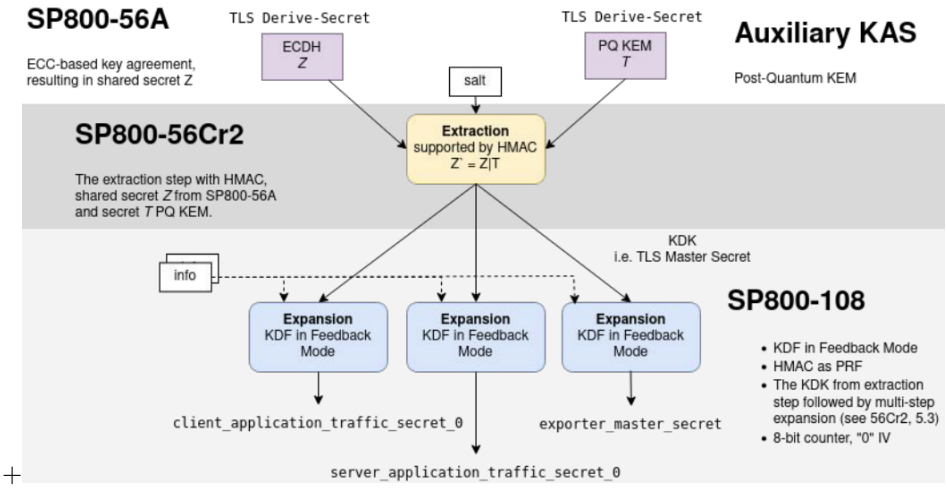


But first, a subtlety that trips people up: a KEM is not a Diffie-Hellman, and the difference shapes the whole wire format. DH is symmetric - both sides do the same thing, exchange g^x and g^y, and multiply. A KEM is asymmetric: one side generates a keypair and sends the public (encapsulation) key; the other side runs a randomized encapsulation against it, producing a ciphertext and the shared secret; the first side then runs a deterministic decapsulation, which needs both its keys. The two peers perform different operations, so ML-KEM cannot be dropped into TLS's ECDH slot as if it were another curve - roles had to be assigned. The client generates the keypair; the server encapsulates against it in its reply - which keeps the whole exchange inside the one round trip TLS 1.3 already uses. In exchange, ML-KEM brings something DH never had at this layer: IND-CCA2 security, meaning the shared secret stays indistinguishable from random against an active attacker who can even submit arbitrary ciphertexts for decapsulation.

There is a related point the RFC does not dwell on but it is worth knowing. TLS 1.3 uses each KEM key exactly once and throws it away, so an attacker gets at most a single decapsulation query per key - which means the full IND-CCA2 machinery is technically stronger than TLS strictly needs; a weaker notion, IND-1-CCA (security under one decapsulation query), provably suffices for the handshake. That matters because ML-KEM buys its IND-CCA2 security with the Fujisaki-Okamoto transform, whose decapsulation re-encrypts the message to check it - real cost, and a real side-channel surface. Constructions that target only IND-1-CCA can skip the re-encryption and decapsulate meaningfully faster. The ecosystem shipped the conservative choice - ML-KEM as standardized, FO and all - because interoperability and a single vetted primitive beat a cleverer

one, but it is a nice illustration that "use the strongest security notion available" and "use the notion the protocol needs" are not always the same sentence.

The key establishment starts with the client sending both PQ/T shares in one `key_share` entry; the server sends back an ML-KEM ciphertext plus its own ECDH share; each side ends up with two shared secrets, which are concatenated and dropped into the slot of the TLS 1.3 key schedule where the plain ECDH secret used to go. HKDF-Extract mixes the concatenation, and because the full transcript - including both public shares - is bound into the schedule, the analysis in RFC 9954 shows the session keys are secure as long as *either* component secret is indistinguishable from random. A TLS v1.3 key schedule already provides the necessary combiner. Formally, HKDF-Extract over SHA-2 can be modelled as a dual-PRF combiner - pseudorandom output as long as either input key is good, even if the other is maliciously chosen - following Bellare-Lysyanskaya and the hybrid KEM analysis of Bindel et al. Those security analyses assume canonical fixed-length encodings. And it is why the RFC warns you cannot copy this construction into another protocol: the proof leans on TLS 1.3's transcript hashing, not on concatenation being magic.



