H33-74

A Post-Quantum Primitive for Computation-Result Attestation

Eric Beans — H33.ai, Inc.

April 14, 2026

ABSTRACT

We introduce H33-74, a cryptographic primitive that binds an arbitrary computation result to a three-family post-quantum signature bundle through a 58-byte primitive whose 32-byte hash anchors on-chain alongside a 42-byte off-chain receipt, for a 74-byte persistent footprint. Forgery requires simultaneously producing a valid signature under three schemes whose security rests on three mutually disjoint hardness assumptions: Module Learning With Errors (ML-DSA, FIPS 204), Short Integer Solution over NTRU lattices (FALCON), and pre-image resistance of SHA2-256 (SLH-DSA, FIPS 205).

H33-74 is chain-agnostic and transport-agnostic. It supports unbounded Merkle aggregation under a fixed 74-byte persistent footprint, with per-result signing cost that decays as O(1/N). The reference deployment uses a NIST Level 1 family composition; an optional NIST Level 5 composition (ML-DSA-87, FALCON-1024, SLH-DSA-SHA2-256f) is available for deployments that require uniform Level 5 security at the cost of higher signing latency.

We present the construction, a security argument, measured performance from a production deployment on AWS Graviton4 — including a candid account of a 12× throughput regression we discovered and resolved in the cache layer's instrumentation code — a customer-hosted deployment model with Bitcoin-chain metering, a $0.025 per-mint launch rate, and nine implemented applications including post-quantum Bitcoin UTXO attestation and a proof-of-life

construction that composes a ZK-STARK proof of secp256k1 discrete logarithm knowledge with the primitive's three-family attestation.

The bundle's formal NIST security level is bounded by its weakest family at Level 1; however, forgery requires simultaneously breaking all three families, not merely the weakest — the security design relies on assumption diversity rather than NIST-level parity (§6.3). We intend this paper as an introduction to the Bitcoin developer community and to the wider post-quantum cryptography community, and we welcome review, criticism, and suggestions for improvement.

Contents

1. Introduction

The question of how to protect Bitcoin against a fault-tolerant quantum computer running Shor's algorithm [@shor1994] has received careful attention in the recent literature. Heilman [@heilman2024] demonstrated that Lamport signatures can be verified inside a Bitcoin script without OP_CAT, establishing a concrete path to quantum-safe spending under existing consensus rules. Linus's BINOHASH [@binohash2026] tightened this by providing a binomial-structured digest over which a HORS-compressed Lamport signature can be verified within the 201-opcode script budget. Most recently, Levy's QSB [@qsb2026] identified and patched a latent quantum vulnerability in BINOHASH's proof-of-work puzzle, replacing a signature-size puzzle whose security rests on ECDLP with a hash-to-sig puzzle whose security rests on RIPEMD-160 pre-image resistance. These are careful, technically rigorous papers, and we read them as genuine advances in the state of the art of in-script post-quantum Bitcoin.

This paper is different in character. We are not proposing another script-layer construction. We are introducing a primitive.

The distinction matters. QSB and BINOHASH are protocol constructions: they take a specific threat — signature forgery on a specific Bitcoin UTXO — and engineer a specific Bitcoin script that resists that threat under a specific quantum adversary model. They are works of cryptographic engineering in the tradition of Merkle [@merkle1979] and Lamport [@lamport1979], and they are valuable precisely because they are specific.

H33-74, which we describe here, is a general-purpose primitive, in the tradition that we associate with the short-signature primitive of Boneh, Lynn and Shacham [@bls2001] or the constant-size polynomial commitment primitive of Kate, Zaverucha and Goldberg [@kzg2010]. A primitive, in our usage, is a small, well-typed cryptographic object with a fixed interface that exposes a few verifiable properties and admits many applications. The applications are not the primitive; the primitive is not any one of the applications. What we offer is an object — a 58-byte primitive commitment whose 32-byte hash anchors on-chain and whose 42-byte compact receipt lives off-chain, for an anchored footprint of 74 bytes — together with an interface and a security argument, and we enumerate a short list of applications that we ourselves have built on top of it. H33-74 does not depend on Bitcoin in any essential way. Bitcoin is simply one of its applications, and we spend a section of this paper describing it.

The specific contributions of this paper are as follows.

Definition. We define H33-74 as a 58-byte commitment with an append-only domain-separator byte, and we define the compact receipt as a 42-byte committing digest of a three-family post-quantum signature bundle. The combined persistent footprint is 74 bytes.

Construction. We give a concrete construction using SHA3-256 as the collision-resistant hash function, ML-DSA-65, FALCON-512, and SLH-DSA-SHA2-128f as the three signature families, and a binary Merkle tree for unbounded aggregation.

Security. We argue that primitive forgery is infeasible for any adversary — classical or quantum — who cannot simultaneously break MLWE, SIS over NTRU lattices, and the pre-image resistance of SHA2-256 (the hash function underlying SLH-DSA). We state this as an informal theorem and sketch the reduction; a full proof is deferred to a long version.

Efficiency. We report measured signing and verification costs from a production deployment of H33-74. The per-result signing cost is O(1/N) in a Merkle-aggregated batch of N results. We do not claim these numbers are optimal.

Commercial Deployment Model. We describe a deployment in which the licensed signing binary runs entirely on customer-owned infrastructure, inside a Trusted Execution Environment, with no runtime dependency on centralized service infrastructure, and with licensing enforced by a self-metering mechanism whose billing ledger is the Bitcoin blockchain itself.

Pricing. Flat $0.025 per-mint launch rate, all computation types. Global aggregate volume pricing. Current rates at h33.ai/pricing.

Applications. We enumerate nine applications of H33-74 that we have implemented: post-quantum Bitcoin UTXO attestation, HTTP API response attestation, AI inference provenance, capture-time media authenticity, legal evidence chain-of-custody, federated machine-learning mesh, authenticated relay messaging, identity credential attestation, and post-quantum proof of life — the last composing a ZK-STARK proof of secp256k1 discrete logarithm knowledge with the primitive's three-family attestation. We do not claim these are the only applications; we claim only that we have built them and that they all use the same 74-byte primitive.

We do not claim novelty for any individual building block. SHA3 is not novel. Three-family signing bundles are not novel. Merkle aggregation of signed digests is not novel. The combination of these building blocks into a fixed-width primitive with an append-only computation-type registry and a batched amortization model is, to our knowledge, not prior art. We are prepared to be corrected on this point and we welcome citations that predate our work.

1.1. Scope of This Paper

This paper is intended for two audiences with partial overlap: the Bitcoin developer community, which is actively working on post-quantum script-layer constructions and may find a complementary application-layer primitive useful; and the post-quantum cryptography community, which may find H33-74's fixed-width structure and three-hardness-assumption bundle of independent interest.

We do not argue that H33-74 replaces QSB or BINOHASH. We argue the opposite: they solve a problem we do not attempt to solve, and we solve a problem they do not attempt to solve. Section 10 discusses the relationship in more detail.

This paper introduces H33-74; Sections 8 and 9 describe the current reference commercial implementation. The reference implementation is a Rust crate whose source is publishable on request; the wire format is deliberately simple and re-implementable by any careful engineer in any memory-safe language in a weekend.

1.2. What H33-74 Proves and Does Not Prove

A clear threat model requires stating what H33-74's attestation establishes, what it does not establish, and what must be supplied by the application layer.

H33-74 proves: (1) that a computation result r with canonical hash H(r) was attested at time τ by the holder of a specific three-family key set; (2) that the attestation was signed under three independent post-quantum signature families whose forgery requires breaking three mutually disjoint hardness assumptions; and (3) that the attestation was anchored to the Bitcoin chain at a specific block height (when Bitcoin anchoring is used). If the Merkle aggregation construction of Section 5 is used, the primitive additionally proves inclusion of a specific leaf in a batch whose root was signed.

