AffixIO Technical Paper · WP-011
June 2026
affix-io.com

AFFIXIO WHITE PAPER · WP-011

Merkle Tree Audit Architecture for AI Decision Systems

An append-only tree beats a database table nobody trusts.

AffixIO | United Kingdom | affix-io.com | June 2026

ABSTRACT

If your audit root can be edited, your audit story is optional. We document AffixIO's sorted-pair SHA-256 Merkle tree: append-only roots, inclusion proofs, and ML-DSA-signed anchors that third parties verify offline.

CONTENTS

SECTION 1

Introduction

The Merkle tree is a data structure invented by Ralph Merkle in 1979. Its most familiar applications are Bitcoin's transaction tree (where the root commits to all transactions in a block) and Certificate Transparency (RFC 9162, where the root commits to all certificates issued by a certificate authority). In both applications, the same property is exploited: a single 32-byte hash value (the root) commits to the contents of an arbitrarily large set of data, such that any alteration to any item in the set changes the root and is therefore detectable.

AffixIO applies this property to AI governance audit trails. Every AI governance event (a ZK proof generated for an AI response) is represented as a leaf in a Merkle tree. The root is computed after each insertion and signed with an ML-DSA-65 post-quantum key. The signed root is published to an append-only roots log. Any third party who holds a signed root and an inclusion proof for a specific leaf can verify that the leaf was in the tree at the time the root was signed, without querying AffixIO's infrastructure.

The result is an AI audit trail with properties that no log database, however well-secured, can provide: independently verifiable completeness and tamper-evidence, without trust in the infrastructure operator.

SECTION 2

Merkle Tree Fundamentals

A Merkle tree is a complete binary tree where each leaf node contains the hash of a data item, and each internal node contains the hash of the concatenation of its two children. The root node (or Merkle root) is the hash of the two top-level children. The root is a cryptographic commitment to the

entire set of data items: the probability of two different data sets producing the same root is bounded by the collision resistance of the underlying hash function.

An inclusion proof (also called a Merkle proof or Merkle path) for a specific leaf consists of the sibling hashes along the path from the leaf to the root. Given the leaf hash, the sibling hashes, and the root, any party can verify inclusion by recomputing the root from the leaf upward and checking that it matches the signed root. The verification requires only the leaf hash and the path; it does not require access to the complete tree.

The logarithmic size of inclusion proofs is a key operational advantage. For a tree with 10 million leaves, an inclusion proof requires approximately 23 sibling hashes (log2(10,000,000) ~ 23), totalling 736 bytes. This makes inclusion proofs practical to store and transmit as self-contained verification artefacts alongside governance records.

SECTION 3

The SHA-256 Sorted-Pair Scheme

AffixIO uses a sorted-pair Merkle hashing scheme: for any two sibling nodes A and B, the parent hash is computed as `SHA-256(min(A, B) || max(A, B))`, where ordering is lexicographic on the lowercase hex representation of the hashes.

The sorted-pair scheme has two advantages over the unsorted scheme used in simple binary trees. First, the same set of leaves always produces the same root regardless of the order in which leaves were inserted. This property (called order-independence) means that the root commits to the membership of a set, not to the sequence of insertions. An auditor can reconstruct the root from any ordering of the leaves without knowing the insertion sequence. Second, the sorted-pair scheme prevents certain second-preimage attacks that are possible in unsorted schemes when the attacker can choose the positions of leaf nodes. The sorted-pair scheme is consistent with Bitcoin's transaction tree hashing and RFC 9162.

```
def merkle_hash(a: str, b: str) -> str:
    left, right = (a, b) if a <= b else (b, a)
    return hashlib.sha256(
        bytes.fromhex(left) + bytes.fromhex(right)
    ).hexdigest()
```

SECTION 4

Leaf Construction

Each leaf in AffixIO's Merkle tree represents a single AI governance event. The leaf hash is computed as `a composite hash over proof metadata and event context`. The proof digest is itself a SHA-256 hash of the ZK proof bytes. The circuit identifier encodes the policy version. The outcome is "YES" or "NO". The timestamp is the UTC time of proof generation in ISO 8601 format with millisecond precision.

The leaf construction ensures that any two governance events with the same proof digest but different metadata (circuit version, outcome, or timestamp) produce different leaf hashes. A retroactive claim that a governance event occurred under a different circuit version would produce a different leaf hash, conflicting with the leaf hash in the signed root for that event.

