WP-032
June 2026
12 sections

POST-QUANTUM CRYPTOGRAPHY & AUDIT ARCHITECTURE

Sublinear Post-Quantum Attestation: Merkle-Anchored Audit Roots with ML-DSA-65 for Long-Lived Records

One ML-DSA-65 signature on a Merkle root covers a million records. O(log n) inclusion proofs replace linear signature sets. The mathematics of post-quantum attestation at scale.

AffixIO Research | June 2026 | Download PDF

ABSTRACT

The NIST post-quantum cryptography standard ML-DSA (FIPS 204) produces signatures of approximately 3,293 bytes at the NIST level 3 parameter set (ML-DSA-65). Applied naively, post-quantum attestation of n records requires n signatures and O(n) storage and verification time. This paper presents the mathematical optimisation that reduces both to sublinear scale: a Merkle tree construction in which records are organised into a binary hash tree, the root is signed once with ML-DSA-65, and any individual record is verified with an inclusion proof of O(log n) hash values. At one million records, the scheme replaces 3.14 GB of per-record signatures with a single 3,293-byte root signature plus

inclusion proofs of approximately 320 bytes each, reducing attestation storage by over 99%. We formalise the security reduction of the combined scheme to two independent quantum-hard assumptions (SHA-256 collision resistance and ML-DSA-65 unforgeability under MLWE), analyse the interaction with regulatory retention requirements, present batching strategies for high-throughput environments, and describe the incremental Merkle tree construction for append-only audit logs. The architecture is used in AffixIO's audit infrastructure; this paper documents the mathematical foundations and design rationale without disclosing implementation-specific internals.

CONTENTS

SECTION 01

The Attestation Scale Problem

Post-quantum cryptography solves a real and urgent problem: digital signatures and key exchange based on classical hardness assumptions (discrete logarithm, integer factorisation) are broken in polynomial time by Shor's algorithm on a sufficiently powerful quantum computer. Organisations that generate audit records, compliance evidence, or long-lived

cryptographically signed data today face a specific risk: a quantum adversary who captures signed records now can, in the future, forge counterfeit records that appear to have been signed before the quantum attack, retroactively falsifying historical evidence. This is the long-lived record variant of the harvest-now-decrypt-later (HNDL) problem.

The natural response is to adopt a NIST post-quantum signature scheme. ML-DSA (Module-Lattice-based Digital Signature Algorithm, NIST FIPS 204), formerly known as Dilithium, is the primary general-purpose post-quantum signature standard. At the NIST level 3 parameter set (ML-DSA-65), ML-DSA provides 128-bit post-quantum security, is designed for high-volume signing, and produces signatures of 3,293 bytes with public keys of 1,952 bytes.

The problem is scale. Consider a system that generates one million audit records per day, a modest throughput for an AI governance platform, a financial services transaction processor, or a healthcare record system. Signing each record with ML-DSA-65 produces:

- 3,293 bytes per signature × 1,000,000 records = 3.14 GB of signatures per day
- After 365 days: 1.14 TB of signatures
- After 7 years (financial services retention minimum): 8.0 TB of signatures
- After 10 years (EU AI Act Article 12 retention): 11.4 TB of signatures

This is not a storage problem that can be ignored and fixed later; it grows indefinitely. Worse, verification also scales linearly: checking whether n records are all authentically signed requires verifying n ML-DSA-65 signatures, each of which involves a matrix-vector multiplication and a polynomial norm check. At one million records, a single verification pass requires significant compute time even on server hardware. For retrospective compliance audits covering years of records, linear verification becomes impractical.

The solution is not to sign fewer records. Regulatory requirements mandate continuous audit trails; selective signing creates gaps that auditors will question. The solution is to change the attestation structure so that one signature covers many records without sacrificing the ability to verify any individual record. That structure is a Merkle hash tree with a signed root.

