

Per-record ML-DSA-65 in a synchronous write path

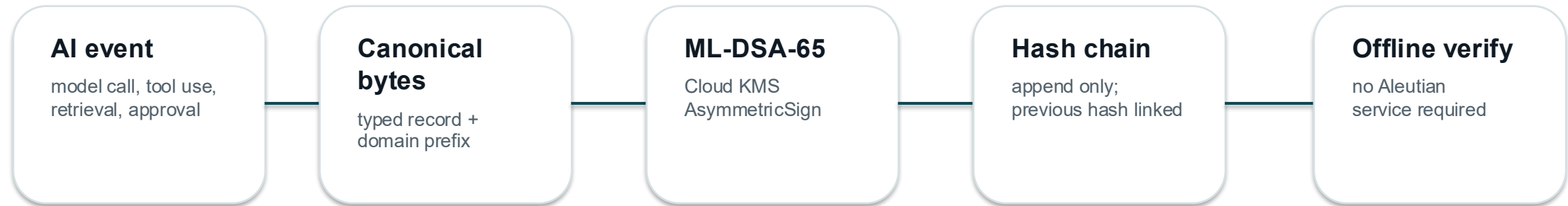
What we built, how we measured it, and the engineering trade-offs we hit

John Interlante · Aleutian AI

IAB Workshop on Accelerating Deployment of Post-Quantum Authentication

Prague · October 2026

The system encrypts the payload, then signs each canonical evidence record before persistence



Encryption: symmetric payload encryption; ML-KEM-768 protects the data-encryption key. Signature: ML-DSA-65 covers the canonical stored record.

- Signing is synchronous and fail-closed: an unsigned evidence record is not written.
- Per-record signing reached production on 2026-08-13. (Correction to submitted paper.)
- Per-record signatures were chosen so a single archived record can be checked without an inclusion proof and signed batch root.

We separated three questions: primitive cost, managed-call cost, and algorithm-specific cost

A · Local primitive

800-byte message
cloudflare/circl ML-DSA-65
fresh keys across runs
sign + verify timing

B · Client A/B

One gRPC connection
300 rounds
randomized order per round:
ML-DSA sign · Ed25519 sign ·
GetKeyVersion
all signatures verified

C · Server-side

Cloud Monitoring
AsymmetricSign request latency
single-algorithm, minute-aligned
blocks
production 30-day mean also
checked

Same 800-byte payload shape throughout: 776-byte canonical record + 24-byte domain-separation prefix.

