# Making ML-DSA Practical for Client mTLS

## Compression limits, protocol architecture, and a research roadmap

**DEEP RESEARCH REPORT**

Evidence cut-off: 4 September 2026

Scope: TLS 1.3 client authentication, X.509 private PKI, hybrid post-quantum key exchange, constrained and lossy networks

**DECISION IN ONE PARAGRAPH**

A conforming FIPS 204 ML-DSA signature cannot be compressed from 2,420-4,627 bytes to RSA-2048 or ECDSA-P256 size. The dominant response vector is already tightly bit-packed and intentionally close to a broad, secret-independent distribution. For ML-DSA-44, even deleting the challenge and hint entirely would leave 2,304 bytes. The practical way to make ML-DSA mTLS feel like classical mTLS is to pay for one full ML-DSA authentication, then use TLS 1.3 PSK-DHE resumption with a fresh hybrid X25519+ML-KEM key exchange on reconnects. If every connection must carry a fresh, independently verifiable post-quantum signature near 70-256 bytes, a different signature construction is required.

Standards, active drafts, peer-reviewed research, and original quantitative derivations are distinguished throughout.

# Contents

The report separates immutable ML-DSA encoding limits from system-level methods that can make post-quantum client authentication deployable.

# Executive decision

The requested goal has three different meanings. They have different answers.

| Goal | Result | Best path |
|---|---|---|
| Make the raw FIPS 204 ML-DSA signature 70-256 bytes | Not achievable while remaining ML-DSA/FIPS 204 | Change signature algorithm or security assumptions |
| Make repeated client-authenticated connections carry classical-like authentication overhead | Achievable | Full ML-DSA bootstrap, then TLS 1.3 resumption with fresh hybrid key exchange |
| Make the first full ML-DSA mTLS handshake robust on constrained links | Achievable with engineering | ML-DSA-44, short chains, certificate compression, cached/pre-distributed issuers, fragmentation testing |

The recommended deployment profile in this report is called **MBAR: ML-DSA Bootstrap and Adaptive Resumption**. It does not invent or weaken a signature scheme. It leaves FIPS 204 unchanged, uses the standardized TLS 1.3 resumption key schedule, and makes reauthentication policy explicit. The main research opportunity is to formalize and measure the security-performance envelope of this profile for VPN and device mTLS deployments.

# 1. What is signed, and when

There are two independent signature events and one independent key-exchange event.

## 1.1 Certificate issuance

A certification authority signs the DER-encoded `TBSCertificate`. That object contains the subject identity, validity, extensions, and the subject public key. The resulting CA signature is stored in the certificate's `signatureValue`. A certificate never contains the subject's private key. The private key remains in the client, HSM, secure element, TPM, or other key store. The CA's signature algorithm need not be the same as the subject public-key algorithm. For example, a CA may sign a certificate containing an ML-DSA public key with a different permitted issuer algorithm. RFC 9881 defines the use of ML-DSA keys and signatures in X.509 [3].

Every certificate in a transmitted chain may therefore contribute its own issuance signature and public key. A longer chain increases the TLS `Certificate` message. It does not change the length of the client's one `CertificateVerify` signature.

## 1.2 Full TLS 1.3 client authentication

The client sends a certificate chain and proves possession of the leaf certificate's private key with `CertificateVerify`. RFC 9846 defines the signed input as:

```
64 space bytes
|| "TLS 1.3, client CertificateVerify"
|| 0x00
|| Transcript-Hash(Handshake Context, Certificate)
```

With SHA-256, this input is 130 bytes; with SHA-384, it is 146 bytes. The transcript hash is fixed length, but signature output length is a property of the signature scheme, not of the input length. RSA-PSS, ECDSA, and ML-DSA can sign the same 130-byte formatted input and produce very different outputs. The receiver verifies with the public key in the client's leaf certificate [2].

The `CertificateVerify` structure adds four bytes around the signature: two bytes for `SignatureScheme` and two for the signature length. The TLS handshake header adds another four bytes. Thus a standalone handshake message is eight bytes larger than the raw signature, before record protection.

| Scheme | Raw signature | CertificateVerify body | Handshake message | Notes |
|---|---|---|---|---|
| RSA-PSS with RSA-2048 | 256 B | 260 B | 264 B | Fixed by 2,048-bit modulus |
| ECDSA P-256 | usually 70-72 B | 74-76 B | 78-80 B | TLS uses DER; about 71 B is a useful average |
| ML-DSA-44 | 2,420 B | 2,424 B | 2,428 B | FIPS 204 category 2 |
| ML-DSA-65 | 3,309 B | 3,313 B | 3,317 B | FIPS 204 category 3 |
| ML-DSA-87 | 4,627 B | 4,631 B | 4,635 B | FIPS 204 category 5 |

The raw-signature ratios are therefore:

| Scheme | Versus RSA-2048 | Versus 71-byte ECDSA |
|---|---|---|
| ML-DSA-44 | 9.45x | 34.1x |
| ML-DSA-65 | 12.93x | 46.6x |
| ML-DSA-87 | 18.07x | 65.2x |

The phrase "about 20x" is reasonable only for some comparisons. It is not a universal ratio.