Threat model is asymmetric. X25519 has survived twenty years of cryptanalysis but dies completely and retroactively the day a cryptographically relevant quantum computer exists: a passive adversary recording traffic today runs Shor's algorithm later and recovers the scalar. ML-KEM closes that hole, but it earns less long-term confidence than the curve on two independent axes. The hardness assumption is well-aged - lattice problems go back to the 1990s and LWE to 2005 - but the *scheme* is young: ML-KEM's specific construction and parameters have far fewer cryptanalyst-years on them than elliptic curves have absorbed since the 1980s, and post-quantum estimates still carry real error bars. And the implementations are young, which is the part people underrate.

The RFC's security considerations spell out one subtle example: ML-KEM's encapsulation randomness m is encrypted to the peer and recovered exactly during decapsulation, so it is a direct window into your RNG's output stream - a Dual-EC style RNG compromise gets a fresh oracle that traditional ECDH never exposed, since ephemeral scalars are never sent anywhere. Hence the mandated hygiene: the modulus check on encapsulation keys from FIPS 203, strict ciphertext length validation, the all-zero output check on X25519, and careful separation of decapsulation failures from malformed inputs.

So the curve insures against the newness of the lattice, and the lattice insures against the physics of the curve. Drop either half and you are betting the confidentiality of recorded traffic on exactly one conjecture holding. Keep both and an attacker needs a quantum computer AND a lattice break.