SECTION 02

ML-DSA-65: Properties and Costs

ML-DSA-65 is the NIST level 3 parameter set of ML-DSA (FIPS 204), targeting 128 bits of post-quantum security. Its core security assumption is Module Learning With Errors (MLWE): the hardness of distinguishing a module lattice sample from uniform, and the hardness of finding short vectors in module lattices. No known quantum algorithm achieves better than exponential time against MLWE; the current consensus expectation is that ML-DSA-65 remains secure against both classical and quantum adversaries at the stated security level.

Parameter Summary

PARAMETER	ML-DSA-44 (LEVEL 2)	ML-DSA-65 (LEVEL 3)	ML-DSA-87 (LEVEL 5)
Post-quantum security (bits)	~100	~128	~168
Public key size	1,312 bytes	1,952 bytes	2,592 bytes
Signature size	2,420 bytes	3,293 bytes	4,595 bytes
Signing throughput (est., 2025 hardware)	~40,000/s	~27,000/s	~18,000/s
Verify throughput (est., 2025 hardware)	~90,000/s	~60,000/s	~40,000/s

ML-DSA-65 is the appropriate choice for attestation anchoring at regulatory security levels: it is the CNSA 2.0 specified signature algorithm for classified US government systems, and it is referenced in BSI TR-02102-1 and ETSI TS 119 312 for European regulated use cases. The signing and verification throughputs are sufficient for high-volume root signing (a single thread can sign thousands of Merkle roots per second) but are not intended for per-record signing at high throughput without batching.

The Signing Cost Problem Stated Precisely

Let n be the number of audit records in a batch period. Per-record signing has the following costs:

- **Signing cost:** n invocations of ML-DSA-65 Sign, each taking approximately 37 μs on a 2025-generation server CPU.
- **Storage cost:** n × 3,293 bytes of signature storage, plus n × |record| bytes of record storage.
- **Verification cost:** for a verifier checking k out of n records, k ML-DSA-65 Verify invocations, each requiring the corresponding public key.

All three costs are O(n). The Merkle-anchored scheme below reduces the signing cost to O(1) (one root sign per batch), the signature storage cost to O(1) (one root signature per batch), and the verification cost for an individual record to O(log n) (one path through the tree).

SECTION 03

Merkle Trees: Structure and Security

A Merkle tree (Ralph Merkle, 1979) is a complete binary tree in which: each leaf node contains the cryptographic hash of a data record; each internal node contains the hash of the concatenation of its two children; and the root node is the single hash that commits to the content of all leaves. Any alteration to any leaf changes all ancestor hashes up to and including the root.

Formal Definition

> MERKLE TREE
>
> Let H: {0,1}* → {0,1}^λ be a collision-resistant hash function with output length λ bits. Let records r_1, r_2, ..., r_n be a sequence of data items with n = 2^k for some k (padded if necessary). The Merkle tree T over (r_1, ..., r_n) with hash H is defined as follows. The leaf nodes are L_i = H(r_i) for i = 1, ..., n. The internal nodes at height h are N_{h,j} = H(N_{h-1, 2j-1} || N_{h-1, 2j}) for j = 1, ..., n/2^h. The root is Root(T) = N_{k, 1}. An inclusion proof for leaf i is the sequence of sibling hashes on the path from L_i to the root: π_i =

> (sib_{k-1}, sib_{k-2}, ..., sib_0) where sib_l is the sibling of the node on the path at height l.

Verification of an inclusion proof proceeds by computing the path hash: starting from H(r_i), repeatedly hash with the sibling at each level, and check that the result equals Root(T). This requires exactly k = log_2(n) hash evaluations.

Security Property: Binding

The binding property of a Merkle tree states that it is computationally infeasible for an adversary to produce two different sequences of records that have the same root, under the assumption that H is collision-resistant. More precisely: if an adversary can find records r'_j ≠ r_j and an inclusion proof π'_j such that the proof verifies correctly for r'_j against Root(T), then the adversary has found a collision in H (either at the leaf level or at some internal node).

This security reduction is standard and well-understood. The binding property of the Merkle tree is exactly as strong as the collision resistance of the underlying hash function. For SHA-256, the best known classical attack finds collisions in O(2^128) time; the best known quantum attack (Grover's algorithm applied to the birthday problem) runs in O(2^{85}) time. For SHA3-256, the security analysis is similar. Both provide substantially more than the 128-bit post-quantum security of ML-DSA-65, so the Merkle tree binding is not the security bottleneck in the combined scheme.