RESULTS

The rerun did not show a measurable ML-DSA-specific penalty through KMS

~0.21 ms

ML-DSA-65 local sign
hedged/randomized mode

102.3 ms

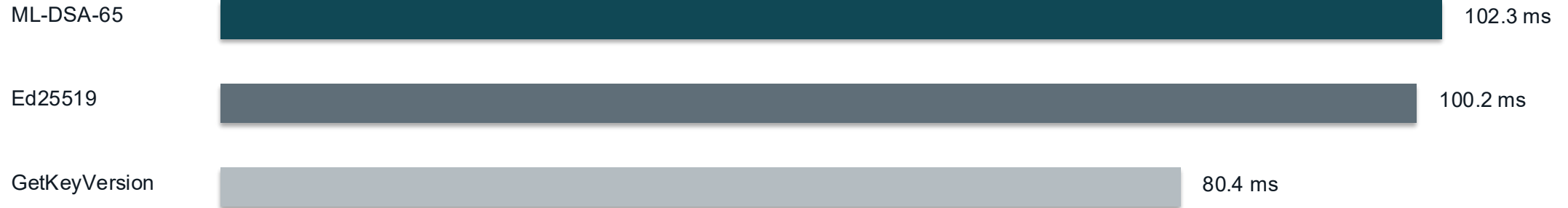
KMS ML-DSA-65 client p50

100.2 ms

KMS Ed25519 client p50

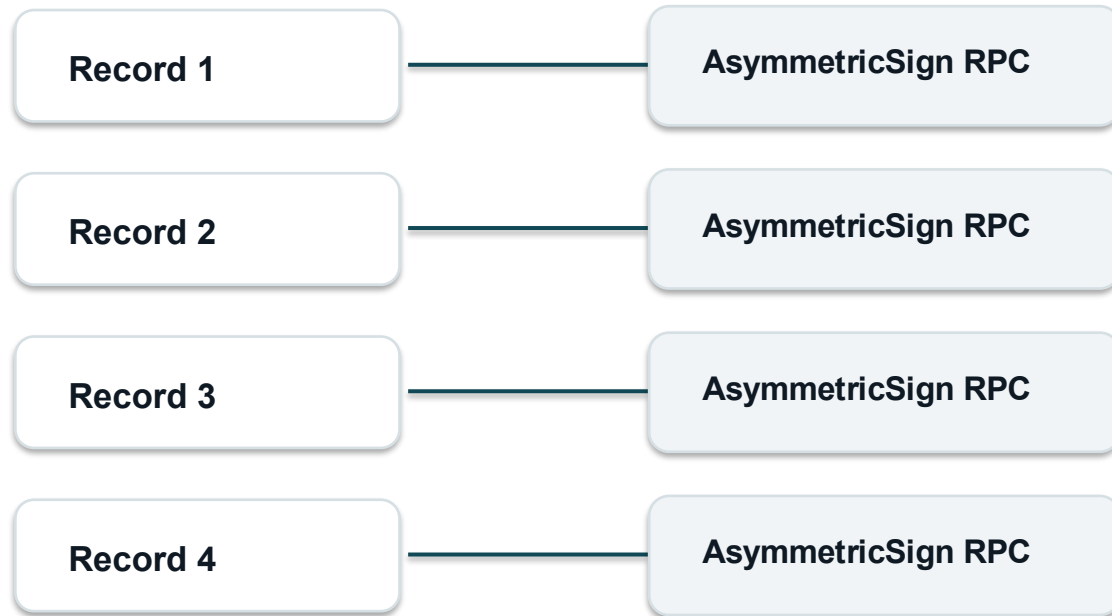
+1.2 ms

paired median difference
95% CI -1.5 to +4.9



Correction to submitted paper: the earlier ~8% difference did not reproduce when call order was randomized; the original harness always called ML-DSA first.

Per-record signing means one managed-signing operation per record



8.6 signs/s **81.1 signs/s**

serial, one connection

8 workers, one connection

Corrected laptop client test — not a provider throughput limit.

- Production uses concurrency to raise throughput.
- Concurrency does not remove the per-record RPC or the per-record service call.
- No multi-message AsymmetricSign operation was used in this architecture.

The clearest PQ-specific pressure we observed was signature size

776 B

canonical record

800 B

signing preimage
+24 B domain prefix

3,309 B

ML-DSA-65 signature

4.3×

signature / record

Per-record signature

Record + signature, with verification key/trust material.
Higher signature volume.

Merkle-root signature

Fewer signatures.
Single-record checking also needs an inclusion proof and signed root.

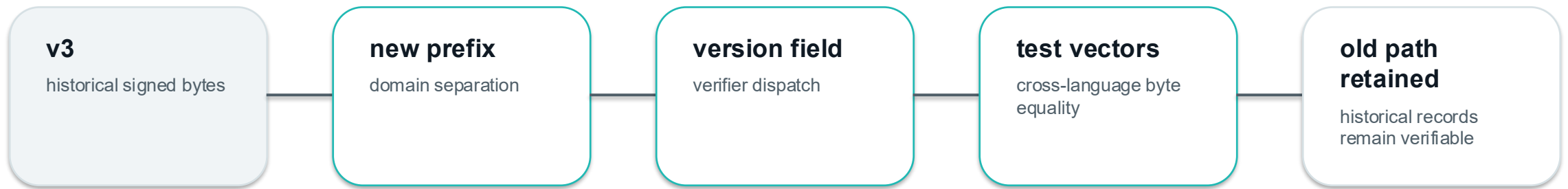
At our current volume this is not a storage problem; it is an architectural trade-off.

The interoperability bugs we hit were in the signed bytes, not in ML-DSA



Records moved to a typed, length-prefixed TLV representation; anchors remain on their historical JSON representation.

Changing a signed representation required an explicit compatibility path



- The record format and anchor format are separate.
- The next anchor format is specified but not deployed.
- This slide describes our migration mechanics, not a claim that this is the only correct approach.

Erasure exposed a verifier-state bug that key retention alone would not prevent



- Go, Python, and JavaScript verifier implementations initially reported this valid erased state as tampering.
- The fix was to encode and test the state transition explicitly.
- We now treat historical record-state semantics as part of verifier compatibility.

Current design choices and the questions we still have

Current choices

- Per-record ML-DSA-65 signatures
- Fail-closed synchronous signing
- Cloud KMS custody
- Typed TLV for records
- Separate historical anchor encoding
- Versioned verifier behavior

Alternatives we considered

- Merkle-root signing
- Process-local private keys
- Different canonical encodings
- More aggressive batching / concurrency

Questions for discussion

- Multi-message managed signing?
- How do existing receipt/proof formats map to this archival case?
- Migration patterns for signed encodings?
- What verifier state should an archive carry?

Reference implementation: github.com/aleutian-ai/proof · production system described here is a separate codebase

Measurement details

Test	Payload	Path	Method
Local	800 B	circl ML-DSA-65	fresh keys; sign/verify
Client A/B	800 B	Cloud KMS	300 randomized rounds; 1 connection
Baseline	n/a	GetCryptoKeyVersion	non-signing RPC comparator
Server	800 B	Cloud Monitoring	minute-aligned single-algorithm blocks

- All returned signatures in the corrected A/B run were length-checked and verified locally.
- The corrected run also exposed a call-position effect that likely explains the submitted-paper delta.

Submitted-paper corrections to state explicitly if asked

Timeline Development began in March; per-record signing reached production 2026-08-13.

KMS delta The submitted ~8% ML-DSA penalty did not reproduce with randomized order.

Absolute KMS latency The original ~250 ms figures included a large client-network component.

Control keyring The two benchmark keys were in the same project/region, but different keyrings.

Throughput Do not quote <4 signs/s as a system limit; corrected serial test was 8.6/s and concurrency was much higher.