> **Leaf formula:** `leaf_hash = H(proof_metadata || event_context)`. The prefix strings ("proof:", "circuit:", etc.) prevent length-extension attacks and make the leaf format self-describing.

SECTION 5

The Append-Only Roots Log

After each leaf insertion, the new Merkle root is published to an append-only roots log. Each entry in the roots log contains the root hash, the index of the most recently inserted leaf (the tree size), and the ML-DSA-65 signature over

(root_hash + leaf_index). The roots log is write-once: entries are never deleted or modified. The log grows by one entry per governance event.

The append-only property of the roots log is enforced at the storage layer using an object storage backend with object versioning disabled and a bucket policy that prevents deletion. The log is also periodically checkpointed to a separate cold storage location. The combination of write-once object storage and periodic checkpointing provides defence in depth against log manipulation.

The signed attestation record for each root (containing the root hash, tree size, timestamp, and ML-DSA-65 signature) can be stored with any governance record that needs to cite the state of the tree at a specific moment. An auditor examining a historical governance record can verify that the cited root was signed at the stated time by verifying the signature against the published ML-DSA-65 public key.

SECTION 6

Inclusion Proofs

An inclusion proof for a specific leaf consists of the leaf hash, the ordered list of sibling hashes on the path from the leaf to the root, and the signed root attestation for the root that includes this leaf. Given these three components, a verifier can confirm inclusion by the following procedure:

1. Start with the leaf hash as the current value.
2. For each sibling hash in the ordered list, compute `merkle_hash(current, sibling)` using the sorted-pair scheme. The result becomes the new current value.
3. After processing all siblings, the current value is the recomputed root.
4. Verify that the recomputed root matches the root in the signed attestation.
5. Verify the ML-DSA-65 signature on the attestation using the published public key.

If all five steps succeed, the inclusion proof is valid: the leaf was in the Merkle tree at the time the attested root was signed. The proof fails if any step fails, indicating either a forged leaf, a tampered path, or a forged attestation.

The verification procedure requires no network requests to AffixIO's infrastructure. It requires only the leaf hash, the sibling path, the attested root, and the published public key. All four components can be stored in a self-contained verification bundle alongside the governance record.

SECTION 7

Third-Party Verification

The independent verifiability of Merkle-anchored governance records is their most important property for regulatory and legal purposes. An auditor, regulator, or counterparty can verify any governance record in AffixIO's system without querying AffixIO's infrastructure, without access to AffixIO's databases, and without AffixIO's cooperation.

AffixIO publishes the ML-DSA-65 verification key as a static file at a well-known URL. The key is also embedded in the governance documentation and can be distributed through any channel. The verification procedure (Section 6) uses only this public key and the components of the verification bundle. There is no query to AffixIO's APIs in the verification procedure.

This property is crucial in adversarial contexts. If AffixIO were subject to a regulatory sanction, a corporate insolvency, or a dispute with a customer, and were uncooperative with verification requests, existing governance records would remain verifiable by any party holding the verification bundle. The governance records are not hostage to AffixIO's continued operation or cooperation.

SECTION 8

Post-Quantum Root Signing

Merkle roots are signed with ML-DSA-65 (NIST FIPS 204) using a key stored in an a certified HSM operating at FIPS 140-2 Level 3. The post-quantum signing protects the long-lived integrity of signed roots against future quantum attacks. A quantum computer capable of breaking classical elliptic curve signatures could forge signed roots for historical governance events if the roots were signed with ECDSA. ML-DSA-65 roots cannot be forged by any known quantum algorithm.

The ML-DSA-65 public key is relatively large (1,952 bytes) compared to a 32-byte ECDSA public key, but this is a manageable size for a key that is published statically and referenced in verification bundles. The signature itself is 3,309 bytes, also larger than classical signatures but acceptable for the attestation use case.

Key rotation policy: the ML-DSA-65 signing key is rotated annually. When a key is rotated, the new public key is published, and all subsequent attestations are signed with the new key. Historical attestations remain valid under the key that was active when they were signed. Verification bundles include the key version used for their attestation, so verifiers can use the correct public key for historical records.

SECTION 9

Multi-Tenant Architecture

AffixIO's Merkle tree service supports multi-tenant deployment: multiple organisations can use the same Merkle tree infrastructure, with their governance events interleaved in the same tree. This is operationally efficient (the tree can be maintained as a single service) and privacy-preserving (the leaf hashes of different tenants' events are indistinguishable in the tree; the tree structure reveals nothing about which tenant owns which leaf).