### Raw signature size on a logarithmic scale

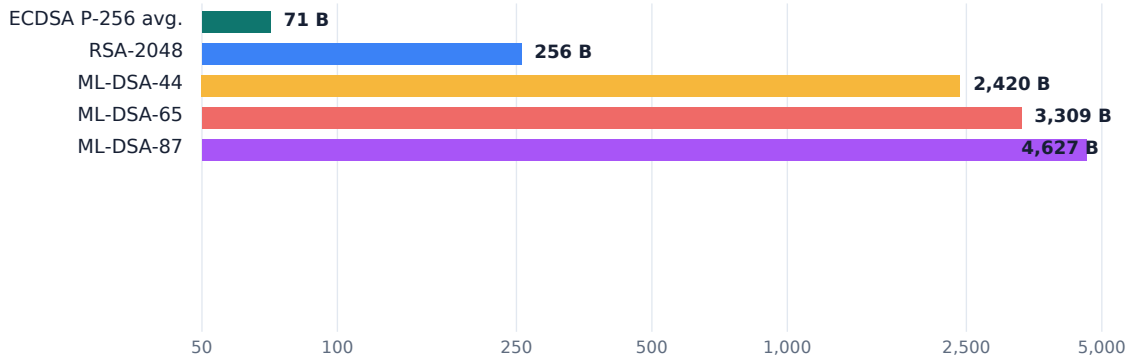


Figure 1. ML-DSA-44 is 9.45x RSA-2048 and 34.1x a typical ECDSA-P256 TLS signature.

## 1.3 Hybrid post-quantum key exchange

Key exchange and certificate authentication are orthogonal. X25519MLKEM768 contributes a 1,216-byte client key share and a 1,120-byte server key share [5]. It establishes fresh shared secrets; it does not prove the client's certified identity. ML-DSA `CertificateVerify` supplies that proof in a full certificate-authenticated handshake. On a resumed PSK-DHE connection, possession of the resumption PSK authenticates continuity with the earlier full handshake, while a fresh hybrid key share restores fresh forward-secret key establishment.

# 2. Why ML-DSA signatures are large

ML-DSA is a module-lattice Fiat-Shamir-with-aborts signature. At a high level, the signer samples a masking vector `y`, forms a commitment, hashes the message and commitment to a challenge, and returns a response `z = y + c*s1`. Rejection sampling discards responses that could reveal information about the secret. Verification reconstructs the commitment using the public key, response, challenge, and a rounding hint [1, 17].

FIPS 204 encodes every signature as:

```
signature = challenge_seed || BitPack(z) || HintBitPack(h)
```

The exact length formula is:

```
lambda/4 + l * 32 * (1 + bitlen(gamma1 - 1)) + omega + k bytes
```

This produces the following component budget.

| Parameter set | Challenge | Response z | Hint h | Total | z share |
|---|---|---|---|---|---|
| ML-DSA-44 | 32 B | 2,304 B | 84 B | 2,420 B | 95.2% |
| ML-DSA-65 | 48 B | 3,200 B | 61 B | 3,309 B | 96.7% |
| ML-DSA-87 | 64 B | 4,480 B | 83 B | 4,627 B | 96.8% |

This decomposition immediately rules out several intuitive ideas. For ML-DSA-44, removing every byte of the challenge and hint would still leave a 2,304-byte response, which is 9x an RSA-2048 signature and more than 32x a typical ECDSA-P256 signature.

### What occupies an ML-DSA signature

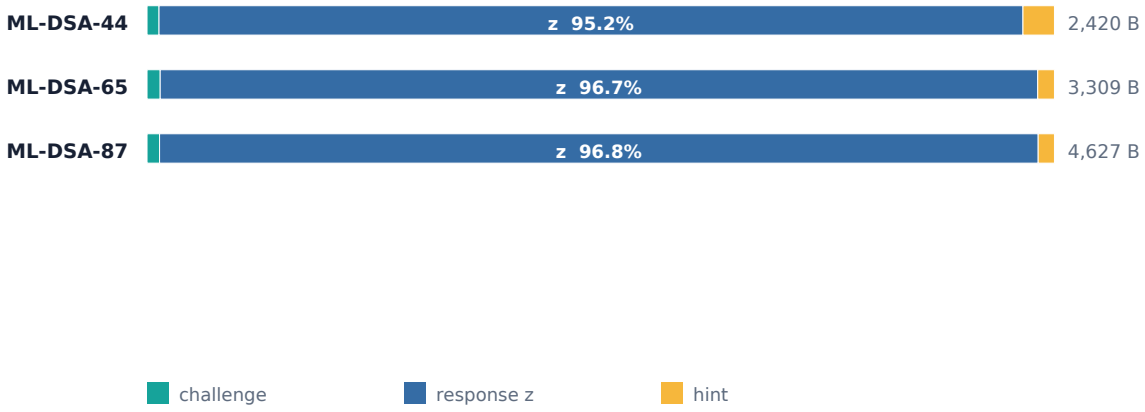


Figure 2. The high-dimensional response `z` occupies more than 95% of every FIPS 204 signature.

The large response is not an accidental DER wrapper. It consists of 1,024, 1,280, or 1,792 small polynomial coefficients, packed at 18 or 20 bits per coefficient. Those coefficients are the short lattice witness response that lets the verifier check knowledge of the secret without learning it.

# 3. Compression limit: a standards and entropy analysis

## 3.1 Standards boundary

FIPS 204 requires exact public-key and signature lengths. A conforming verifier that accepts a different length must return failure [1]. The current TLS ML-DSA specification maps the FIPS 204 algorithms directly into TLS `SignatureScheme` values and uses the FIPS signing and verification algorithms [4]. Therefore any alternate wire encoding needs decoding back to the exact FIPS string before verification and needs a negotiated protocol extension. It is not a smaller ML-DSA signature; it is transport compression around one.

## 3.2 Derived bound for the response range

This subsection is an original calculation from the FIPS 204 parameters. It is a generous best case for a hypothetical canonical range encoder, not a claim that the standard permits a new encoding.

FIPS packs each `z` coefficient over `2*gamma1` values. Verification accepts the tighter norm range with:

```
number of accepted integer values = 2*(gamma1 - beta) - 1
```

If a perfect enumerative coder represented only those accepted values, its ideal saving against the current fixed bit width would be:

```
l*256 * log2((2*gamma1) / (2*(gamma1-beta)-1)) bits
```

| Parameter set | Response coefficients | Current response | Ideal range saving over all coefficients |
|---|---|---|---|
| ML-DSA-44 | 1,024 | 2,304 B | 0.111 B |
| ML-DSA-65 | 1,280 | 3,200 B | 0.087 B |
| ML-DSA-87 | 1,792 | 4,480 B | 0.074 B |

The result is less than one bit per complete response vector. The accepted range almost exactly fills the power-of-two bit packing. There is no hidden two-kilobyte margin in `z`.

