

Per-record post-quantum signatures for long-lived AI evidence: deployment

Position paper — IAB Workshop on Accelerating Deployment of Post-Quantum Authentication

John Interlante — Aleutian AI, New Jersey — jinterlante@aleutian.ai

Since March 2026 we have operated a system that captures AI system interactions—including model calls, tool invocations, retrievals, and human approvals—as append-only hash chains scoped by tenant and region. Every entry is signed synchronously at ingest with ML-DSA-65 (FIPS 204) through Google Cloud KMS, while record content is encrypted using keys established with ML-KEM-768 (FIPS 203). The design requirement that drove these choices is long-term independent verification: a record must remain verifiable offline without trusting Aleutian or requiring access to the original application infrastructure. For evidentiary or compliance use cases, both the verification requirement and the underlying evidence may outlive the originating system and may be subject to retention or legal holds lasting seven years or more.

The workload is high-volume signing of small structured records with archival verification as the primary mode, operated by one engineer. Our signing preimage is 776 bytes and the ML-DSA-65 signature over it is 3,309 bytes, so the signature is 4.3 times the size of the signing preimage. One million records comprise 0.78 GB of record bodies and 3.31 GB of signatures, compared with approximately 64 MB of raw P-256 signature material. The deployment is pre-scale, so figures here are stated per record and are reproducible using the procedure in Appendix A. This paper reports what it cost to bring ML-DSA-65 to a fail-closed write path under managed key custody, and which obstacles proved load-bearing.

Per-record signing cost is dominated by key custody rather than by the signature algorithm. We measured Google Cloud KMS AsymmetricSign against an ML-DSA-65 key and an Ed25519 key on the same keyring, in the same region, over the same connection, with calls interleaved and $n=200$ per arm. ML-DSA-65 returned p50 266.6 ms and p95 328.2 ms; Ed25519 returned p50 245.4 ms and p95 296.3 ms. The observed p50 difference was 21.2 ms, or approximately 8% of the ML-DSA-65 call. Both algorithms therefore paid roughly 245 ms for the shared managed-signing path.

Computed in-process on the same client, an Apple M4 Max running Go 1.25.6 with github.com/cloudflare/circl/sign/mldsa/mldsa65, ML-DSA-65 signing costs 0.228 ms and verification 0.029 ms. By comparison, the same client observed a p50 of 266.6 ms when signing through Google Cloud KMS, making the managed key-custody path—not ML-DSA computation itself—the dominant per-record signing cost in this deployment. At the measured verification rate, one million signature verifications require approximately 29

seconds of single-core CPU time, excluding parsing, I/O, and chain validation. Google Cloud KMS provides no batch signing operation, so each record requires one RPC; serial throughput on a single connection remained below four records per second for both algorithms.

We rejected two alternatives. Holding private keys in process removes the managed-signing round trip but abandons managed key custody. Signing Merkle roots instead of individual records reduces signature volume substantially, but changes single-record validation into a two-artifact operation requiring both the record and an inclusion proof tied to a signed root. A form of AsymmetricSign that accepts multiple messages against one key version would remove the largest structural constraint we encountered without requiring new cryptography. We would also value workshop discussion of how batch-signed records and inclusion proofs should be represented for long-lived independent verification. This is the design decision in our deployment for which we have the least confidence, and the pressure toward batching comes primarily from signature size.

We use X-Wing for content-key establishment, combining ML-KEM-768 with X25519 so that compromise of either component alone does not compromise the resulting shared secret. This created an assurance problem rather than a cryptographic one: ML-KEM is covered by FIPS 203, but in our validation context we could not establish an equivalent approved key-establishment claim for the combined hybrid construction. Moving the key into an HSM would have improved key protection, but it would not have changed the validation status of the hybrid construction.

Our ML-DSA-65 signing keys are exposed by Google Cloud KMS at the SOFTWARE protection level. We therefore kept the deployment at software protection and based our assurance statement on the standardized algorithms themselves while explicitly documenting the module-level limitation. The deployment lesson is that the assurance path for a hybrid construction was substantially less clear than the path for a single approved scheme.

Every interoperability failure in this deployment was a canonicalization failure. Our verifiers are implemented independently in Go, Python, and JavaScript and must construct identical signed byte sequences and return equivalent verification results. The ML-DSA-65 primitives interoperated on the first attempt; the bytes supplied to them did not. Failures involved JSON key ordering, Unicode normalization, numeric representation, and timestamp precision.