2.2 What it looks like on the wire

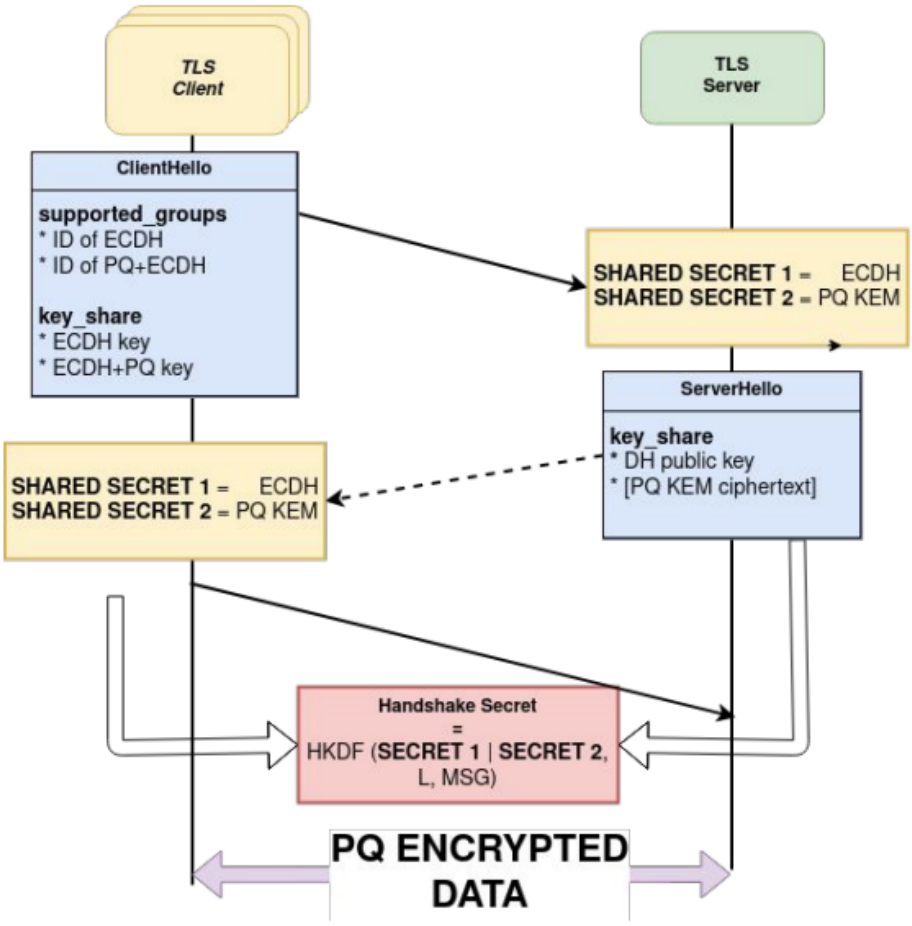


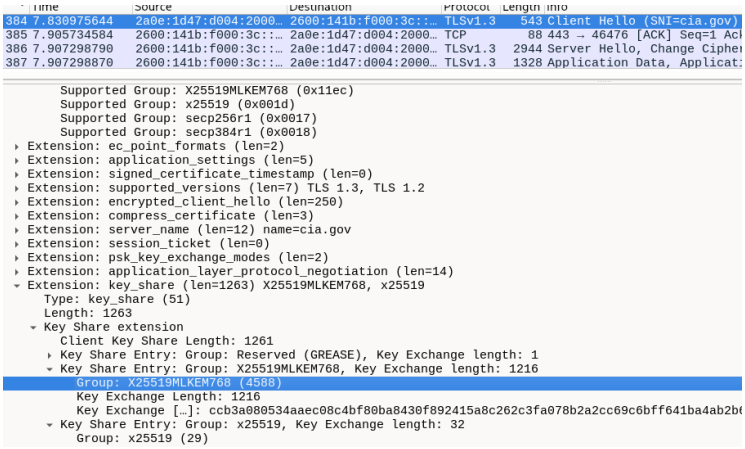
Figure 1: The one-round-trip TLS 1.3 handshake with a hybrid key_share.

If you open a capture in Wireshark, the migration is visible as a weight change. A classical `ClientHello` is a few hundred bytes. A hybrid one carries, inside the `key_share` extension, an opaque 1216-byte blob for group 0x11EC (X25519MLKEM768) - the ML-KEM-768 encapsulation key with the X25519 point appended - next to whatever traditional shares the client still offers. The server answers with its own 1120-byte `key_share`: the ML-KEM ciphertext plus its X25519 point.

The size is not an accident. ML-KEM-768 was the deliberate choice over ML-KEM-1024 precisely so that a ClientHello carrying it still has a chance to fit

inside a single packet - under the typical ~1500-byte Ethernet MTU - as long as the client is not also stacking many other extensions. Over QUIC the budget is tighter: 1216 bytes pushes the ClientHello past a single Initial, so it spans two packets.

Everything after the hellos - certificates, Finished, application data - looks byte-for-byte like any other TLS 1.3 connection, because past the key schedule the hybrid has already dissolved into ordinary traffic secrets.



2.3 The quiet workhorse: HelloRetryRequest

The negotiation itself reuses TLS 1.3 machinery unchanged, and the underrated piece of it is HelloRetryRequest. TLS 1.3 clients predict: they send key shares only for the groups they guess the server will pick. Guess right and the handshake completes in one round trip. Guess wrong - the server supports a hybrid group you advertised in `supported_groups` but did not include a share for - and the server sends HRR asking for it, costing a full extra round trip while the client resends a ClientHello with the right share.