This agrees with the design rationale: rejection sampling is used to make the response distribution secret-independent and close to uniform over a broad hypercube [17]. Broad near-uniform cryptographic values are deliberately poor inputs for generic lossless compression.

## 3.3 Derived best case for the hint

The hint has at most `omega` set positions among `k*256` coefficients. FIPS stores those positions and per-polynomial boundaries in `omega+k` bytes. A generous enumerative lower length for all subsets of weight at most `omega` is:

```
ceil(log2(sum from j=0 to omega of C(k*256,j)) / 8) bytes
```

| Parameter set | FIPS hint | Generous enumerative length | Maximum structural saving | Total-signature saving |
|---|---|---|---|---|
| ML-DSA-44 | 84 B | 51 B | 33 B | 1.36% |
| ML-DSA-65 | 61 B | 43 B | 18 B | 0.54% |
| ML-DSA-87 | 83 B | 58 B | 25 B | 0.54% |

This bound assumes a coder can exploit the full sparse-set model without operational cost and says nothing about the actual hint distribution. Even the theoretical prize is small because the hint is a small fraction of the signature. Work on Dilithium hints also finds an inherent tradeoff: compressing the public key more aggressively transfers information into the per-signature hint [18].

## 3.4 Required reduction versus plausible reduction

| Target | ML-DSA-44 reduction required | ML-DSA-65 | ML-DSA-87 |
|---|---|---|---|
| RSA-2048, 256 B | 89.4% | 92.3% | 94.5% |
| ECDSA-P256, 71 B average | 97.1% | 97.9% | 98.5% |

By contrast, exploiting the obvious response-range and hint structure yields at most about 1.4% for ML-DSA-44 and below 0.6% for the larger sets. This is a two-orders-of-magnitude mismatch between the desired saving and the available encoding slack.

## 3.5 Why common compression proposals fail

| Proposal | Assessment | Reason |
|---|---|---|
| gzip, Brotli, zstd, or TLS certificate compression on `CertificateVerify` | No material saving | The response and challenge are high entropy; RFC 8879 compresses only `Certificate`, not `CertificateVerify` [6] |
| Sign a prehash instead of the transcript | No size saving | Pure and HashML-DSA have the same output lengths; TLS already signs a compact transcript-derived value [1, 2] |
| Deterministic signing | No size saving | It changes randomness derivation, not the response representation |
| Transmit a 32-byte seed that expands to $z$ | Not a valid encoding | A verifier cannot reconstruct the correct algebraically related response from a generic seed; making this work changes the scheme and proof |
| Omit or derive the challenge | Circular or insecure | The challenge binds the message and commitment; omitting it changes Fiat-Shamir verification |
| Lossy quantization of $z$ | Invalid signatures | Verification requires exact modular arithmetic and norm bounds |
| Reuse a previous signature | Fails authentication | Every TLS transcript is different; replayed `CertificateVerify` does not verify |
| Aggregate signatures | Wrong workload | One client creates one signature; current lattice aggregation gives limited savings and changes verification semantics [20] |
| Composite classical+ML-DSA signature | Larger | It adds a classical signature for transition diversity; it is not compression [28] |

# 4. What can be changed inside a lattice signature

To reduce the raw signature substantially, the construction must change. That means the result is no longer FIPS 204 ML-DSA even if it remains a module-lattice signature.

### 4.1 Change the geometry and sampler

ML-DSA's uniform hypercube response is simple and implementation-friendly, but not size-optimal. HAETAE uses a bimodal distribution and hyperball sampling/rejection, reporting signature and verification-key sizes up to 39% and 25% smaller than Dilithium [19]. This is a real research direction, but a 39% reduction from 2,420 bytes is still roughly 1,476 bytes, not RSA or ECDSA size. It also introduces a new design, parameter set, proof, implementation, and cryptanalytic target.

### 4.2 Use NTRU trapdoors and Gaussian hash-and-sign

Falcon, now named FN-DSA by NIST, uses the GPV hash-and-sign paradigm over NTRU lattices. Falcon-512 has an 897-byte public key and a 666-byte signature [13]. It is much smaller than ML-DSA-44 but remains 2.6x RSA-2048 and about 9.4x a 71-byte ECDSA signature. Its tradeoff is substantially more complex signing, including careful Gaussian sampling and floating-point behavior. NIST has selected it for FIPS 206, which was still under development at the evidence cut-off [11, 12].

### 4.3 Move information between public key and signature

Several schemes obtain short signatures by accepting very large public keys. UOV has reported category-1 signatures as short as 96 bytes but expanded public keys over 200 KB. NIST also reports that attacks reduced the security of three of four second-round parameter sets, so reparameterization is expected [10]. This may work where one public key is pinned once and millions of signatures are verified, but it is unattractive for X.509 mTLS where the public key may be transmitted in a certificate.

The broader lesson is that signature bytes cannot be optimized alone. The relevant objective is often:

```
first-use cost = certificate chain + leaf public key + CertificateVerify signature
```

and, for repeated connections:

```
amortized cost = first-use cost / reuse_count + resumed-authentication overhead
```

### 4.4 Change the hardness family

SQIsign's third-round parameters list public keys of 83/129/169 bytes and signatures of 200/306/406 bytes for NIST categories I/III/V [14]. This demonstrates that post-quantum signatures are not subject to a universal multi-kilobyte lower bound. It does not make ML-DSA compressible. SQIsign has a complex signer, high cycle counts, newer assumptions, and continuing constant-time/side-channel work. It is a NIST additional-signature candidate, not a production standard [10, 14].