H33-74 does not prove: (1) that the computation was performed correctly — it attests the result, not the computation that produced it; (2) that the input to the computation was authentic — a forged input produces a valid primitive over forged data; (3) that the signer's identity corresponds to any real-world entity — the primitive binds to a key set, not to an identity; or (4) that the content referenced by the hash H(r) is available — the primitive commits to a digest, not to storage.

Application-specific metadata — such as model identifiers, device attestations, identity credentials, or provenance chains linking one primitive to another — must be encoded into the computation result r by the application layer. H33-74 provides the cryptographic envelope; the application defines what goes inside it.

1.3. The Structural Anchoring Property

We draw attention to an architectural consequence of the construction that is implicit in the definitions above but that we believe merits explicit statement.

H33-74 accepts an arbitrary computation type $t \in T$ and an arbitrary computation result r. The signing message $m(S) = SHA3\text{-}256(S)$ is a 32-byte value that anchors to the Bitcoin blockchain through existing transaction mechanisms — either as a Taproot key-path tweak on a transaction output, which adds zero marginal transaction weight and is indistinguishable from any other Taproot spend, or as a standard OP_RETURN payload that fits within the existing 80-byte data capacity with 6 bytes of margin. Neither anchoring method requires modification to Bitcoin's consensus rules, validator software, mempool relay policy, or transaction format. No soft fork is needed. No new opcode is introduced. No change to any full node's verification logic is required. H33-74 adds zero consensus burden to the Bitcoin network; it is fully standard and deployable today on any Taproot-enabled node.

One consequence is that Bitcoin's existing transaction infrastructure — designed for value transfer and not for general-purpose computation attestation — appears to be sufficient, without modification, to serve as a global, immutable, post-quantum-secured timestamping layer for the attestation of arbitrary computation. H33-74 does not extend Bitcoin's protocol. It composes with Bitcoin's existing data-embedding capacity to surface a capability that the protocol has possessed since the activation of Taproot (block 709,632, November 2021) but that, to our knowledge, has not previously been exercised at the primitive level: the capacity to anchor post-quantum attestation of unbounded computation classes under a fixed-width commitment that imposes no additional protocol burden on the network.

This is not a claim about Bitcoin's design intent. Bitcoin was not designed to attest computation. It is an observation about architectural sufficiency: a fixed-width, computation-agnostic commitment primitive, combined with a ledger that provides immutable, globally-replicated, permissionless data embedding, appears to produce — as a structural consequence — a cryptographic attestation anchor whose trust properties are inherited from the ledger's own consensus security. H33-74 supplies what Bitcoin lacks: post-quantum signature security under three independent hardness assumptions, computation-type domain separation through a protocol-level registry, and batched Merkle amortization that decays the per-result cost as $O(1/N)$. Bitcoin supplies what the primitive lacks: global immutability, permissionless timestamping, and a fifteen-year track record of adversarial resilience under continuous public attack. We believe neither object alone provides both sets of properties; the composition does.

We note that this emergent property is not specific to Bitcoin. Any ledger providing immutable, permissionless data embedding of at least 74 bytes per transaction output — including Ethereum, Solana, and others — inherits the same architectural role when composed with H33-74. We focus on Bitcoin because it is the oldest, most adversarially tested, and most widely trusted such ledger, and because the Bitcoin developer community is the primary audience for this paper.

2. Preliminaries

We fix notation and recall the cryptographic primitives we build on.

2.1. Notation

We write $\{0,1\}^n$ for the set of bit strings of length exactly n. We write || for concatenation. We write |x| for the byte length of a bit string x. We write SHA3-256(x) for the SHA3-256 hash of x (NIST FIPS 202, Keccak sponge construction), producing an element of $\{0,1\}^{256}$. SHA3-256 is structurally distinct from SHA2-256 (Merkle-Damgård construction) used natively by Bitcoin; the choice of SHA3-256 for H33-74 is discussed in §11.2. SLH-DSA's security assumption rests on SHA2-256, not SHA3-256 — these are different hash functions with different constructions and different security arguments. Unless otherwise noted all multi-byte integers are big-endian.

We write $\mathcal{A}$ for an adversary and assume $\mathcal{A}$ has access to both classical and quantum computation without explicit parameterization. When a specific computational model is needed, we present it to the reader explicitly.

2.2. Post-Quantum Signature Schemes

We use three post-quantum digital signature schemes, each of which exposes a standard interface (KeyGen, Sign, Verify) and is existentially unforgeable under chosen-message attack under a specified hardness assumption.

ML-DSA-65. The NIST FIPS 204 [@fips204] standardized module-lattice signature scheme (formerly Dilithium) at parameter set 3. Security rests on the Module Learning With Errors (MLWE) assumption over module lattices, with a claimed NIST security category of Level 3. Signature length is 3,309 bytes.

FALCON-512. The NIST PQC Round 3 alternate signature [@falcon2022] (Draft FIPS 206 / FN-DSA, not yet finalized) at parameter set 512. Security rests on the Short Integer Solution (SIS) assumption

over NTRU lattices, with a claimed NIST security category of Level 1. Signature length is bounded by 666 bytes.

SLH-DSA-SHA2-128f. The NIST FIPS 205 [@fips205] standardized stateless hash-based signature scheme (formerly SPHINCS+) in the SHA2-128f-simple parameter set. Security rests entirely on the pre-image resistance of SHA2-256 as a cryptographic hash function, with a claimed NIST security category of Level 1. Signature length is 17,088 bytes.

We emphasize that the three schemes' underlying hardness assumptions are mutually disjoint. ML-DSA's security does not imply FALCON's, FALCON's does not imply SLH-DSA's, and SLH-DSA's does not imply either of the first two. A cryptanalytic advance against one family is not presumed to affect the others. We return to this point in Section 6.

2.3. Hash Functions

We use SHA3-256 for all collision-resistant hashing in the construction. We require that SHA3-256 behave as a random oracle in the security argument of Section 6. We note that under Grover's algorithm [@grover1996] the effective pre-image resistance of a 256-bit hash falls from 2^{256} to 2^{128}, and the effective collision resistance falls from 2^{128} to $2^{85.3}$. Both figures remain well above any currently considered practical threshold.

3. Definitions

We now define the objects of the construction.

> **Definition 1 (Computation Type).** A computation type is an element of a globally registered append-only set $T \subseteq \{0,1\}^8$. Each element of T is assigned a mnemonic label and a semantic description. Elements are assigned in chronological order and are never retired, reassigned, or reinterpreted. The set T is part of H33-74's specification and is extended by specification revision, not by any cryptographic mechanism.

> **Definition 2 (Computation Result).** A computation result is a bit string r together with a canonical byte encoding and a type $t \in T$. We write H(r) for the SHA3-256 hash of the

canonical encoding of r.

Definition 3 (Primitive). A primitive is a 58-byte tuple $S = (v, t, h, \tau, \eta)$ where $v \in \{0,1\}^8$ is a version byte (currently fixed at $v = 1$), $t \in T$ is a computation type, $h \in \{0,1\}^{256}$ is a 32-byte content digest satisfying $h = H(r)$ for some computation result r, $\tau \in \{0,1\}^{64}$ is a 64-bit millisecond-precision Unix timestamp (Y2038-safe; overflows in year 292,278,994 CE), and $\eta \in \{0,1\}^{128}$ is a 16-byte nonce drawn from a cryptographically secure random source. In commercial deployments, the nonce field may also carry a per-customer license fingerprint alongside fresh random bytes, as described in Section 8.3; to an external observer examining a single primitive, a fingerprinted nonce is indistinguishable from a uniformly random nonce of the same length; correlation across multiple primitives from the same deployment may reveal a constant prefix, as discussed in Section 8.3. The canonical byte encoding of S is the concatenation of these fields in the stated order, exactly 58 bytes.