SECTION 04

The Sublinear Construction

The sublinear post-quantum attestation scheme for a batch of n records proceeds as follows:

Construction

1. **Hash records:** For each record r_i, compute the leaf hash L_i = H(r_i || metadata_i), where metadata_i includes at minimum a timestamp and a sequence number to prevent reordering attacks.
2. **Build tree:** Construct the Merkle tree T over (L_1, ..., L_n). If n is not a power of 2, pad with leaves H("empty" || i) to the next power of 2.
3. **Sign root:** Compute Root(T) and sign it with ML-DSA-65: σ = ML-DSA.Sign(sk, Root(T) || batch_metadata), where batch_metadata includes the batch identifier, timestamp range, and record count.
4. **Store:** Retain the signed root (σ, pk, Root(T), batch_metadata) and the records r_i. Optionally store pre-computed inclusion proofs for frequent access patterns.
5. **Publish:** Optionally publish Root(T) and σ to an immutable registry (public blockchain, regulatory record, timestamped notarisation service) for independent verification.

Verification of an Individual Record

A verifier who holds record r_i and wishes to confirm it was included in the attested batch:

1. Obtain the inclusion proof π_i = (sib_{k-1}, ..., sib_0) from the record holder or a proof server.
2. Compute the path hash: start with H(r_i || metadata_i), and repeatedly hash with siblings to reconstruct Root'.
3. Verify the ML-DSA-65 signature: ML-DSA.Verify(pk, Root' || batch_metadata, σ) = accept.
4. If both the path reconstruction yields Root' = Root(T) and the signature verifies, the record is authenticated.

The verifier's computation requires log_2(n) hash evaluations and one ML-DSA-65 verification. For n = 1,000,000, this is approximately 20 hash evaluations and one signature verification. The verifier does not need any other records from the batch; they do not need the signing key; and they do not need any ongoing connection to the signing infrastructure.

Delegation of Inclusion Proof Serving

Inclusion proofs can be served by any party in possession of the tree. They are not secret and do not need to be generated by the key holder. A proof server, a regulatory archive, or a peer record-holder can provide the proof. The security of the verification does not depend on the trustworthiness of the proof server: if the proof server provides a fraudulent inclusion proof, the path reconstruction will not produce the correct root, and the ML-DSA-65 signature verification will fail.

SECTION 05

Security Reduction

The security of the sublinear ML-DSA-65 Merkle attestation scheme reduces to two independent cryptographic assumptions.

> **THEOREM: SECURITY OF MERKLE-ML-DSA ATTESTATION**
>
> Let H be a (q_H, ε_H)-collision-resistant hash function and let ML-DSA-65 be an (q_S, ε_S)-unforgeable signature scheme under MLWE. Then the Merkle-ML-DSA attestation scheme is (q_H + q_S, ε_H + ε_S)-secure in the following sense: no adversary making at most q_H hash queries and q_S signing queries can produce a valid attestation (π, r') for a record r' that was not included in the original batch, except with probability at most ε_H + ε_S.

The proof proceeds by case analysis. An adversary who produces a valid attestation for r' ∉ {r_1, ..., r_n} must either: (a) find a valid inclusion proof π' such that the path reconstruction from H(r' || metadata') yields Root(T), which requires finding a collision in H (a preimage at some internal node); or (b) produce a valid ML-DSA-65 signature on a different root Root' ≠ Root(T), which requires forging an ML-DSA-65 signature. Case (a) breaks collision resistance of H; case (b) breaks unforgeability of ML-DSA-65. Neither is achievable by a quantum adversary under current assumptions, since SHA-256/SHA3-256 retain substantial security margins against quantum attack, and ML-DSA-65 is designed to resist quantum algorithms at the 128-bit post-quantum security level.