MPC-in-the-head and many code-based candidates generally remain in the kilobyte signature range. SLH-DSA offers conservative hash-based assumptions but starts at 7,856 bytes for the small category-1 parameterization and is worse for this wire-size objective [11].

## 5. Recommended system: MBAR

### 5.1 Design objective

MBAR keeps a full, standards-conforming ML-DSA proof at the trust bootstrap and replaces repeated public-key proofs with TLS 1.3's cryptographically bound resumption PSK. Every resumed connection also carries a fresh X25519MLKEM768 key share, so authentication continuity and fresh hybrid key establishment are combined.

The semantic change must be explicit:

- A full handshake proves current possession of the ML-DSA private key and validates the certificate chain.
- A resumed handshake proves possession of a PSK derived from a prior authenticated handshake.

- The resumed connection is cryptographically tied to the earlier handshake, but it is not a fresh publicly verifiable ML-DSA signature [2].

For VPN and EAP-TLS-like environments, RFC 9190 already provides the closest standards precedent: authorization during resumption must be based on securely retrieved cached data from the original full handshake, and fresh revocation checks may be performed [7].

### MBAR separates one full proof from compact continuity proofs

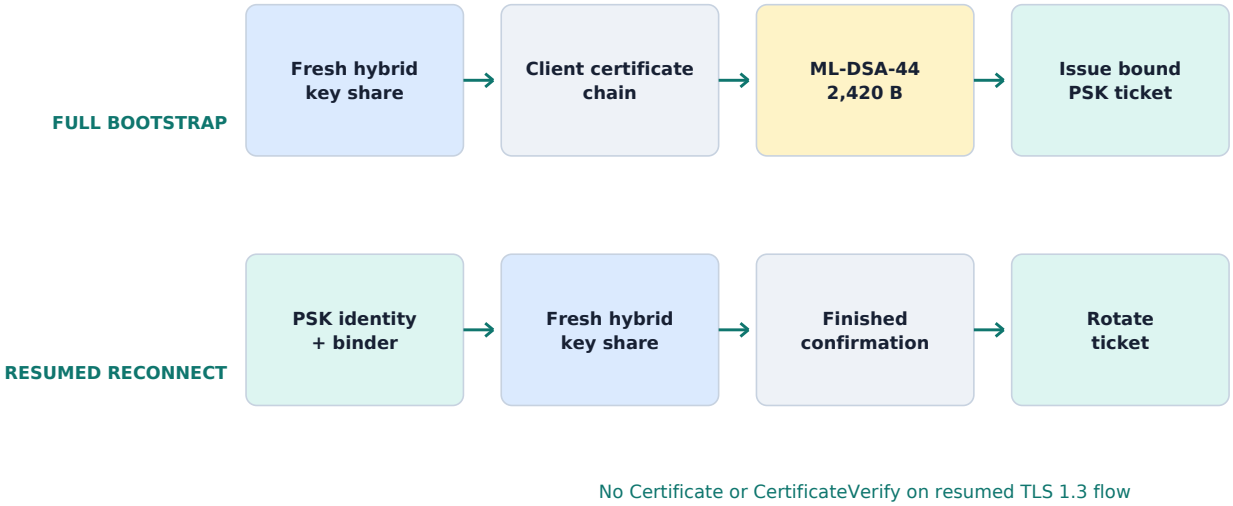


Figure 3. Resumption changes repeated authentication from a fresh public signature to proof of a PSK derived from the full ML-DSA handshake.

## 5.2 Full bootstrap flow

1. Client and server negotiate TLS 1.3 and a fresh hybrid X25519MLKEM768 key share.
2. Server authenticates and requests a client certificate.
3. Client sends the shortest acceptable ML-DSA certificate chain and one ML-DSA-44 `CertificateVerify` signature.
4. Server validates certificate path, revocation, identity, tenant, device posture, and policy.
5. After the client Finished, server issues one or more `NewSessionTicket` values.
6. Server stores or authenticates compact resumption state bound to the authorization decision.

## 5.3 Resumed flow

1. Client sends a PSK identity, binder, and fresh X25519MLKEM768 key share.
2. Server retrieves and validates the cached authentication/authorization state.
3. Server accepts PSK-DHE only if freshness and policy checks pass.
4. Client and server complete Finished authentication. No `Certificate` or `CertificateVerify` is sent in the standard TLS 1.3 resumption flow [2].
5. Server rotates the ticket. Prefer single-use tickets where the deployment can maintain state.

## 5.4 Ticket state and binding

For a private VPN/mTLS deployment, a stateful opaque ticket identifier is usually the most bandwidth- and revocation-friendly choice. The server-side record should bind at least:

| Field | Purpose |
|---|---|
| Client identity and tenant | Prevent cross-account and cross-tenant reuse |
| Leaf SPKI hash and certificate serial/issuer | Bind to the authenticated ML-DSA key and credential |
| Full-authentication time | Enforce periodic fresh proof-of-possession |
| Ticket expiry | Bound continuity window; TLS caps tickets at 7 days [2] |
| Certificate expiry and revocation status time | Prevent use beyond credential validity |
| Authorization/policy epoch | Invalidate tickets when roles or access policy change |
| Revocation epoch | Force refresh after revocation data changes |
| Device-posture epoch | Re-evaluate endpoint health or compliance |
| Negotiated PQ suite and security profile | Prevent silent algorithm-policy drift |
| Server audience/cluster | Prevent ticket use at an unintended service |
| Unique nonce and use state | Support rotation and single-use anti-replay policy |

Stateless encrypted tickets remain possible, but they are usually larger and harder to revoke immediately. Short lifetimes, ticket-encryption-key rotation, and a denylist or epoch mechanism become more important.