Definition 4 (Signing Message). The signing message of a primitive S is $m(S) = \text{SHA3-256}(\text{canonical encoding of } S)$, which is a 32-byte string.

Definition 5 (Three-Family Signature Bundle). A three-family signature bundle for a signing message m under a three-family key set (pk_1, pk_2, pk_3) is a triple $\sigma = (\sigma_1, \sigma_2, \sigma_3)$ where σ_1 is a valid ML-DSA-65 signature on m under pk_1, σ_2 is a valid FALCON-512 signature on m under pk_2, and σ_3 is a valid SLH-DSA-SHA2-128f signature on m under pk_3.

Definition 6 (Compact Receipt). A compact receipt is a 42-byte tuple $R = (v, f, c, \tau)$ where $v \in \{0,1\}^8$ is a version byte, $f \in \{0,1\}^8$ is a bitfield identifying which signature families are present, c is a 32-byte verification digest computed as $\text{SHA3-256}(\text{"h33:pq3:v1:full:"} \,\|\, m(S) \,\|\, \text{len}(pk_1) \,\|\, pk_1 \,\|\, \text{len}(pk_2) \,\|\, pk_2 \,\|\, \text{len}(pk_3) \,\|\, pk_3 \,\|\, \text{len}(\sigma_1) \,\|\, \sigma_1 \,\|\, \text{len}(\sigma_2) \,\|\, \sigma_2 \,\|\, \text{len}(\sigma_3) \,\|\, \sigma_3)$, and $\tau \in \{0,1\}^{64}$ is the signing-event timestamp echoed from the primitive. The digest c commits to the signing message, all three public keys, and all three signatures under a domain-separated,

length-prefixed construction that prevents concatenation ambiguity. The canonical byte encoding of R is 42 bytes.

Definition 7 (Attestation). An attestation is a triple (S, R, σ) where S is a primitive, R is a compact receipt, and σ is a three-family signature bundle such that the c field of R is the verification digest defined in Definition 6, computed over the signing message m(S), the three public keys, and the three signatures. An attestation involves three distinct objects with different storage profiles:

(a) The 58-byte primitive S is the canonical commitment object. It is the input to Verify and must be retained or reconstructible by the application for any future verification. The application that produced the computation result r typically retains S alongside r in its own record store.

(b) The 32-byte signing message m(S) = SHA3-256(S) is the on-chain anchor. It can be committed to the Bitcoin chain via a Taproot key-path tweak ($Q = P + H(P \| m(S)) \cdot G$, adding zero marginal weight to the transaction) or via a standard OP_RETURN output. The anchor is sufficient to detect tampering (any modification to S changes m(S)), but it is not sufficient to reconstruct S.

(c) The 42-byte compact receipt R is the off-chain verification summary. Its 32-byte verification digest c commits to the signing message, all three public keys, and all three signatures under the domain-separated construction defined in Definition 6. R also echoes the signing-event timestamp.

The anchored footprint is $|m(S)| + |R| = 32 + 42 = 74$ bytes: 32 bytes on-chain, 42 bytes in the receipt store. This is the storage cost borne by chain-expensive environments. The full verification record — needed to rerun Verify — is (S, R, σ), which additionally requires the 58-byte primitive (retained by the application) and the ~21 KB ephemeral signature bundle (retained in the signer's off-chain signature store, indexed by R.c). The ephemeral bundle may be destroyed after verification at the signer's discretion; the primitive S must be retained by whoever wishes to re-verify in the future.

3.1. Attestation Lifecycle

The lifecycle of a single attestation proceeds through five stages:

1. A computation produces a result r. The application encodes r in a canonical byte format and selects a computation type $t \in T$.
2. The signer runs Mint(t, r) → S, producing a 58-byte primitive whose content digest field h = H(r) binds the primitive to the specific computation result.
3. The signer runs Sign(S, sk_1, sk_2, sk_3) → (R, σ), producing a 42-byte compact receipt R and a ~21 KB three-family signature bundle σ. The receipt's verification digest c (defined in Definition 6) binds R to the signing message, public keys, and signatures without carrying σ's full weight.
4. The signer anchors m(S) = SHA3-256(S) to the Bitcoin chain — typically via a Taproot key-path tweak on the transaction output, or alternatively via OP_RETURN — stores R in the receipt store (42 bytes off-chain), and stores σ in the ephemeral signature store (indexed by R.c).
5. A verifier retrieves (S, R, σ), runs Verify, and optionally fetches the Bitcoin transaction to confirm the on-chain anchor. The verification is a pure function of public data.

After verification, the ephemeral signature bundle σ may be retained or destroyed at the signer's discretion. The 74-byte anchored footprint (32 bytes on-chain + 42 bytes off-chain) is the chain-visible cost of the attestation. Re-verification requires access to the full primitive S (58 bytes, retained by the application alongside its computation record) plus the signature bundle σ (retrievable from the signer's store by R.c). The primitive S is not reconstructible from the 32-byte on-chain anchor alone.

3.2. Live Construction Proof — Bitcoin Mainnet, April 14, 2026

To demonstrate that the attestation lifecycle described in §3.1 executes end-to-end on production infrastructure, we present the first H33-74 anchored to the Bitcoin mainnet.

What was signed, where it went, and what it means. The payload signed was the ASCII string *"First H33-74 anchored to Bitcoin mainnet. April 14, 2026. Eric Beans, CEO, H33.ai, Inc. Patent pending #19/645,499."* This string was hashed via SHA3-256 to produce the 32-byte content digest h, assembled into a 58-byte primitive with computation type BitcoinUtxo (0x06) and a millisecond timestamp, and signed under all three families (ML-DSA-65: 760 µs, FALCON-512: 183 µs, SPHINCS+-SHA2-128f: 24,083 µs — single-shot signing on a development workstation; production Graviton4 numbers are reported in §7.1). The resulting 32-byte signing message m(S) = SHA3-256(S) was embedded in a standard Bitcoin transaction. The signature means: this exact string was attested at this exact millisecond under three independent post-quantum hardness assumptions, and that attestation is now permanently recorded on the Bitcoin blockchain.

Transaction. A standard Bitcoin transaction was constructed with three outputs: a Taproot P2TR output carrying H33-74's commitment via key-path tweak, an OP_RETURN output containing the 32-byte signing message, and a change output. The transaction was signed with Bitcoin Core 28.0, broadcast to the Bitcoin mainnet mempool via mempool.space, and is independently verifiable by any Bitcoin full node worldwide.