We stopped canonicalizing JSON for records and adopted a typed, length-prefixed TLV encoding: u32 big-endian length prefixes, NFC normalization applied before length computation, fixed field order, and unconditional emission of every field with a zero-length prefix when absent. Encoding and validation cost 0.004 ms per record. Two consequences are worth passing on. First, our anchors still use sorted-key canonical JSON because they predate the change; altering that representation would invalidate historical signatures. A canonical form, once signed into an archive, becomes part of the protocol. Second, changing that form requires an explicit protocol transition: a new domain-separation

prefix, a version field on which verifiers dispatch, a lockstep release of independent verifier implementations with cross-language test vectors, and a shadow period before producers emit the new version. We have canonical v4 specified but not yet deployed for this reason. We have found little guidance on sequencing this kind of canonical-form migration across independent verifier implementations.

The archival risk we encountered concerns verifier semantics as much as key lifetime. Records in this system can be erased according to the application’s retention rules. On erasure, the content hash is destroyed and replaced while the chain hash—the value over which anchors were signed—is retained, so a verifier must not attempt to recompute an erased record’s original content hash. Three independently written verifier SDKs implemented this rule incorrectly in the same way and reported erased chains as tampered. While writing an open-source verifier this year, we reproduced the same defect by hand even with the intended rule available for reference.

This suggests that long-term validation depends not only on retaining verification keys but also on retaining correct semantics for historical record states. A verification key available in 2040 is insufficient if the verifier available in 2040 cannot interpret a record state introduced in 2027. Because our signatures cover a content hash rather than the content itself, destruction of decryption material can render content unrecoverable while leaving the signed record and its position in the chain verifiable; the erasure event is itself represented as a signed record. Preserving that separation between content lifecycle and integrity lifecycle is therefore a requirement of our record design.

What would accelerate deployment. A batch-signing operation in managed KMS offerings. An assurance path for hybrid constructions that is legible to a small operator without validation counsel. Parity of hardware protection for FIPS 203 and FIPS 204 keys across providers. Documented message-shape conventions for post-quantum signing APIs. Canonical-encoding guidance for structured records, including migration sequencing across independent verifiers. Archival guidance covering verifier compatibility and record-state semantics alongside key retention.

The cryptographic primitives presented the fewest difficulties in this deployment. The obstacles were in managed key-custody surfaces and in the available guidance for signing and preserving structured data over long periods.

I will not be able to attend in person. I would welcome remote participation if the Program Committee offers it, and I consent to publication of this paper on the workshop’s Datatracker page without restriction.

Appendix A — Measurement procedure

The procedure is included so that the central measurement can be checked or contradicted.

Setup. Ingest signs each record through a per-tenant KMS-resolver-backed signer with no process-local fallback and one AsymmetricSign RPC per record. The live canonical encoder yields 776 bytes for a fully populated record; a 24-byte domain-separation prefix gives the 800-byte payload used in every measurement. Local primitive floor, cloudflare/circl, n=2000, Apple M4 Max, Go 1.25.6: sign 0.228 ms, verify 0.029 ms, canonical encode and validate 0.004 ms. Enumerating the estate confirms that all PQ_SIGN_ML_DSA_65 keys report protection level SOFTWARE and locates an enabled EC_SIGN_ED25519 key on the same keyring to serve as a control.

A discarded measurement. Signing 60 times against the ML-DSA-65 key alone gave p50 302 ms, 1,326 times the in-process primitive cost. We do not attribute that ratio to post-quantum cryptography: it largely reflects the managed signing path and client-to-region round trip that any algorithm using the same service would pay.

Method. We signed the same payload alternately against the ML-DSA-65 key and the Ed25519 control, one immediately after the other, on one client and one connection, with the same project, location, and protection level. Alternation reduces the effect of network variance and server load shared by both arms and lets us estimate the algorithm-associated difference. Both arms were warmed once before timing. Each response's signature length was asserted—3,309 bytes versus 64 bytes—so a misrouted key version could not pass unnoticed.

Result and limitations. At n=50, p50 was 293.2 ms versus 278.2 ms, a delta of 15.0 ms (5.1%); at n=200, 266.6 ms versus 245.4 ms, a delta of 21.2 ms (7.9%). Both arms improved as the connection warmed while the delta retained its sign and order of magnitude. At n=50 alone, a 15 ms delta against a 400 ms p95 is not separable from noise, so n=200 is the reported figure. Measurements were taken from a laptop, so absolute round-trip latency is an upper bound and would likely be lower from a co-located service; the one-RPC-per-record structure does not depend on client location. The measurement is single-client, single-connection, and serial, so it describes per-record cost rather than achievable aggregate throughput. It was taken in a development environment using keys with the same algorithm, protection level, and configuration as production, and represents n=200 per arm on one occasion rather than a longitudinal sample.