Independence of Assumptions

The two security assumptions are structurally independent: breaking ML-DSA-65 (by solving MLWE) does not help find collisions in SHA-256, and finding SHA-256 collisions does not help forge ML-DSA-65 signatures. This independence means the scheme does not have a single point of cryptographic failure: an adversary must simultaneously break both hash collision resistance and signature unforgeability to forge an attestation, which is not achievable with any known quantum or classical technique.

Forward Security of Historical Records

A key property for long-lived records is forward security: records attested in the past remain authentic even if the signing key is later compromised, provided the root signature was made before the compromise and the root was published or notarised independently. If the signing key sk is compromised after time T, an adversary can sign new Merkle roots but cannot retroactively change roots that were signed before T, because those roots are bound to their batch contents by the hash tree. The published root signatures serve as the historical record; their authenticity depends only on the ML-DSA-65 unforgeability at the time of signing, not on the ongoing security of the key.

This forward security property does not extend to records within a batch whose root has not yet been published: an adversary with the signing key can create fraudulent batches and sign their roots. The architectural response to this is root publication as close to signing time as possible, using an append-only external registry (public blockchain, trusted timestamping authority, regulatory record) that makes historical root tampering detectable.

SECTION 06

Storage and Verification Costs

The following table quantifies the storage and verification costs of the sublinear scheme compared with per-record ML-DSA-65 signing, at representative scales.

METRIC	PER-RECORD ML-DSA-65	MERKLE-ML-DSA-65 (SUBLINEAR)	REDUCTION
Signatures stored per 1K records	3.14 MB	3,293 bytes (root) + 0 (proofs on demand)	>99.9%
Signatures stored per 1M records	3.14 GB	3,293 bytes (root) + proofs on demand	>99.9%
Signatures stored per 1B records	3.14 TB	3,293 bytes (root) per batch	>99.9%
Inclusion proof size (1M records, k=20)	N/A	20 × 32 bytes = 640 bytes	N/A
Verify one record out of 1M	1 ML-DSA verify	20 SHA-256 + 1 ML-DSA verify	Equal (one verify each)
Verify all 1M records	1M ML-DSA verifies (~16,667 seconds)	1M × 20 SHA-256 + 1 ML-DSA verify (~5 seconds for hashes)	>99.9% time reduction
Signing cost per batch of 1M	1M × 37 µs = ~10 hours	1 × 37 µs (root signing) + tree build time	>99.9%
Tree build time (1M records)	N/A	~2M SHA-256 hashes ≈ 0.3 seconds	N/A

The tree build time is linear in n (O(n) hash evaluations to build the tree) but uses only SHA-256 hashing, not ML-DSA signing. SHA-256 throughput on modern hardware is approximately 500 MB/s to several GB/s depending on hardware acceleration, making tree construction fast even for large batches. The single ML-DSA-65 root signing takes approximately 37 µs, regardless of batch size.

Pre-Computed vs On-Demand Proofs

Inclusion proofs can be stored at generation time (adding O(n log n) bytes of proof storage) or recomputed on demand from the stored tree (requiring the tree to be retained). The choice depends on the access pattern:

- **High query frequency:** pre-compute and store proofs alongside records. Storage cost is n × k × 32 bytes = n × log_2(n) × 32 bytes. For 1 million records, this is 640 MB, compared with 3.14 GB for per-record signatures.
- **Low query frequency / regulatory audit only:** store the tree and recompute proofs on demand. Storage is the tree itself: 2n × 32 bytes = 64 MB for 1 million records.
- **Minimal storage:** store only records and the signed root. Inclusion proofs can be recomputed from scratch when needed, provided the records are available. This is appropriate when the records are already stored for other reasons and proof generation latency is acceptable.

SECTION 07

Incremental Trees for Append-Only Logs

A standard Merkle tree is a static structure: it is built once over a fixed set of records and its root changes if any record is added. Audit logs are inherently append-only: new records are continuously added. The sublinear attestation scheme must accommodate this requirement without requiring the tree to be rebuilt from scratch on each addition.