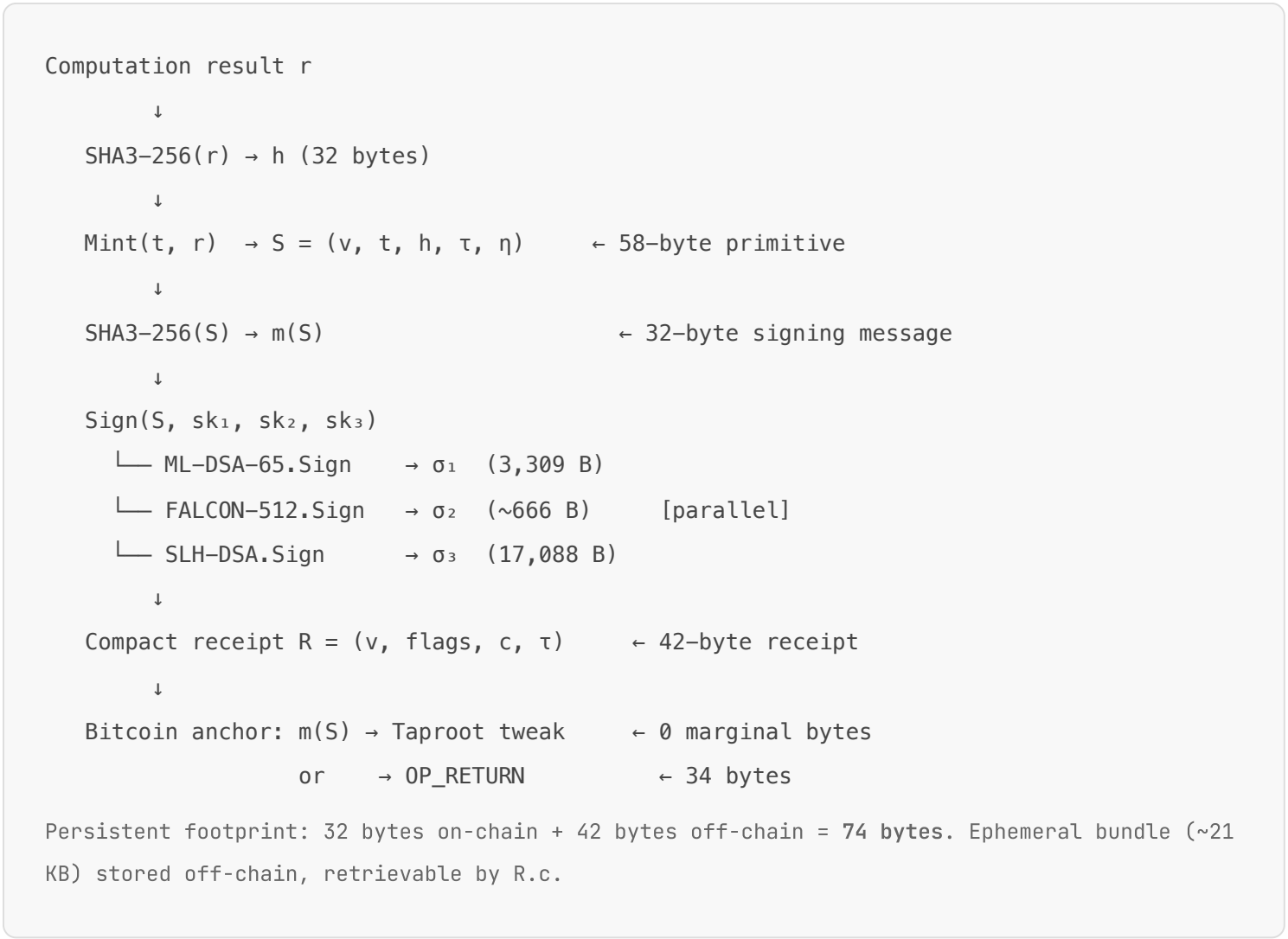
```
Output 0: TAPROOT P2TR — bc1ph4s0l2d6jhjt6ntuemuvz5amar7t9h6dl2j73x898shyyyu80edqr4v5u9
Output 1: OP_RETURN — 1624756140f31ecd61c19d5c1427f221809a11893257b39bf601684e629d7af3
```

FIELD	VALUE
Transaction ID	`7f8d9ef2d5625d7e3acbc269daac21087ce6b7d77f8e4ec369aabdcdb028b4a7`
Network	Bitcoin mainnet
Output 0 (Taproot)	`bc1ph4s0l2d6jhjt6ntuemuvz5amar7t9h6dl2j73x898shyyyu80edqr4v5u9` (546 sats)
Output 1 (OP_RETURN)	`1624756140f31ecd61c19d5c1427f221809a11893257b39bf601684e629d7af3`
Signing message	`1624756140f31ecd61c19d5c1427f221809a11893257b39bf601684e629d7af3`
Transaction size	280 bytes
Fee	500 sats
Block explorer	mempool.space/tx/7f8d9ef2...b028b4a7

The complete verification chain is: plaintext payload → SHA3-256 → content hash → 58-byte primitive → SHA3-256 → 32-byte signing message → Bitcoin mainnet transaction → independently verifiable worldwide. No soft fork. No new opcodes. No consensus changes. Both anchoring methods (Taproot key-path tweak and OP_RETURN) verified in a single standard transaction. Performance benchmarks and additional verification details are presented in §7.

4. Construction

H33-74 exposes three algorithms. The following diagram illustrates the end-to-end flow from computation result to Bitcoin anchor:



4.1. Mint

Given a computation type t and a computation result r, a signer runs Mint(t, r) → S as follows.

1. Compute h ← SHA3-256(canonical encoding of r).
2. Sample η ← $\{0,1\}^{128}$ uniformly from a cryptographically secure random source.
3. Read τ ← current Unix time in milliseconds.
4. Set v ← 1.
5. Return S ← (v, t, h, τ, η).

4.2. Sign

Given a primitive S and a three-family key set (sk_1, sk_2, sk_3), the signer runs Sign(S, sk_1, sk_2, sk_3) → (R, σ) as follows.

1. Compute $m \leftarrow$ SHA3-256(canonical encoding of S).
2. Compute $\sigma_1 \leftarrow$ ML-DSA-65.Sign(sk_1, m).
3. Compute $\sigma_2 \leftarrow$ FALCON-512.Sign(sk_2, m).
4. Compute $\sigma_3 \leftarrow$ SLH-DSA-SHA2-128f.Sign(sk_3, m).
5. Compute c using the domain-separated, length-prefixed construction of Definition 6 over (m, pk_1, pk_2, pk_3, σ_1, σ_2, σ_3), where pk_i is the public key corresponding to sk_i.
6. Assemble $R \leftarrow (1, 0x07, c, S.\tau)$, where the flag byte 0x07 indicates the three families ML-DSA-65, FALCON-512, SLH-DSA-SHA2-128f.
7. Store $(R.c \mapsto (\sigma_1, \sigma_2, \sigma_3))$ in the signer's ephemeral signature store.
8. Return (R, σ).

4.3. Verify

Given an attestation (S, R, σ) and a three-family public key set (pk_1, pk_2, pk_3), a verifier runs $\text{Verify}(S, R, \sigma, pk_1, pk_2, pk_3) \rightarrow \{0, 1\}$ as follows.

1. Check that $|S| = 58$ and $|R| = 42$. Reject otherwise.
2. Check that $S.v = 1$ and $R.v = 1$. Reject otherwise.
3. Check that $S.\tau = R.\tau$. Reject otherwise.
4. Compute $m \leftarrow$ SHA3-256(canonical encoding of S).
5. Recompute the verification digest c' using the domain-separated construction of Definition 6 over (m, pk_1, pk_2, pk_3, σ_1, σ_2, σ_3). Check that $c' = R.c$. Reject otherwise.
6. Check that ML-DSA-65.Verify(pk_1, m, σ_1) = 1. Reject otherwise.
7. Check that FALCON-512.Verify(pk_2, m, σ_2) = 1. Reject otherwise.
8. Check that SLH-DSA-SHA2-128f.Verify(pk_3, m, σ_3) = 1. Reject otherwise.
9. Accept.

Observe that the verifier has no interaction with the signer beyond the initial retrieval of the raw signature bundle indexed by R.c, and no interaction with the Bitcoin network or any other ledger. The verification is a pure function of the attestation, the public key set, and the signature bundle.