## 5.5 When to force a new full ML-DSA handshake

Force full authentication when any of the following occurs:

- no ticket, invalid ticket, expired ticket, or already-used single-use ticket;
- certificate expiration, known revocation, or stale revocation information;
- user role, tenant, policy, device posture, or risk score changed;
- server audience or cluster binding does not match;
- required algorithm suite or cryptographic policy changed;
- maximum full-authentication age reached;
- incident response or key-compromise signal requires reauthentication;
- a configurable random sample is selected for measurement or assurance.

Do not enable 0-RTT application data for security-sensitive VPN actions. TLS 1.3 0-RTT has replay caveats. One-RTT PSK-DHE resumption still removes the certificates and signatures while providing a fresh asymmetric key share [2].

## 5.6 Amortization calculation

Ignoring bytes common to all handshakes, one ML-DSA-44 signature amortized over $N$ authenticated connections contributes $2420/N$ bytes per connection.

| Full auth followed by resumptions | Average ML-DSA signature bytes per connection |
|---|---|
| 1 connection | 2,420 B |
| 5 connections | 484 B |

| Full auth followed by resumptions | Average ML-DSA signature bytes per connection |
|---|---|
| 10 connections | 242 B |
| 20 connections | 121 B |
| 35 connections | 69 B |
| 100 connections | 24 B |

If a 32-byte SHA-256 PSK binder is charged to each resumed connection, the average proof material is approximately:

```
(2420 + 32*(N-1)) / N
```

This falls below 256 bytes at 11 total connections and below 71 bytes at 62 total connections, excluding the ticket identity and length fields. A compact stateful identity may be tens of bytes; a stateless encrypted ticket is deployment-specific and can be much larger. These calculations do not claim that the whole resumed handshake is 32 bytes. The fresh hybrid key shares alone remain 1,216 and 1,120 bytes [5]. They show that recurring certificate-authentication cost can be made classical-like or smaller without modifying ML-DSA.

# 6. Make the full handshake robust

## 6.1 Use the smallest acceptable ML-DSA set

ML-DSA-44 is the natural bandwidth choice when its FIPS 204 category and organizational policy are sufficient. Moving to ML-DSA-65 adds 889 signature bytes and 640 public-key bytes. Moving to ML-DSA-87 adds 2,207 signature bytes and 1,280 public-key bytes relative to ML-DSA-44 [1]. Choose security level from threat and compliance requirements, not from an assumption that all post-quantum settings are interchangeable.

## 6.2 Shorten the private-PKI chain

For a controlled fleet:

- use one intermediate CA where governance permits;
- omit a known trust anchor from the TLS chain;
- pre-distribute stable intermediates and omit certificates the verifier is known to possess, as TLS 1.3 permits;
- remove unnecessary names, postal data, policy OIDs, and extensions;
- avoid duplicating authorization data in certificates when a server-side identity record suffices;
- keep revocation and rotation workable rather than over-optimizing a static credential.

RFC 9191 reports that EAP peer chains commonly contain two to six intermediates and recommends pre-distribution, omission of known CAs, fewer intermediates, and minimal certificate contents [8].

## 6.3 Enable certificate compression, with the right expectation

RFC 8879 lets a server advertise compression support in `CertificateRequest`, enabling compressed client `Certificate` messages [6]. This can compress repeated ASN.1 structure, names, extensions, and other redundancy. It will barely compress ML-DSA public keys and CA signatures because those fields are high entropy. It never compresses the separate `CertificateVerify` message.

Use it, measure it on the actual chain, and do not count on web-certificate compression ratios carrying over to PQ-heavy private certificates.

### 6.4 Consider raw public keys only in a controlled trust model

RFC 7250 raw public keys remove most X.509 metadata and the chain. Client identity must then be securely bound to the raw SubjectPublicKeyInfo out of band; revocation and rotation move into the provisioning system [9]. This can be compelling for a managed device fleet with an existing enrollment database. It still leaves the 2,420-byte ML-DSA-44 `CertificateVerify` signature.

### 6.5 Engineer fragmentation deliberately

Large handshakes are not just a byte-cost issue. They cross packet, record, congestion-window, EAP, RADIUS, and radio-loss thresholds. RFC 9191 reports common EAP fragment sizes of 1,020-1,500 bytes and access points that abandon authentication after 40-50 round trips [8]. At a 1,020-byte EAP payload, the standalone client `CertificateVerify` handshake message alone needs at least:

| Signature | Handshake bytes | Minimum 1,020-byte fragments |
|---|---|---|
| ECDSA-P256 typical | 78-80 | 1 |
| RSA-2048 | 264 | 1 |
| ML-DSA-44 | 2,428 | 3 |
| ML-DSA-65 | 3,317 | 4 |
| ML-DSA-87 | 4,635 | 5 |

The certificate chain is additional. Measure loss, retransmission, handshake timeout, radio wakeups, and CPU on the actual VPN transport and access network. Byte totals alone miss threshold effects.

### 6.6 Reduce the number of handshakes

Keep the authenticated tunnel alive, pool application connections through it, and use connection migration where the transport supports it. This does not reduce a signature, but it directly reduces signatures per user-hour and is often more valuable than shaving tens of bytes from certificate metadata.

## 7. Alternatives when every connection needs fresh PQ authentication

### 7.1 Compact-signature landscape

| Mechanism | Public key | Signature/ciphertext | Maturity at cut-off | Main tradeoff |
|---|---|---|---|---|
| ECDSA-P256 | about 64-65 B | usually 70-72 B DER | Standard, classical only | Broken by a cryptographically relevant quantum computer |
| RSA-2048 | modulus 256 B | 256 B | Standard, classical only | Broken by a cryptographically relevant quantum computer |
| ML-DSA-44 | 1,312 B | 2,420 B | FIPS 204 | Best-established standardized general-purpose PQ option; large response |
| FN-DSA/Falcon-512 | 897 B | 666 B | Selected; FIPS 206 pending | Complex Gaussian/FFT signer and side-channel engineering |

| Mechanism | Public key | Signature/ciphertext | Maturity at cut-off | Main tradeoff |
|---|---|---|---|---|
| HAETAE-like design | scheme-specific | roughly 30-40% below Dilithium in reports | Research/non-NIST standard | New sampler, parameters, proof, and implementation |
| SQIsign category I, round 3 | 83 B | 200 B | NIST candidate | Complex and relatively slow signer; newer assumptions and side-channel work |
| UOV category 1, round 2 | over 200 KB expanded | as low as 96 B | NIST candidate; parameters under revision | Huge public key; recent attacks affected parameters |
| SLH-DSA-128s | 32 B | 7,856 B | FIPS 205 | Conservative hash assumptions; very large signatures |
| AuthKEM with ML-KEM-768 | 1,184 B | 1,088 B ciphertext | IETF individual draft | Not a signature; changes handshake and client auth adds a flight |