Batch Approach

The simplest approach is batching: records accumulate in a pending list until a batch period expires (for example, every 60 seconds, every 1,000 records, or whenever a configurable threshold is met). At the end of each batch period, a tree is built over the pending records and the root is signed. Records from different batches are independent: each has its own signed root, and verification of a record from batch b requires only the root of batch b, not the roots of any other batch.

The choice of batch period involves a trade-off:

- **Shorter batches (e.g., every second):** lower latency between record creation and attestation; more roots to store and publish; smaller trees with shorter inclusion proofs.

- **Longer batches (e.g., every hour or day):** fewer roots; larger trees; longer latency between record creation and attestation. For records that need to be attested immediately on creation, this is unsuitable.

For most regulatory compliance use cases, a batch period of 60 seconds to 5 minutes provides latency that is acceptable (records are attested within minutes) while keeping the number of roots manageable (12 to 1,440 roots per day).

Incremental Merkle Tree (Continuous Append)

For applications requiring continuous append with verifiable history at any point in time, an incremental Merkle tree (also called a "persistent" or "running" Merkle tree) maintains a set of partial subtree roots that can be combined to compute the full root at any point. This is the structure used in certificate transparency logs (RFC 6962) and append-only blockchains.

In an incremental tree, after n insertions, the structure maintains at most log_2(n) partial roots (one for each bit of n that is set to 1). Inserting a new leaf requires at most log_2(n) hash computations. The full root at any point can be computed in O(log n) time. Proofs of inclusion and proofs of consistency (that a later tree is an extension of an earlier tree) are both O(log n) in size.

The incremental tree is appropriate for situations where the full tree root needs to be updated frequently and signed, while maintaining verifiable consistency with all previous signed states. The periodic ML-DSA-65 signing of the incremental root provides a post-quantum authenticated record of the entire log history up to the signing time.

SECTION 08

Batching Strategies

The efficiency of the sublinear scheme depends on choosing appropriate batching parameters for the deployment context. The following strategies address different throughput and latency requirements.

Time-Based Batching

Records are batched by time interval (T seconds). Every T seconds, all records received during the interval are assembled into a Merkle tree and the root is signed. This is simple to implement and predictable, but batch sizes vary with load: during peak periods a batch may contain millions of records; during quiet periods a batch may contain only a few. Small batches lose little efficiency (the tree is small), but high-frequency small batches generate many roots. The recommended default is T = 60 seconds for most compliance applications, producing at most 1,440 roots per day.

Count-Based Batching

Records are batched until n_max records have accumulated. This produces uniform batch sizes and predictable tree depths (always log_2(n_max) levels), but variable latency: during quiet periods a batch may not fill for hours. For applications with SLA requirements on attestation latency, count-based batching is combined with a maximum time threshold: the batch closes when it either reaches n_max records or T seconds have elapsed, whichever comes first.

Hierarchical Batching

For very high throughput (millions of records per second), a hierarchical batching structure can be used. Within each second, records are assembled into a second-level tree with a local root. Within each minute, second-level roots are assembled into a minute-level tree. The minute-level root is signed with ML-DSA-65. This creates a hierarchy of O(log(total_records)) depth, with O(log n_total) hash evaluations for any cross-level inclusion proof. The structure is equivalent to a deep Merkle tree but allows the signing to happen at a coarser time granularity than the record insertion rate.

Cross-Batch Linking

For applications requiring a verifiable chain of history across batches (that batch j is a continuation of batch j-1), the batch_metadata field included in each root signature should include the root of the previous batch: batch_metadata_j = (j, t_start, t_end, n_records, Root(T_{j-1})). This creates a hash chain over signed roots, analogous to a blockchain but without

distributed consensus. A verifier who holds the genesis root can verify the full chain of signed roots in O(batches) time; an auditor can verify any individual record in O(log n) time given the relevant batch root.

SECTION 09

Regulatory Retention Compatibility

The sublinear attestation scheme is designed to be compatible with regulatory data retention requirements, including those that predate post-quantum cryptography standards. The following analysis covers the major regulatory frameworks that impose retention requirements on audit records.