That one mechanism is what made incremental deployment possible, and it shows up on both sides of the tradeoff. A client that is unsure whether servers speak hybrid yet can advertise it cheaply and let HRR upgrade the connection - no share sent, no wasted 1216 bytes, but an RTT penalty when the upgrade happens. That is precisely how Cloudflare first rolled out post-quantum to customer origin servers: advertise only, let willing origins HRR their way up, break nothing. The cost was real - the retry cancels out the round trip TLS 1.3 had saved over 1.2 - which is why the endgame, shipped in 2026, is prediction: remember or scan what the peer prefers and put the hybrid share in the first flight. HRR is the safety net that lets prediction be aggressive. The two together are why this migration needed no version bump and no flag day: an unknown peer costs you a round trip, never a connection.

Two small experiments from my own bench are worth recording here, because they are the kind of thing no spec tells you. First, the *order* of key shares in the ClientHello is a policy knob in disguise: many server stacks simply take the first offered share they support, so a client that lists the hybrid share before X25519 has effectively opted the whole connection into post-quantum, while the reverse order quietly opts out - same bytes, same groups, different outcome. Second, sending both a hybrid and a traditional share "just in case" costs you the full duplication - roughly 1.2 KB of keyshare the server will throw away - which is the direct price of avoiding HRR, and measuring that tradeoff on real links is what settles the predict-versus-retry question for your deployment rather than anyone else's.

2.4 Can an attacker just strip it?

The obvious attack on any negotiated upgrade is to un-negotiate it: sit in the middle, strip the hybrid groups from the `ClientHello`, and force both ends down to plain X25519. TLS 1.2 was burned by exactly this - FREAK and Logjam - because it let an attacker tamper with negotiation in ways the handshake never caught.

TLS 1.3 closes that door against a *classical* attacker. The `ClientHello` is not signed, but the server's `CertificateVerify` signs the transcript hash - which includes the `ClientHello` as the server received it - and the `Finished` MACs cover it too. Strip the hybrid group and the server signs a transcript the client cannot reproduce from what it actually sent; verification fails and the handshake dies before any application data flows.

But notice what holds that up: the server's signature, which today is traditional (non-PQ). A quantum attacker who breaks X25519 breaks that signature too - and then "can it strip the hybrid group?" is the wrong question, because it can impersonate the server. Transcript integrity offers nothing, because the thing vouching for the transcript is exactly the thing that broke.

That splits the quantum threat in two. A *passive* harvest-now-decrypt-later attacker is the one RFC 10024 is built for: the recorded session's confidentiality rests on ML-KEM, which the future CRQC does not break - as long as the server never offered a traditional-only session to harvest in the first place. That is part of why we marked X25519MLKEM768 as Recommended (`RECOMMENDED=Y`): a nudge toward making the hybrid the default offered first and, eventually, the only key exchange offered at all. An *active* attacker with a working CRQC is a different problem that key exchange cannot solve - forging the server's traditional signature means live impersonation no matter how quantum-safe the key agreement is. Closing that gap means moving signatures to post-quantum too.

One caveat stays at the traditional layer: the guarantee stops at the edge of TLS. If a client retries without the hybrid groups after a failed handshake - fallback logic above the transcript - an attacker can farm that retry. The defense is policy,

not protocol: do not fall back. The ecosystem learned that in the TLS 1.2 fallback wars after POODLE.

2.5 Turning it on yourself

None of this requires special software anymore. OpenSSL ships ML-KEM and the hybrid groups natively since 3.5, and X25519MLKEM768 sits in the default TLS 1.3 group list, so an up-to-date stack often speaks it with no configuration at all. To be explicit - or to control ordering, which as noted is policy - an NGINX server sets:

```
ssl_protocols TLSv1.3;
ssl_ecdh_curve X25519MLKEM768:X25519:prime256v1;
```

with the hybrid first so it wins whenever the client offers it, and traditional groups behind it for older peers. The name goes in the same directive that has always carried curves; that a lattice KEM hides behind `ssl_ecdh_curve` is the framework's `supported_groups` abstraction paying off one more time. Verification is one command:

```
openssl s_client -connect example.com:443 -groups X25519MLKEM768
```

Clients in Go, BoringSSL, Firefox, and Chrome do the equivalent by default now. The entire server-side migration, for a typical deployment, is two lines of config and one of verification - compare that with what enabling HTTPS used to take.