Numbers from schemes in an active competition can change between rounds. In particular, NIST IR 8610 describes a 148-byte second-round SQIsign category-1 signature, while the current round-3 project page lists 200 bytes. This report uses the current round-3 number and preserves the older number only as history [10, 14]. HAWK was selected into round 3 and then withdrawn after new cryptanalysis, a useful warning against deploying compact candidates before they mature [12].

## 7.2 KEM-based authentication

KEMTLS replaces handshake signatures with possession of a long-term KEM private key. Its paper reported less than half the bandwidth of a size-optimized PQ TLS 1.3 instantiation and major server-cycle savings [15]. The current AuthKEM Internet-Draft adapts this direction to TLS. A peer encapsulates to the certified long-term KEM public key; only the private-key holder derives the secret that authenticates subsequent messages [16].

For client authentication, the server must receive the client's KEM certificate and then encapsulate to it, adding a server-to-client authentication flight. ML-KEM-768's 1,088-byte ciphertext is about 55% smaller than an ML-DSA-44 signature, but still 4.25x RSA-2048 and over 15x 71-byte ECDSA. AuthKEM is promising when bandwidth, protected-code size, or KEM hardware dominate, but it is a protocol change and an active draft, not a drop-in ML-DSA compressor.

## 7.3 Merkle Tree Certificates and credential caches

Merkle Tree Certificates batch CA issuance under Merkle commitments and can reduce repeated CA signatures in certificate delivery [21]. They optimize the `Certificate` side of the flight, not the endpoint's fresh `CertificateVerify` proof. The work remains an Internet-Draft.

A private deployment can explore a **PQ Client Credential Cache** extension: after a full handshake, the server issues a short opaque handle for a cached client chain and public key. A later non-PSK full authentication sends the handle plus a fresh ML-DSA `CertificateVerify`. This preserves fresh key proof and saves the chain, but it cannot save the 2,420-byte signature. It needs downgrade protection, cache-miss fallback, tenant/audience binding, expiry, revocation semantics, and standardization if used beyond one controlled system.

# 8. Original research contribution: a disciplined feasibility program

This report does not claim a new cryptographic primitive. Calling an unreviewed construction "secure" would be irresponsible. It proposes a protocol profile, quantitative bounds, and experiments that can produce publishable evidence without weakening ML-DSA.

## 8.1 Research claim A: standards-compatible compression is a closed path

Hypothesis: lossless compression of a conforming ML-DSA-44 `CertificateVerify` cannot deliver a practically important median reduction across independent handshakes.

Evidence already available:

- exact FIPS lengths are mandatory;
- $z$ is 95.2% of ML-DSA-44;
- the accepted-range enumerative saving is less than one bit over the full response;
- deleting all non-$z$ fields still leaves 2,304 bytes;
- the design seeks a broad secret-independent response distribution.

Experiment: collect at least one million signatures across keys, transcripts, randomized and deterministic modes; test zstd, Brotli, arithmetic models by coefficient position, delta models, learned models, and cross-signature dictionaries. Report distribution, not only mean. The expected result is expansion or negligible saving. This experiment is useful as reproducible negative evidence and can prevent wasted implementation effort.

## 8.2 Research claim B: MBAR reaches classical-like amortized authentication cost

Hypothesis: in a reconnect-heavy VPN workload, ML-DSA-44 full authentication plus one-RTT PSK-DHE resumption can reduce median authentication bytes, radio wake time, and time-to-tunnel to the classical baseline without reducing hybrid key-exchange strength.

Experiment matrix:

| Axis | Values |
|---|---|
| Authentication | ECDSA full, RSA full, ML-DSA-44 full, ML-DSA-44+MBAR |
| Key exchange | X25519, X25519MLKEM768 |
| Link | Ethernet, Wi-Fi, 4G/5G, 1-5% induced loss, 20/80/200 ms RTT |
| Credential delivery | full chain, compressed chain, pre-distributed intermediate, raw public key |
| Ticket | stateful short ID, stateless encrypted, single-use stateful |
| Reauth age | 15 min, 1 h, 8 h, 24 h, policy-triggered |

Primary outcomes: total uplink/downlink bytes, packets, retransmissions, round trips, handshake success, p50/p95/p99 latency, client CPU, server CPU, energy, ticket cache hit, forced-full-auth rate, revocation convergence, and failure recovery.

### 8.3 Research claim C: adaptive full reauthentication can be formalized

Define a maximum authentication staleness function:

```
staleness_limit = min(
  ticket_policy,
  certificate_remaining_lifetime,
  revocation_freshness,
  authorization_epoch_freshness,
  posture_epoch_freshness,
  risk_policy
)
```

Resume only when the original full-authentication age is below this limit and all bound epochs match. This turns a vague operational choice into a testable policy. The formal security model should distinguish:

- initial post-quantum credential authentication;
- authentication continuity under the resumption PSK;
- fresh forward secrecy from hybrid PSK-DHE;
- compromise of the ML-DSA key after ticket issuance;
- compromise of the ticket database or ticket-encryption key;
- revocation and authorization freshness;
- replay and cross-service ticket use;
- downgrade and fallback to a full handshake.