EU AI Act (FIPS Reg. 2024/1689)

Article 12 requires providers of high-risk AI systems to maintain logs for at least 10 years from the date the system is placed on the market or put into service. The record of AI decisions must be sufficient to reconstruct the reasoning and identify the data used. Merkle-anchored attestation satisfies Article 12 in the following manner: each AI decision record is included in a Merkle batch; the signed root and the record itself are retained for 10 years; inclusion proofs are either stored or recomputable from the retained tree data. The ML-DSA-65 root signature over a SHA-256 Merkle root is expected to remain unforgeable for at least 10 years under current post-quantum security estimates.

DORA (EU 2022/2554)

DORA requires ICT-related incident records to be retained for a minimum of 5 years. For financial institutions subject to DORA, the Merkle-anchored scheme satisfies this requirement: signed roots for each day's or hour's batch of ICT governance records provide tamper-evident attestation of the record set for the retention period. The root publication step (Section 10) ensures that signed roots are independently verifiable even if the institution's internal infrastructure is later compromised.

HIPAA (US, 45 CFR Part 164)

HIPAA requires covered entities to retain security and audit documentation for 6 years from the date of creation or the date when last in effect, whichever is later. The Merkle-anchored scheme provides an additional privacy benefit in the healthcare context: the signed root commits to the existence of records without revealing their content. A verifier who holds only the signed root and a specific inclusion proof can verify the authenticity of one record without gaining access to any other record in the batch.

Financial Services (MiFID II, EMIR, SEC 17a-4)

MiFID II (EU) requires records of all orders and transactions to be retained for 5 to 7 years. SEC Rule 17a-4 (US) requires broker-dealer records to be retained for 3 to 6 years in write-once, non-erasable format. The Merkle-anchored scheme is compatible with write-once storage requirements: the signed root and the leaf records are written once at batch close and never modified. Subsequent verification does not require modifying any stored data.

REGULATION	RETENTION MINIMUM	KEY REQUIREMENT	MERKLE-ML-DSA COMPATIBILITY
EU AI Act Article 12	10 years	Tamper-evident AI decision logs	Yes: ML-DSA-65 root + hash tree
DORA Article 12	5 years	ICT incident records	Yes: signed batch roots
HIPAA 45 CFR 164	6 years	Security audit documentation	Yes: privacy-preserving batch proofs
MiFID II RTS 22	5-7 years	Transaction order records	Yes: batch root per trading session
SEC Rule 17a-4	3-6 years	Write-once, non-erasable	Yes: WORM-compatible batch design
NIS2 Annex I/II	Variable	Incident logging for critical entities	Yes: continuous incremental tree

SECTION 10

Root Publication and Anchoring

The forward security argument in Section 5 requires that signed roots be published to an independent, append-only record before their authenticity can be relied upon for regulatory purposes. Publication prevents retroactive substitution of a fraudulent root in the event that the signing key is later compromised. Several publication mechanisms are available, each with different trust properties.

Public Blockchain Anchoring

Publishing the signed root (or its hash) in a transaction on a public blockchain (Ethereum, Bitcoin) creates an immutable, timestamped record. The cost is the transaction fee; the latency is the block time of the chosen chain (seconds to minutes). The trust model is the security of the blockchain's consensus mechanism, which is independent of the organisation's own infrastructure. For AffixIO, this is one of several available anchoring mechanisms.

A potential concern with blockchain anchoring and PQC is the quantum vulnerability of blockchain consensus and transaction signing (which typically use ECDSA). However, the security property being relied upon here is the immutability of the historical ledger record, not the validity of the transaction's own signature. Even if ECDSA-signed blockchain transactions are eventually forgeable by a quantum adversary, the ledger history (showing that a transaction recording root R was included in block B at time T) is protected by the hash chaining of the blockchain's block structure, which relies on SHA-256 or similar hash functions rather than ECDSA.

Trusted Timestamping (RFC 3161)

RFC 3161 timestamping authorities (TSAs) issue cryptographically signed timestamps over submitted data hashes. The timestamp's signature algorithm can be ML-DSA-65 if the TSA supports NIST PQC algorithms (several TSAs have added PQC support in 2025 and 2026). A trusted timestamp over

H(Root || σ) provides an independently verifiable record that the signed root existed at the stated time. This is a lower-cost alternative to blockchain anchoring for organisations with existing relationships with accredited TSAs.