One counterintuitive thing the benchmarks teach you: in hybrid mode, optimizing ML-KEM barely moves the needle, because the handshake cost is dominated by the traditional half. Measurements of ML-KEM in OpenSSL 3 show hybrid handshakes running at a fraction of the PQ-only rate - the ECDH operation, not the lattice one, is the bottleneck once both are present. Hand-tuning ML-KEM speeds up the KEM by a large factor in isolation and leaves the hybrid handshakes-per-second almost unchanged. So much for the instinct that the post-quantum part must be the expensive part: for X25519MLKEM768 on a modern server, the twenty-year-old curve is the slow half.

3 What is still missing

Key agreement was the easy half. Confidentiality is fixed; authentication is not, and the second half of the migration drags certificate authorities, HSMs and revocation behind it - which is why it will be slower than everything above.

3.1 The debate that would not die

The longest-running argument was never about how to do hybrid. It was whether to do hybrid at all. One camp: ML-KEM went through the most scrutinized public cryptanalysis effort in history, so ship it alone and stop paying for elliptic

curves. The other camp had lived through the SIKE break and wanted the curve to stay until lattices have aged. It produced rough consensus that hybrid is the conservative default (for now), plus code points for those whose threat model says otherwise - and one of those threat models has an agency behind it. NSA's CNSA 2.0 requires ML-KEM alone, which is why draft-ietf-tls-mlkem exists: the working group's answer was to publish the pure code points marked not recommended rather than argue a national policy out of its own position.

The "ship ML-KEM alone" camp isn't wrong that it's the most-studied option we have. They're wrong about how much that's worth. "Most-studied" is a statement about the past - and here is what it waves away: there is no bench. If ML-KEM falls, we do not calmly swap in the next lattice KEM. Even if HQC is standardized, it still will be too big to swing at TLS-handshake scale; nothing else is close. ML-KEM is not the best of several post-quantum options - it is very nearly the only one that fits on the wire. So a lattice break is not a bad afternoon, it is game over for post-quantum confidentiality. The traditional half is not a spare tyre; it is the whole fallback. Strip it off and you are betting the confidentiality of the internet on a single unbroken conjecture - and I have watched a NIST finalist die in production.

That "for now" is the whole point. At IETF 126 I learned that customers of a large security vendor need *both* components FIPS-certified - the reason the NIST-curve groups exist, since X25519 will never be certified. They already run FIPS-certified P-256; ML-KEM certification is still catching up, and FIPS validation takes on the order of two years. They are not trying to run two algorithms forever - they are trying to get from a fully-traditional, fully-certified world to a post-quantum one without being out of compliance for a single day in between. The same pressure shapes the silicon: FIPS-certifiable primitives sell better, so the ECDH accelerator in a smartcard or HSM often speaks P-256/P-384. SecP256r1MLKEM768 and SecP384r1MLKEM1024 are not there because anyone loves them; they are there so the regulated, embedded corner of the Internet can cross too, on its own slow clock.

Which is the thing to hold onto: a hybrid is not a destination. It is scaffolding - load-bearing while the migration happens, meant to come down once both ends have arrived. That reframes every awkward compromise in the RFC as a property of a structure built to be temporary, not a flaw in the cryptography.

You certify because you need the implementation correct, and you hybridize because the math might fall. FIPS validation checks the first: that an implementation computes the algorithm correctly - no RNG shortcuts, no side-channel howlers. It says nothing about whether the math holds. A perfectly certified implementation of a broken algorithm is a perfectly certified way to leak your keys. Demanding certified components and budgeting for one of them to break is not a contradiction - it is the whole idea.