The important limitation is post-compromise continuity: stealing a live resumption PSK can authenticate resumptions until it expires or is invalidated even if the ML-DSA key remains safe. Stateful single-use tickets, short lifetimes, secure client storage, and risk-triggered full authentication reduce this window.

### 8.4 Research claim D: protocol work has better expected value than bit coding

Prioritize research effort in this order:

1. Implement and measure MBAR with stateful single-use tickets.
2. Minimize and compress the actual private-PKI chain.
3. Measure fragmentation thresholds on target VPN access networks.
4. Prototype a credential-cache handle only if certificate-chain bytes remain material after resumption.
5. Evaluate AuthKEM as an experimental branch.
6. Track FN-DSA/FIPS 206 and SQIsign/UOV standardization.
7. Explore new lattice samplers or geometries only as a new signature research project, not as an ML-DSA encoding patch.

## 9. Security gates for any proposed smaller signature

Any proposal that changes the raw signature must be treated as a new primitive. Before deployment it needs, at minimum:

- a precise key generation, signing, verification, and encoding specification;
- EUF-CMA and preferably SUF-CMA security goals in a quantum random-oracle model where applicable;

- reductions to clearly stated lattice or other hardness assumptions;
- concrete classical and quantum security estimates with current lattice-estimator methodology;
- multi-key, chosen-message, fault, and misuse analysis;
- strict canonical decoding and malformed-input behavior;
- constant-time signing and verification, including sampler behavior;
- masking and side-channel evaluation on the target hardware;
- deterministic and hedged-signing failure analysis;
- interoperability test vectors and differential tests;
- several years of public cryptanalysis before protecting high-value credentials.

The history of SIKE, multivariate parameter breaks, and HAWK's withdrawal shows why compactness cannot be accepted in place of maturity [10, 12].

# 10. Deployment decision matrix

| Requirement | Recommended choice | Why |
|---|---|---|
| FIPS-approved PQ client signature today | ML-DSA-44 if policy permits | Smallest FIPS 204 signature |
| Classical-like reconnect overhead | MBAR: TLS 1.3 PSK-DHE resumption + fresh hybrid KEX | Removes certificate and CertificateVerify from resumed handshakes |
| Fresh public-key proof on every connection | Accept ML-DSA size or change algorithm | Resumption changes proof semantics; compression cannot close the gap |
| Managed fleet with strong enrollment database | Consider raw ML-DSA public keys or cached credential handles | Removes X.509 chain, not signature |
| Very bandwidth-constrained experimental system | Evaluate AuthKEM and SQIsign in isolated trials | Smaller wire objects, but not mature replacements |
| Conservative non-lattice backup | SLH-DSA | Mature hash assumptions, but wire size is much worse |
| Transition diversity | Composite/hybrid authentication only when explicitly required | Security hedge, but increases size |

# 11. Final conclusion

ML-DSA is not large because TLS signs the wrong object or because the transcript hash is too long. It is large because a secure module-lattice proof carries a high-dimensional short response. FIPS 204 already packs that response at essentially its full coefficient range, and the standard fixes the exact byte string. No lossless compressor, prehash mode, seed trick, certificate-chain change, or TLS certificate compressor can turn 2,420 bytes into 70 or 256 bytes while retaining the same per-connection ML-DSA proof.

The feasible production answer is architectural. Use a full ML-DSA-44 certificate authentication as the bootstrap, issue tightly bound short-lived resumption tickets, resume with one-RTT PSK-DHE plus a fresh X25519MLKEM768 share, and force full ML-DSA reauthentication on expiry, revocation, policy, posture, risk, or periodic age. In parallel, minimize the certificate chain and test fragmentation on the real network.

If the hard requirement is a fresh 70-256-byte post-quantum signature on every connection, the research target is not "compressed ML-DSA." It is a different primitive such as a mature future SQIsign-like scheme, a short-signature/large-key multivariate design, or another construction that survives standardization and

years of cryptanalysis. Today, no production-standard general-purpose PQ signature offers ECDSA-sized signatures.

# Appendix A. Reproducible calculations

## A.1 Component formula

For ML-DSA parameter values `(lambda, l, gamma1, omega, k)`:

```
challenge_bytes = lambda / 4
z_bytes = l * 32 * (1 + bitlen(gamma1 - 1))
hint_bytes = omega + k
signature_bytes = challenge_bytes + z_bytes + hint_bytes
```

## A.2 ECDSA DER average

An ECDSA-P256 signature is a DER SEQUENCE of two INTEGER values `(r,s)`. Each is normally 32 bytes, but a leading zero is added when the high bit would make the INTEGER negative. Ignoring rare shorter magnitudes, this gives 70, 71, or 72 bytes with approximate probabilities 1/4, 1/2, and 1/4, hence about 71 bytes average. Implementations that normalize `s` or use different curves can change the distribution. TLS requires the DER representation, so 64 bytes is the raw `(r,s)` width, not the usual TLS wire length [2, 11].

## A.3 Standalone encrypted-record overhead

When `CertificateVerify` occupies its own TLS 1.3 encrypted record with a 16-byte AEAD tag and no padding, add approximately 22 bytes to the handshake message: five-byte outer record header, one inner content-type byte, and sixteen-byte tag. Records may coalesce multiple handshake messages, so this is not a universal packet-size formula.