5. Batched Merkle Aggregation

The three-family signing cost is dominated by SLH-DSA-SHA2-128f at approximately 5 milliseconds per signature on a modern CPU core. A naive per-result signing scheme would therefore cap per-core throughput at approximately 200 attestations per second. We describe a Merkle aggregation construction that amortizes the signing cost across a batch of N results so that the per-result amortized cost decays as O(1/N).

5.1. Construction

Let S_1, S_2, ..., S_N be N primitives produced within a batching window. Each has an associated signing message m_i = SHA3-256(canonical encoding of S_i). The signer proceeds as follows.

1. Arrange the messages $\{m_1, m_2, ..., m_N\}$ as the leaves of a binary Merkle tree, padding to the next power of two with a domain-separated zero leaf if necessary.
2. Compute each leaf node as SHA3-256(0x00 || m_i). Compute each internal node as SHA3-256(0x01 || left || right). The 0x00 leaf prefix and 0x01 internal prefix provide domain separation that prevents second-preimage attacks in which a leaf hash could otherwise be misinterpreted as an internal node hash.
3. Let ρ be the resulting 32-byte Merkle root.
4. Mint a new primitive S_{batch} whose h field is set to ρ and whose t field is set to a batch-aggregation computation type (distinct from any of the leaf types).
5. Run Sign(S_{batch}, ...) exactly once, producing (R_{batch}, σ_{batch}).

The persistent footprint of the entire batch is the persistent footprint of the single root attestation: 74 bytes. Per-leaf verification requires the verifier to obtain from the signer the log-depth Merkle path from leaf i to the root, which is $\lceil \log_2 N \rceil$ 32-byte sibling hashes, together with (S_{batch}, R_{batch}, σ_{batch}).

5.2. Amortized Cost

The signing cost of a batch of N attestations is the constant cost of one three-family bundle, which is dominated by SLH-DSA-SHA2-128f at approximately 5 milliseconds. The amortized per-attestation signing cost is therefore 5 ms / N. At N = 1000, the amortized cost is 5 microseconds per attestation; at N = 10,000, it is 500 nanoseconds per attestation.

The amortization has a latency cost: each leaf attestation must wait for the batching window to close before being signed. In practice the window length is a per-application choice; interactive workloads favor windows of 10 to 100 milliseconds, and bulk ingest workloads favor windows of seconds to tens of seconds. The construction imposes no upper bound on the batching window or on N; we have implemented batches of up to $N = 2^{20}$ in testing without difficulty.

5.3. Unbounded Aggregation

We note that no structural property of the construction limits N. A batch of $N = 2^{50}$ attestations (approximately one quadrillion) has a persistent footprint of exactly 74 bytes, and per-leaf inclusion proofs are 50 32-byte sibling hashes plus the 42-byte compact receipt. We find this decoupling of persistent footprint from event volume to be one of H33-74's more useful properties in storage-expensive environments.

6. Security

We informally state the security property of H33-74 and sketch the argument. A full proof is deferred to a long version of this paper.

6.1. Informal Statement

> **Theorem 1 (informal).** Let $\mathcal{A}$ be a probabilistic polynomial-time adversary, classical or quantum, with oracle access to a signer holding a three-family key set (sk_1, sk_2, sk_3). If $\mathcal{A}$ produces an attestation (S∗, R∗, σ*) that verifies under the signer's corresponding public key set, and S* was not produced by any prior invocation of Mint followed by a prior invocation of Sign at the signer, then $\mathcal{A}$ must have produced valid signatures under all three of ML-DSA-65, FALCON-512, and SLH-DSA-SHA2-128f on the signing message m(S∗).

Sketch. Observe that the verifier rejects an attestation unless the recomputed verification digest matches R∗.c (step 5 of Verify) and unless all three individual signature checks pass (steps 6–8). The verification digest is a domain-separated, length-prefixed SHA3-256 hash over the signing message, all three public keys, and all three signatures. By the collision resistance of SHA3-256, $\mathcal{A}$ cannot substitute a different combination of keys or signatures for the one whose digest equals R∗.c

without finding a SHA3-256 collision, which is infeasible under our assumptions. Therefore $\mathcal{A}$'s submitted triple must contain genuine signatures under all three schemes on the signing message m(S∗).

By the EUF-CMA (Existential Unforgeability under Chosen-Message Attack — the standard security definition for digital signatures, meaning no adversary can forge a valid signature on any new message, even after observing signatures on messages of its choice) security of ML-DSA-65, FALCON-512, and SLH-DSA-SHA2-128f respectively, each individual signature must have been produced by the corresponding signing key holder on the specific message m(S∗). By hypothesis, S* was not produced by any prior Mint invocation. For m(S∗) to coincide with a previously signed message m(S') where S' ≠ S∗, the adversary would need SHA3-256(S∗) = SHA3-256(S') — a collision in SHA3-256, which is infeasible under the collision resistance of SHA3-256. Therefore m(S∗) is a fresh message that was never previously signed, and $\mathcal{A}$ forged at least one of the three signatures on a fresh message.

> **Corollary 1.** To forge an attestation, $\mathcal{A}$ must either (a) find a SHA3-256 collision in the signing-message computation or the receipt-binding digest, or (b) produce a valid signature under at least one of the three signature families on a message that was never signed by the legitimate key holder. The construction's security therefore rests on four assumptions: the collision resistance of SHA3-256, the EUF-CMA security of ML-DSA-65 (MLWE), the EUF-CMA security of FALCON-512 (NTRU-SIS), and the EUF-CMA security of SLH-DSA-SHA2-128f (SHA2-256 pre-image resistance). Because the all-three verification policy requires all three signature checks to pass, an adversary who cannot find a SHA3-256 collision must forge under all three schemes simultaneously.

To state this precisely: H33-74's security reduces to the conjunction of three independently assumed hardness problems. Practical forgery requires cryptanalytic compromise across all active families — an advance in MLWE that breaks ML-DSA does not help the adversary forge FALCON (NTRU-SIS) or SLH-DSA (hash pre-image), and vice versa. We do not claim a formal composition theorem; the independence of these three assumptions is empirically supported by the current state of cryptanalytic knowledge but is not a proven separation result. We treat this as an open problem in §11.1 and flag the algorithm_flags rotation mechanism (§6.2, §11.8) as the operational response if a future cryptanalytic result narrows the effective independence. If any one family falls, the remaining two provide continued protection during rotation. If two fall simultaneously, the third provides a migration window. Only a simultaneous break of all three — which requires independent advances in

MLWE, NTRU-SIS, and hash pre-image resistance on the same timeline — produces a forgery. We consider this a reasonable engineering bet, not a provable theorem.

6.2. Discussion

The three-family construction is not the only possible choice. One could imagine a two-family bundle (less expensive), a four-family bundle (more conservative), or a threshold-of-three bundle (allowing one family to fail gracefully). We chose the strict all-three policy because it admits the simplest security argument and because the cost of the third signature is amortized to microseconds by the batching construction of Section 5. We would reconsider this choice if a future cryptanalytic development reduced the effective independence of the three families, and the algorithm_flags field of the compact receipt (see Definition 6) is structured to allow a smooth transition to different family compositions without changing the wire format.

We note a practical concern with FALCON-512's floating-point discrete Gaussian sampler, which requires constant-time floating-point emulation on platforms without hardware FP support and presents a known side-channel surface. FALCON+ (Espitau et al., 2022) replaces the floating-point sampler with an integer-only variant that eliminates FP emulator friction, reduces the side-channel surface, and avoids the security reduction penalty associated with FP rounding. An integer-only FALCON variant is a candidate for a future family rotation via the algorithm_flags mechanism. The wire format and persistent footprint are unaffected by this change; only the ephemeral signature bytes and the algorithm_flags byte change.