Regulatory Registry Publication

Some regulatory frameworks are establishing or planning registries for AI governance records. EU AI Act Article 71 establishes a public EU database of high-risk AI systems; proposals for extending this to audit root registration are under discussion. If such registries become available, they provide the most direct regulatory compliance value: the regulatory authority itself holds the historical record of signed audit roots.

AffixIO Merkle Anchoring

AffixIO's audit infrastructure includes a Merkle-anchored record service in which batched audit roots are signed with ML-DSA-65 and published to an external anchoring service at configurable intervals. The anchoring provides a layer of protection against internal infrastructure compromise: even if AffixIO's internal systems were compromised, the publicly anchored roots cannot be retroactively altered, and the records' authenticity against those roots remains verifiable. The specific anchoring mechanism and the internal batching parameters are not disclosed in this public documentation.

SECTION 11

Limitations and Open Problems

Tree Reconstruction Dependency

The minimal-storage variant of the scheme (storing only records and the signed root) requires that the records be available to recompute inclusion proofs when needed. If records are lost or deleted, inclusion proofs cannot be computed even though the root remains valid. This creates a dependency between record retention and proof computability. The architectural response

is to separate record storage from proof storage: records may be archived in cold storage with strict retention controls, while inclusion proofs (which are not sensitive) can be stored in more accessible warm or hot storage.

Non-Power-of-Two Batches

When batch sizes are not powers of two, the tree must be padded. Padding choices affect the proof sizes for records near the boundary and can, if not implemented carefully, create distinguishability between padded and non-padded positions in the tree. Standard approaches include padding with H("empty" || position) or using a sparse Merkle tree construction that avoids explicit padding. The choice of padding scheme should be documented in the implementation specification and consistently applied.

Tree Structure as Metadata

The structure of an inclusion proof (the specific sibling hashes on the path) reveals the position of the record within the batch. In most compliance contexts this is not sensitive. However, if the position of a record within a batch reveals information (for example, record position correlates with submission order which correlates with identity), the tree structure itself leaks information. Mitigations include shuffling records within a batch before tree construction, or using position-hiding Merkle tree constructions (which exist but are less standard).

Algorithm Agility

The scheme is described with ML-DSA-65 and SHA-256. If either algorithm is weakened by future cryptanalysis, the historical records signed with the weakened algorithm lose their security guarantees. For long-lived records (10+ years), algorithm agility planning is important: the root signature algorithm should be negotiated and recorded in batch_metadata, and organisations should plan migration paths to newer algorithms (such as ML-DSA-87 for higher security, or SHA3-256 in place of SHA-256) as the threat landscape evolves. Records signed under a now-weakened algorithm should be re-anchored under a new algorithm before the break becomes practically exploitable.

Open Problem: Succinct Batch Proofs

The inclusion proof for a single record in a Merkle tree of n records requires O(log n) hash values, each of 32 bytes. For very large trees (n = 2^40, approximately a trillion records), the proof size is 40 × 32 = 1,280 bytes: still modest in absolute terms but larger than a Groth16 SNARK proof (~192 bytes). An open research question is whether a SNARK proof of Merkle tree inclusion can be combined with ML-DSA-65 post-quantum anchoring to produce proofs of constant size (the SNARK) with post-quantum binding (the ML-DSA signature over the circuit's verification key). This would be the fully succinct variant of the sublinear post-quantum attestation scheme.

SECTION 12

Conclusion

The adoption of ML-DSA-65 for post-quantum audit attestation is a necessary step for organisations with long-lived record requirements. The mathematics of the problem, however, are clear: per-record ML-DSA-65 signing is not scalable at the throughput levels required by AI governance, financial services, or healthcare audit systems. Linear growth in signature storage and verification time are fundamental properties of the per-record approach, not implementation details that can be engineered away.