If that sounds theoretical, it stopped being theoretical the week I wrote this. HAWK - a lattice-based signature candidate that had survived a few years of

expert review in NIST's process - just lost roughly half its security bits to an attack found by an AI system - Anthropic's Claude - running largely on its own. Cryptanalysis is now accelerated by machines, and the gap between "reviewed by everyone" and "broken by someone" is closing faster than any certification cycle. ML-KEM's problem is different and unaffected, and remains the best-studied option we have. That is exactly the confidence that justifies shipping it *inside a hybrid*. SIKE taught the lesson once with humans; HAWK just taught it again with machines.

The curve stays.

3.2 What is left: the road to 2029

I say that having just spent a whole post on how hard it was. Two things remain.

The first is negotiation without guessing. Today a client learns whether a server speaks hybrid by trying; the future is learning it from DNS. The HTTPS record already carries service parameters before the first TLS packet flies, and a flag advertising post-quantum key agreement there would let clients send the right share on the first attempt - no `HelloRetryRequest`, no wasted kilobyte, no probing.

The second, and far bigger, is authentication. Everything in this post protects confidentiality - a recorded session cannot be decrypted later. But the signature on the certificate chain is still ECDSA or RSA, so an attacker with a working quantum computer can impersonate the server, forge a software update, or mint a credential and walk in the front door. Confidentiality had to be fixed *years* before Q-Day because of harvest-now-decrypt-later; authentication only has to be fixed *by* Q-Day - a comfortable ordering right up until Q-Day estimates started moving.

It will also be the harder book-end, for reasons that have nothing to do with quantum computers. It drags certificate authorities, transparency logs, HSMs, embedded roots of trust, and revocation behind it. The signatures themselves are a regression, worst exactly where it hurts most: ML-DSA signing is a variable-time rejection loop rather than the fixed cost of a key exchange, and the size blowup breaks the single-packet assumptions on the constrained devices least able to absorb it. And it has a downgrade problem transcripts cannot solve - a quantum attacker who can forge the traditional signature can forge the whole handshake, so eventually the traditional option must be turned *off*, not merely deprioritized.

History sets the pace, and the hash functions are the cautionary tale. MD5 had a forged certificate demonstrated in 2008, yet browsers took until 2013 to fully deprecate it - five years to remove a hash everyone agreed was broken. SHA-1 followed the same arc, and its deprecation was slowed by something more stubborn than inertia: the replacement was not free at the edges (Cloudflare measured exactly this). SHA-2 was missing from Windows XP before Service Pack 3 and from a swathe of older Android phones, so cutting over meant cutting off the

oldest, least-updatable corner of the web - the same corner, the developing-world and legacy-device long tail, that could least afford to be cut off. That was a hash swap, no new hardware, no kilobytes on the wire. Post-quantum certificates cost both, and ML-DSA is not broken yet, so the temptation to wait will be stronger, not weaker. When someone writes the RFC 10024 retrospective for post-quantum authentication, I expect it to describe threads that make ours look brisk.

3.3 Why the boring outcome is the achievement

While we argued, the thing shipped. Experimental code points for the draft version went into browsers and CDNs years before the RFC existed, and by the time the document reached the RFC Editor, a huge share of TLS handshakes on the public internet were already hybrid-protected against harvest-now-decrypt-later collection. The final RFC then did something quietly satisfying: it obsoleted the experimental code points, including the pre-standard Kyber ones, like scaffolding coming down.

The result is that the largest cryptographic migration in the history of the internet is happening and almost nobody notices. No flag day. No broken connections. A user's browser negotiates 1216 bytes of keyshare instead of 32, and their traffic recorded today is worthless to whoever is storing it for a future quantum computer.

Nine pages. If you only read the RFC, you would think it was easy. The other several thousand pages are in the mailing list archive, and that archive - slow, repetitive, occasionally maddening - is where the actual security of your TLS connection was negotiated. I would do it again. Slightly faster, ideally. But again.