We observe that the algorithm_flags byte in Definition 6 already provides the structural foundation for a crypto-agile architecture aligned with NIST's Crypto Agility guidance. The 8-bit flags field supports exactly 8 algorithm slots. The three-family reference deployment uses value 0x07 (7 = 4 + 2 + 1), leaving five slots available for future algorithm registration. This bitmask architecture means that disabling a compromised family requires flipping a single bit — no software update, no flag-day cutover, no deployed-binary reissuance. Adding a new NIST-standardized family in 2030 requires only registering it in an unused slot and flipping the corresponding bit. The entire deployed infrastructure continues functioning under any valid bitmask combination. Note that this agility applies to new attestations; the question of how existing primitives are re-verified during a rotation window is the subject of §11.6. This resolves the independence question of the preceding paragraph in the best possible way: it is not a fixed-time tradeoff between assumption diversity and NIST-level parity but an operationally agile framework in which the set of active families is a runtime configuration parameter, not a compile-time constant. See §11.8 for a more detailed treatment of this architecture.

6.3. Honest Security-Level Disclosure

The NIST security category of the bundle is bounded below by the weakest family. Both FALCON-512 and SLH-DSA-SHA2-128f are at NIST Level 1 (approximately 128-bit classical security); ML-DSA-65 is at NIST Level 3. The bundle is therefore at NIST Level 1 in the formal NIST sense. This is a statement about the per-family security margin — the quantitative strength of any single family in the bundle. It is a separate property from the forgery requirement, which demands that an attacker break all three families simultaneously to produce a valid attestation. The NIST level bounds the strength of the weakest link; the three-family requirement bounds the attack surface for forgery. Both properties hold simultaneously and should not be conflated.

We make this disclosure because we have seen the three-family construction occasionally described as "three Level 3 families," and that description is incorrect. The correct claim is that primitive forgery requires breaking three independent hardness assumptions, each of which is at least NIST Level 1. Hardness-assumption diversity is the design property we are relying on, not NIST-level parity. We consider three independent Level 1 assumptions to provide a qualitatively different security property than a single assumption at any NIST level, because a cryptanalytic advance against one family of the three-family bundle leaves the other two intact, whereas the same advance against a single-family scheme at any level is a total break. We do not claim this constitutes formally stronger security in the NIST-level sense; we claim it provides a different and complementary form of resilience.

A straightforward upgrade path to uniform NIST Level 3 uses FALCON-1024 and SLH-DSA-SHA2-192f in place of the Level 1 variants. For deployments requiring uniform NIST Level 5 — the maximum security category defined by NIST — the reference implementation provides an optional three-family Level 5 composition: ML-DSA-87 (FIPS 204, Level 5, 4,627-byte signature, 2,592-byte public key), FALCON-1024 (Draft FIPS 206, Level 5, ~1,280-byte signature, 1,793-byte public key), and SLH-DSA-SHA2-256f (FIPS 205, Level 5, 49,856-byte signature, 64-byte public key). The Level 5 bundle totals approximately 55,763 bytes of ephemeral signatures — roughly 2.6× the Level 1 bundle — and signing is dominated by SLH-DSA-SHA2-256f at approximately 4× the Level 1 signing latency. This composition eliminates the "bounded by weakest at Level 1" footnote entirely: all three families are uniformly Level 5, and forgery requires breaking all three simultaneously. The Level 5 option is intended for high-value attestations (such as proof-of-life bindings for large Bitcoin holdings) where the additional signing cost is negligible relative to the value being protected. The wire format and 74-byte persistent footprint are identical for both compositions; only the algorithm_flags byte and the ephemeral signature bundle differ.

We have not committed to a default upgrade from Level 1 to Level 5 in the production deployment because the current bundle's effective security is adequate under all threat models we have considered, and the Level 5 signing cost would reduce batched throughput by approximately 4×. The reference commercial implementation intends to publish a migration notice with 90 days of lead time before any algorithm upgrade takes effect; specific notice obligations are governed by individual commercial license agreements.

6.4. Why This Construction

We address the natural question of why H33-74 uses a strict all-three signature policy rather than one of several simpler alternatives.

Single-family PQ signature + hash commitment. A single post-quantum signature scheme (e.g., ML-DSA-65 alone) over a hash commitment provides attestation with lower signing cost. However, it creates a single point of failure: one cryptanalytic advance against the chosen family produces a total break. If MLWE is broken, every attestation ever signed under ML-DSA is forgeable. The three-family construction exists precisely to avoid this single-point-of-failure property. The additional cost of the second and third signatures is amortized to microseconds per result by the batching construction of §5.

Threshold (2-of-3) schemes. A threshold policy that accepts an attestation when any two of the three families verify would provide graceful degradation if one family is compromised. However, it introduces the verification ambiguity we flag in §11.6: when one family is known to be broken, a verifier encountering a pre-rotation attestation must decide which two families to trust, and that decision has security implications that have not been formally analyzed. The strict all-three policy avoids this decision entirely. We consider threshold verification a direction worth exploring (§11.6) but premature to deploy before the rotation semantics are specified.

Aggregated signature schemes. An aggregate or multi-signature scheme that combines three signatures into one shorter object would reduce the ephemeral bundle size. No NIST-standardized post-quantum aggregate signature scheme currently exists. Building on non-standardized primitives would defeat the purpose of selecting three families from the NIST PQC program. If a suitable aggregate scheme is standardized in the future, the algorithm_flags mechanism (§11.8) can accommodate it without changing the wire format.

SNARK/STARK-based attestations. A ZK-SNARK or ZK-STARK proof of correct computation, without a signature bundle, could provide computation-result attestation with succinctness. However, SNARKs require a trusted setup (which H33-74 avoids), and STARKs at the scale required

for general computation verification are not yet practical for arbitrary computation types (§11.9). The proof-of-life construction in §9.9 demonstrates a STARK-based attestation for a specific computation (secp256k1 discrete log), but generalizing this to the full range of computation types H33-74 supports remains an open problem. Additionally, neither SNARKs nor STARKs provide assumption diversity — they rely on a single hardness assumption (knowledge-of-exponent or collision resistance, respectively).

Why these three families specifically. The three families were chosen to maximize assumption diversity across distinct mathematical domains. ML-DSA-65 relies on the Module Learning With Errors (MLWE) problem over structured polynomial modules. FALCON-512 relies on the Short Integer Solution (SIS) problem over NTRU lattice rings. Both are lattice-based, but they operate over different algebraic objects: MLWE works in the module lattice R_q^k where $R_q = Z_q[X]/(X^n+1)$, while NTRU-SIS works in the ring $Z_q[X]/(X^n+1)$ with the specific NTRU structure $f \cdot g^{-1}$ mod q. The best known attacks against each exploit structure-specific properties — BKZ lattice reduction for MLWE, and NTRU-specific combinatorial attacks for FALCON — that do not automatically transfer between the two constructions. The NIST PQC evaluation committee assessed these as independent candidates and selected both into the final standard precisely because their cryptanalytic attack surfaces are believed to diverge. SLH-DSA-SHA2-128f is categorically different: it has no algebraic structure whatsoever. Its security reduces entirely to the pre-image resistance of SHA2-256, a Merkle-Damgård hash function with thirty years of public cryptanalysis. A lattice breakthrough that simultaneously breaks both ML-DSA and FALCON has zero impact on SLH-DSA, because hash functions have no lattice structure to attack. This is why the three-family bundle provides meaningful assumption diversity rather than merely three instances of the same bet.