The Merkle-anchored approach presented in this paper achieves sublinear attestation overhead: one ML-DSA-65 root signature per batch of arbitrary size, with O(log n) inclusion proofs for individual record verification. The security reduction is clean and standard, reducing to SHA-256 collision resistance and ML-DSA-65 unforgeability under MLWE. Both assumptions are expected to remain hard under quantum attack at the 128-bit post-quantum security level for at least the duration of typical regulatory retention periods.

The practical gains are substantial. At one million records per batch:

- Signature storage: 3.14 GB (per-record) reduced to 3,293 bytes (one root), a 99.9% reduction.

- Signing time: approximately 10 hours of CPU time for per-record signing, reduced to 37 µs for the root signature plus approximately 0.3 seconds for tree construction.
- Verification of one record: one ML-DSA-65 verify in both schemes, with 20 additional SHA-256 hashes in the Merkle scheme (negligible overhead).
- Verification of all n records: 1 million ML-DSA verifies (hours) reduced to n × 20 SHA-256 hashes plus one ML-DSA verify (seconds).

AffixIO's audit infrastructure uses Merkle-anchored ML-DSA-65 attestation for this reason: the mathematical properties make it the only viable design for compliance at scale. The open problem of fully succinct post-quantum batch proofs (SNARK-compressed Merkle inclusion with ML-DSA binding) is a natural extension that combines the work in WP-031 (ZK PQC circuits) with the Merkle architecture described here. Both the problem and the direction of the solution are sufficiently clear that practical implementations should be expected within the research community in the next two to three years.

> Related AffixIO whitepapers: WP-002: Post-Quantum Attestation covers ML-DSA-65 production deployment. WP-011: Merkle Tree Audit Architecture covers the Merkle structure without PQC-specific analysis. WP-030: Stateless Post-Quantum Verification covers the ZK layer above the attestation infrastructure. WP-031: ZK PQC Hybrid Key Exchange covers ZK proofs of ML-DSA and ML-KEM operations.

FREQUENTLY ASKED

Common Questions

Why is per-record ML-DSA-65 signing not scalable?

ML-DSA-65 signatures are 3,293 bytes each. At one million records per day, per-record signing produces 3.14 GB of signatures daily, growing to over 11 TB after 10 years of EU AI Act mandated retention. Verification scales linearly: checking n records requires n ML-DSA verifies. Merkle-anchored attestation

replaces n signatures with one root signature (3,293 bytes) plus O(log n) inclusion proofs (~640 bytes at 1 million records), reducing storage by over 99% while preserving individual record verifiability.

What is the security reduction for Merkle-anchored ML-DSA-65 attestation?

The combined scheme's security reduces to two independent quantum-hard assumptions: SHA-256 collision resistance and ML-DSA-65 unforgeability under MLWE. An adversary forging an attestation for a record not in the original batch must either find a SHA-256 collision (to produce a fraudulent inclusion proof) or forge an ML-DSA-65 signature (to change the attested root). Both are expected to remain computationally infeasible against quantum adversaries under current cryptographic consensus.

How does Merkle anchoring interact with regulatory retention requirements?

The scheme is compatible with EU AI Act (10 year), DORA (5 year), HIPAA (6 year), MiFID II (7 year), and SEC 17a-4 (6 year) retention requirements. The signed root and the records are retained for the full period; inclusion proofs are either stored or recomputable. Root publication to an immutable external registry at signing time ensures historical roots cannot be retroactively substituted, satisfying the tamper-evidence requirements of all the above frameworks.

What happens if the ML-DSA signing key is compromised after records are attested?

Records attested before the key compromise retain their authenticity, because the ML-DSA-65 signature was valid at the time of signing and the signed root is bound to its batch by the hash tree. An adversary with the compromised key can sign new fraudulent roots but cannot alter the hash values of previously published roots. Forward security depends on roots being published to an independent registry before the compromise occurs; roots that exist only in the signer's infrastructure at the time of compromise are at risk of substitution.

WP-002: PQ Attestation | WP-011: Merkle Architecture | WP-030: Stateless PQ Verification

▶ About

▶ Solutions

▶ Legal

▶ Trust & Security

Contact

truth layer | yes | no | proof