Each tenant's governance records include their tenant identifier as part of the leaf construction formula, ensuring that a leaf generated by one tenant cannot be presented as a leaf from another tenant. The tenant identifier is included in the hashed leaf data, not in the tree structure itself, preserving the privacy-by-structure property.

The multi-tenant tree also provides a completeness assurance benefit for each tenant: because the tree contains leaves from multiple tenants, its size and growth are independently observable from the perspective of any single tenant. A tenant who notices unexpected gaps in the roots log (roots published less frequently than expected) has evidence of a potential anomaly in the governance infrastructure, even if that anomaly only affects other tenants' leaves.

SECTION 10

Regulatory Alignment

The Merkle tree audit architecture addresses the audit trail requirements of multiple regulatory frameworks more robustly than log-based alternatives. The EU AI Act Article 12 requires that high-risk AI systems have logging capabilities to enable automatic recording of events. The Merkle-anchored audit trail satisfies this requirement with tamper-evident records. Article 15 requires cybersecurity measures; the ML-DSA-65 post-quantum signing satisfies this for long-lived records. ISO 42001 Clause 9 requires performance measurement and record maintenance; the Merkle tree provides continuous, append-only record maintenance with independent verifiability.

For financial services AI applications subject to FCA record-keeping requirements, the Merkle-anchored audit trail provides a stronger evidential basis than conventional database records: the tamper-evidence is cryptographic, not procedural. For regulated AI systems that may be subject to regulatory investigation or litigation, the independent verifiability of Merkle-anchored records means that the records can be submitted as evidence without requiring expert testimony about the trustworthiness of the record-keeping infrastructure.

SECTION 11

Operational Considerations

Running a production Merkle tree audit system requires attention to three operational concerns that do not arise with log databases.

Tree state durability. The Merkle tree state (all leaf hashes and internal node hashes) must be durable. If the tree state is lost, it cannot be reconstructed from the signed roots alone. AffixIO maintains a daily backup of the full tree state to cold storage, in addition to the always-available primary tree state.

Verification bundle retention. For each governance event, the verification bundle (leaf hash, sibling path, signed root) must be retained alongside the governance record. If only the governance record is retained without the bundle, later verification requires reconstructing the bundle from the tree state, which depends on the tree state being available. Retaining the bundle with the record makes verification independent of tree availability.

Root log consistency. The sequence of roots in the append-only log must be consistent: each root must be reachable from the previous root by the addition of one or more leaves. AffixIO performs a consistency check after each root publication: the new root must produce a valid consistency proof against the previous root (a standard Certificate Transparency consistency proof). Consistency proof failures halt root publication and trigger an operational alert.

SECTION 12

Conclusion

Merkle tree audit architecture provides AI governance audit trails with tamper-evidence and independent verifiability that log-based approaches cannot match. The SHA-256 sorted-pair scheme ensures that the same set of governance events always produces the same root. The append-only roots log ensures that no root is deleted or overwritten. The ML-DSA-65 signing

ensures that roots remain unforgeable against future quantum attacks. The self-contained verification bundle enables any third party to verify any governance record without querying AffixIO's infrastructure.

These are not incremental improvements on log-based audit trails. They represent a qualitative change in what kind of evidence AI governance records constitute. A Merkle-anchored, post-quantum-signed AI governance record is a cryptographic artefact whose integrity is guaranteed by mathematics and verifiable by anyone. This is the appropriate foundation for AI audit trails in regulated contexts where decisions may be challenged years after they were made.

Related reading

- WP-003: The Proof-Not-Log Paradigm for AI Audit Trails
- WP-002: Post-Quantum Attestation in Production with ML-DSA-65
- WP-001: Cryptographic AI Governance: A Technical Framework

Frequently asked questions

Why sorted-pair Merkle hashing?

Lexicographic ordering removes ambiguity in parent hash computation and matches Certificate Transparency practice.

What does an inclusion proof contain?

The leaf hash, sibling path, and signed root let verifiers recompute the root without AffixIO infrastructure access.

How often are roots signed?

After each batch of proof insertions, with roots published to an append-only log.

AffixIO | affix-io.com | hello@affix-io.com

All whitepapers | Download PDF

- About
- Solutions
- Legal
- Trust & Security

Contact

truth layer | yes | no | proof