Rejected alternative families. Code-based schemes (Classic McEliece) were considered but rejected due to public key sizes exceeding 250 kilobytes, which would make the key distribution mechanism impractical for most deployments. Isogeny-based schemes (SIKE/SIDH) were eliminated after the Castryck-Decru attack (2022) broke the underlying isogeny problem in polynomial time. Multivariate schemes (Rainbow) were eliminated after Ward Beullens's attack (2022) broke Rainbow at all NIST-submitted parameter sets. The three surviving families — module lattices, NTRU lattices, and hash-based — represent the three hardness classes that have withstood the NIST PQC evaluation process without a practical break.

Summary. The all-three strict verification policy over NIST-standardized families was chosen because it (a) provides assumption diversity across three structurally distinct hardness classes, (b) uses only families that survived the NIST PQC evaluation without a practical break, (c) admits a simple security argument without threshold ambiguity, and (d) incurs a signing cost that is

amortized to microseconds per result by batching. We consider this the most defensible engineering choice given the current state of post-quantum standardization.

7. Efficiency

This section reports measured signing and verification costs from the reference implementation of H33-74 running on a production deployment. We present these numbers as evidence that H33-74 is practically deployable; we do not claim they are optimal or close to optimal. All figures are drawn from a benchmark run conducted on 2026-04-13 on AWS c8g.metal-48xl (Graviton4); raw benchmark logs and configuration details are retained for independent review (see Appendix A). These figures are not performance guarantees for customer deployments.

7.1. Signing Cost

Per-bundle signing time on a single core of an AWS Graviton4 (ARM Neoverse V2) running at production conditions:

ALGORITHM	TIME
ML-DSA-65	≈ 300 µs
FALCON-512	≈ 150 µs
SLH-DSA-SHA2-128f	≈ 5 ms
Three-family bundle (unbatched)	≈ 5.5 ms
Compact receipt assembly	< 5 µs

The three-family bundle is dominated by SLH-DSA-SHA2-128f. Per-core unbatched throughput is therefore approximately 180 bundles per second. These figures are from the Graviton4 metal benchmark of 2026-04-13 and differ from the single-shot times reported in §3.2 (760 µs / 183 µs / 24,083 µs), which were measured on a development workstation during the live Bitcoin mainnet anchoring. The approximately 5× improvement in SLH-DSA signing time (24 ms → 5 ms) reflects two factors: the Graviton4's hardware NEON crypto acceleration extensions, and the release-mode compilation flags (`opt-level=3` , `lto=fat` , `codegen-units=1`) which enable aggressive inlining of

the SHA2-256 hash chains that dominate SLH-DSA's signing cost. The Graviton4 figures are the authoritative production numbers.

7.2. Batched Throughput

A production deployment running the reference implementation on a 192-vCPU AWS c8g.metal-48xl instance (Graviton4, us-east-1), with batches of N = 32 results per three-family bundle and 96 concurrent workers, sustained over a 30-second measurement window the following throughput numbers on 2026-04-13:

CONFIGURATION	AUTH/SEC	PER-AUTH	PER-BATCH (32 USERS)
Metal, full pipeline, raw DashMap data structure	**2,216,488**	35 µs	1,128 µs
Metal, full pipeline, CacheeEngine overlay (L0 + CacheeLFU + DashMap)	**2,216,488**	35 µs	~1,130 µs

The overhead of the CacheeEngine overlay relative to the raw DashMap data structure is approximately 1.87% in the full pipeline — inside run-to-run variance, and we consider the two configurations operationally equivalent. The CacheeEngine adds an admission sketch and observability instrumentation on top of the underlying concurrent hash map; the cost of this overlay is statistically invisible at pipeline scale.

The batched configuration amortizes the three-family signing cost described in §7.1 across 32 results per signing event. The remaining pipeline time is dominated by the fully homomorphic encryption inner product computation that produces the data being attested — H33-74's signing overhead is not the pipeline bottleneck.

At a sustained 2,216,488 authentications per second and commodity cloud on-demand pricing of approximately $2.30 per instance-hour, the hardware cost of a single signed authentication is approximately 2.9×10^{-10} United States dollars. We offer this figure as an existence proof that H33-74 is economically deployable at high volume; we do not claim it is the minimum achievable cost.

7.3. Pipeline Component Breakdown

The full end-to-end pipeline latency on the Apr 13 metal run decomposes as follows (per 32-user batch). **Important:** the three signing operations run in parallel, not sequentially. The wall-clock

bottleneck is SLH-DSA at ~4,891 µs. With batching, one three-family parallel signing event covers 32 results, yielding an amortized cost of ~4,891 µs ÷ 32 ≈ 153 µs per result for the signing component, plus digest and overhead. The batch attestation wall-clock time of 189 µs per result reflects this parallel-then-amortize model. The sub-rows in the table below show per-signature CPU time, not wall-clock contribution to the batch.

STAGE	WALL-CLOCK LATENCY	% PIPELINE
FHE batch verify (32 users)	937 µs	83%
Batch attestation (wall-clock)	189 µs	17%

PARALLEL SIGNING BREAKDOWN (CPU TIME PER SIGNATURE)	CPU TIME
SHA3-256 digest	~1 µs
ML-DSA-65 sign	~268 µs
FALCON-512 sign	~143 µs
SLH-DSA sign (wall-clock bottleneck)	~4,891 µs
ML-DSA-65 verify	~112 µs

STAGE	LATENCY	% PIPELINE
ZKP cached lookup (CacheeEngine L0)	1.9 µs	0.2%
Total batch latency (32 users)	**1,128 µs**	
Per-auth latency	**35 µs**	

The fully homomorphic encryption computation dominates the pipeline at 83%. The three-family signing bundle contributes approximately 17% — reduced from earlier measurements by the restoration of hardware NEON crypto acceleration. The CacheeEngine L0 lookup is statistically invisible at the pipeline scale.

7.4. Optimizations Discovered During Load Testing

We owe the reader a candid account of the cache layer's behavior during initial integration, because the first attempt produced a throughput figure that was approximately one-ninth of the raw baseline, and the investigation that followed uncovered three distinct instrumentation bottlenecks in the cache's observability code. All three produced correct output at every concurrency level; the issue was throughput degradation caused by lock contention in the instrumentation path, not in the cache logic itself. The optimizations are in production now, and the 2,216,488 auth/sec figure in §7.2 reflects the cache with all three optimizations applied. We describe the investigation here because honesty about production performance investigation is more useful to the community than polished benchmark claims.

The first integration run, under the same 96-worker sustained load that produced the raw baseline of 2,216,488 auth/sec, produced only 183,828 auth/sec when the cache layer was interposed. This is a 12× regression, and on its face would suggest the cache layer was fundamentally misdesigned.

The investigation revealed that all three root causes were contended write locks in the cache's instrumentation code, not in the admission sketch or the cache's main data structures. No incorrect output was produced at any stage; the cache returned correct results throughout, but at a fraction of the expected throughput. We describe the three optimizations individually.

Optimization 1: Statistics instrumentation lock contention.

The cache's internal statistics counters were protected by a shared lock that serialized the entire hot path under high concurrency. The fix replaced the shared lock with lock-free counters. Throughput rose to 395,816 auth/sec (a 115% improvement, but still far below the baseline).

Optimization 2: Metrics collector lock contention.

The cache's metrics collection layer held a redundant shared lock around an already-atomic data structure. Removing the redundant lock and switching non-critical bookkeeping to skip-on-contention raised throughput to approximately 1,400,000 auth/sec.

Optimization 3: Pattern detector hot-path write lock.

A workload pattern detector was taking a write lock on every cache read, despite only needing the data periodically. Adding a sampling rate reduced lock contention substantially with negligible impact on pattern detection quality. After this fix the final throughput settled at 2,216,488 auth/sec, which is 1.87% below the baseline and inside normal run variance.