# Appendix B. Evidence classification

| Class | Examples | How used |
|---|---|---|
| Final standards | FIPS 204, RFC 9846, RFC 9881, RFC 8879, RFC 9190, RFC 9191, RFC 10024 | Normative sizes, formats, and deployed protocol behavior |
| Standards-process reports | NIST IR 8610, NIST project pages | Candidate maturity, attacks, and selection status |
| Active Internet-Drafts | TLS ML-DSA, AuthKEM, Merkle Tree Certificates | Emerging options only; explicitly not treated as final standards |
| Peer-reviewed papers and primary specifications | Dilithium, Falcon, KEMTLS, HAETAE, aggregation work | Construction rationale and research tradeoffs |
| Derived analysis in this report | response-range bound, hint bound, amortization thresholds | Labeled calculations reproducible from cited parameters |

The search was systematic across NIST standards and candidate reports, IETF TLS/PKIX/EAP specifications, primary scheme specifications, and representative peer-reviewed work on module-lattice response compression, alternative sampling geometry, aggregate signatures, compact PQ signature families, and KEM authentication. It is not literally possible to prove that every paper ever published was read; the report instead targets the authoritative and decision-relevant corpus and records the evidence cut-off.

# References


[1] NIST, FIPS 204, Module-Lattice-Based Digital Signature Standard, 2024. https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.204.pdf

[2] IETF, RFC 9846, The Transport Layer Security (TLS) Protocol Version 1.3, 2026. https://www.rfc-editor.org/rfc/rfc9846.html

[3] IETF, RFC 9881, Algorithm Identifiers for ML-DSA in X.509, 2025. https://www.rfc-editor.org/rfc/rfc9881.html

[4] Hollebeek, Schmieg, Westerbaan, Use of ML-DSA in TLS 1.3, draft-ietf-tls-mldsa-05, work in progress, 2026. https://www.ietf.org/archive/id/draft-ietf-tls-mldsa-05.html

[5] IETF, RFC 10024, Post-Quantum Traditional Hybrid Key Agreement Mechanisms for TLS 1.3, 2026. https://www.rfc-editor.org/rfc/rfc10024.html

[6] IETF, RFC 8879, TLS Certificate Compression, 2020. https://www.rfc-editor.org/rfc/rfc8879.html

[7] IETF, RFC 9190, EAP-TLS 1.3, 2022. https://www.rfc-editor.org/rfc/rfc9190.html

[8] IETF, RFC 9191, Handling Large Certificates and Long Certificate Chains in TLS-Based EAP Methods, 2022. https://www.rfc-editor.org/rfc/rfc9191.html

[9] IETF, RFC 7250, Using Raw Public Keys in TLS and DTLS, 2014. https://www.rfc-editor.org/rfc/rfc7250.html

[10] NIST, NIST IR 8610, Status Report on the Second Round of Additional Digital Signature Schemes, 2026. https://nvlpubs.nist.gov/nistpubs/ir/2026/NIST.IR.8610.pdf

[11] IETF, RFC 9958, Post-Quantum Cryptography for Engineers, 2026. https://www.rfc-editor.org/rfc/rfc9958.html

[12] NIST, Round 3 Additional Signatures, status page, updated 29 July 2026. https://csrc.nist.gov/projects/pqc-dig-sig/round-3-additional-signatures

[13] Fouque et al., Falcon: Fast-Fourier Lattice-Based Compact Signatures over NTRU, specification and project data. https://falcon-sign.info/falcon.pdf

[14] SQIsign team, SQIsign Round 3: Sizes and Performance, 2026. https://sqisign.org/

[15] Schwabe, Stebila, Wiggers, Post-Quantum TLS Without Handshake Signatures (KEMTLS), ACM CCS 2020. https://eprint.iacr.org/2020/534

[16] Wiggers et al., KEM-Based Authentication for TLS 1.3, draft-celi-wiggers-tls-authkem-07, work in progress, 2026. https://datatracker.ietf.org/doc/html/draft-celi-wiggers-tls-authkem-07

[17] Ducas et al., CRYSTALS-Dilithium: Digital Signatures from Module Lattices, 2017. https://cryptojedi.org/papers/dilithium-20170627.pdf

[18] Berman et al., A Note on the Hint in the Dilithium Digital Signature Scheme, 2024. https://eprint.iacr.org/2024/1660

[19] Cheon et al., HAETAE: Shorter Lattice-Based Fiat-Shamir Signatures, 2023/2024. https://hal.science/hal-04909666

[20] Boudgoust et al., Sequential Half-Aggregation of Lattice-Based Signatures, 2023. https://eprint.iacr.org/2023/159

[21] IETF PLANTS WG, Merkle Tree Certificates for Batch Certification, draft-ietf-plants-merkle-tree-certs-05, work in progress, 2026. https://datatracker.ietf.org/doc/html/draft-ietf-plants-merkle-tree-certs-05

[22] NIST, FIPS 203, Module-Lattice-Based Key-Encapsulation Mechanism Standard, 2024. https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.203.pdf

[23] NIST, FIPS 205, Stateless Hash-Based Digital Signature Standard, 2024. https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.205.pdf

[24] Lyubashevsky, Fiat-Shamir With Aborts: Applications to Lattice and Factoring-Based Signatures, ASIACRYPT 2009. https://www.iacr.org/archive/asiacrypt2009/59120596/59120596.pdf

[25] IETF, RFC 7924, TLS Cached Information Extension, 2016. https://www.rfc-editor.org/rfc/rfc7924.html

[26] IETF, RFC 9973, TLS 1.3 Extension for Using Certificates with an External PSK, 2026. https://www.rfc-editor.org/rfc/rfc9973.html

[27] Zheng et al., Optimized Vectorization Implementation of CRYSTALS-Dilithium, 2023. https://arxiv.org/abs/2306.01989

[28] IETF LAMPS WG, Composite ML-DSA for X.509 and CMS, draft-ietf-lamps-pq-composite-sigs-19, work in progress, 2026. https://datatracker.ietf.org/doc/html/draft-ietf-lamps-pq-composite-sigs-19

[29] IETF, RFC 9935, ML-KEM in X.509 Public Key Infrastructure, 2026. https://www.rfc-editor.org/rfc/rfc9935.html

[30] Yu, Jia, Wang, Compact Lattice Gadget and Its Applications to Hash-and-Sign Signatures, 2023. https://arxiv.org/abs/2305.12481