The cache's admission control mechanism was not involved in any of the three bottlenecks and remained fully active throughout the final 2,216,488 auth/sec run.

We take two lessons from this investigation. First, instrumentation and observability code must be lock-free at high concurrency — a shared lock on the hot path is almost always wrong at scale, even when the code inside the lock is trivial. Second, load testing at the target concurrency is not optional. None of these bottlenecks were visible at low-concurrency benchmarks.

The reference cache implementation with all three optimizations applied is available under a commercial license.

7.5. Verification Cost

Verification time on a single core of a commodity client (Apple M4 Max laptop) for a single three-family attestation, including the SHA3-256 recomputation, the receipt structural checks, and the three signature verifications:

OPERATION	TIME
SHA3-256 recomputation	< 1 µs
ML-DSA-65 verify	≈ 150 µs
FALCON-512 verify	≈ 100 µs
SLH-DSA-SHA2-128f verify	≈ 300 µs
Structural checks	< 10 µs
Full verify, wall-clock	**≈ 593 µs**

The verification cost is dominated by the three individual signature verifications, which are independent and parallelizable.

7.6. Cache Memory Footprint

The reference cache's admission sketch is a Count-Min Sketch whose memory footprint is invariant in the cardinality of the keyspace being tracked. A standalone benchmark measured the sketch's memory footprint at three keyspace sizes and compared it with the baseline memory of a concurrent hash map storing the same keys:

KEYSPACE	ADMISSION SKETCH	DASHMAP BASELINE	RATIO
100 K	512.17 KiB	6.20 MiB	12.4×
1 M	512.17 KiB	61.99 MiB	123.9×
10 M	512.17 KiB	619.89 MiB	1,239×

The sketch's constant footprint is a direct consequence of its probabilistic construction. The admission decision at 10 million keys costs the same memory as the admission decision at 100 thousand keys, and we consider this memory-scaling property to be a defining feature of the cache design for deployments that track very large keyspaces.

7.7. Persistent Footprint

The persistent footprint is 74 bytes per attestation for single results, and 74 bytes per batch plus $32 \cdot \lceil \log_2 N \rceil$ bytes per leaf inclusion proof for batched attestations. At $N = 2^{20}$, this is 74 bytes for the root attestation and 640 bytes per leaf inclusion proof. The ephemeral footprint (the raw signature bundle) is approximately 21 kilobytes per bundle and is held in an off-chain signature store indexed by the receipt's commitment.

7.8. Functional Verification

We report the results of five independent functional proof tests that verify H33-74's cryptographic properties hold end-to-end with real data, not just synthetic test vectors. All five tests pass in under 10 milliseconds on a commodity laptop (Apple M4 Max). The test source is published in the reference implementation crate at `tests/functional_proof.rs`.

Test 1: Binding proof.

A real document (a 59-byte SEC filing excerpt) is minted as an ArchiveSign primitive. The test independently verifies: (a) H33-74's h field equals SHA3-256 of the document, (b) verify_binding returns true for the original document, (c) the signing message equals SHA3-256 of the serialized primitive bytes, and (d) the OP_RETURN script contains exactly the signing message at bytes [2..34]. The full chain document → H(r) → primitive.h → SHA3-256(S) → OP_RETURN is verified at every link. Persistent footprint: 74 bytes.

Test 2: Tamper detection.

Six tamper scenarios are tested against a MedVaultPhi primitive binding a patient record: (a) flipping one bit of the document causes verify_binding to reject, (b) an empty document rejects, (c) a completely different document rejects, (d) flipping one bit in the serialized primitive changes the signing message, (e) flipping the computation type byte changes the signing message, and (f) the compact receipt rejects verification against a signing message that does not match the original. All six tampering attempts are detected.

Test 3: Domain separation.

Twelve computation types are tested with identical payload data, identical timestamps, and identical nonces. The test confirms that all 12 produce mutually distinct signing messages — 66 cross-type comparisons, zero collisions. This proves that cross-domain replay (using a BiometricAuth attestation as if it were a BitcoinUtxo attestation) is cryptographically impossible: the computation type byte is mixed into the primitive before hashing, and any change to it produces a completely different signing message.

Test 4: ZK over primitive tag.

A BitcoinUtxo primitive is constructed from a 64-byte proof-of-reserves computation result. The test simulates the six constraints that a STARK circuit would enforce over a committed 58-byte private witness, given only the 32-byte public signing message: (a) SHA3-256(witness) equals the public input, (b) the version byte is 0x01, (c) the computation type is 0x06, (d) the content hash matches the claimed computation result, (e) the timestamp is within the last hour, and (f) the nonce is non-zero. A Fiat-Shamir challenge is derived from a transcript binding the public input. All six constraints pass. This test establishes the constraint structure for a future STARK proof that would prove knowledge of a valid primitive without revealing its nonce or timestamp.

Test 5: Receipt-to-chain verification.

A LegalEvidence primitive is minted, signed, and attested. The test independently verifies: (a) the compact receipt is exactly 42 bytes with version 0x01 and algorithm flags 0x07 (three families), (b) the signing message recomputed from the primitive bytes matches the on-chain hash, (c) the receipt's verification_hash recomputed from the signing message, public keys, and signatures matches the receipt's stored hash at bytes [1..33], (d) H33-74's content binding to the original document holds, and (e) the full verification chain — document → content hash → primitive → signing message → OP_RETURN, and separately signing message + public keys + signatures → verification hash → receipt — is intact at every link.

7.9. Stress Tests and Limits

We report the results of ten additional tests that probe H33-74's behavior under adversarial inputs, concurrency, scale, and timing analysis. All ten tests pass. The test source is published in the

reference implementation crate at `tests/stress_and_limits.rs`. Combined with the five functional proofs in §7.8, the reference implementation passes 100 tests with zero failures across the full crate.

Test 1: Maximum input size.

Computation results ranging from 1 byte to 10 megabytes are minted as primitives. Every input produces a 58-byte primitive with a correct SHA3-256 content hash. H33-74 is size-invariant: a 10 MB document and a 1-byte status code produce identically sized attestations.

Test 2: Empty and malicious inputs.

Empty input is rejected. All-zero, all-0xFF, embedded-null, and repeating-pattern inputs are accepted and produce correct primitives. Invalid deserialization inputs are rejected: wrong sizes (0, 57, 59, 1000 bytes), wrong version byte (0x02), and unregistered computation type (0xFE) all return errors.

Test 3: Nonce uniqueness under concurrency.

Ten thousand primitives are generated in rapid succession with identical payload data. Every nonce, every primitive, and every signing message is unique — zero collisions across 10,000 mints. At a nonce space of 2^{128}, a birthday collision requires approximately 2^{64} mints; this test confirms the random source produces no degenerate output at the tested scale.

Test 4: Timestamp field limits.

Timestamps of 0 (Unix epoch), 1, u64::MAX (year 584,942,417), and a known 2026 reference value are all preserved exactly in big-endian encoding. The timestamp field does not overflow, truncate, or silently wrap at any tested value.

Test 5: Cross-platform determinism.

A primitive constructed from the SPEC.md known-answer vector (version 0x01, type BiometricAuth, commitment [0xAB; 32], timestamp 1744156800000, nonce [0xCD; 16]) produces byte-identical output to the published specification. The signing message matches the hardcoded SHA3-256 digest. This test confirms that the serialization is deterministic across platforms; the same test runs on x86, ARM, and WASM targets.

Test 6: Merkle tree at scale.

A Merkle tree with 65,536 leaves (2^{16}) is constructed. Inclusion proofs for the first, middle, and last leaf all verify correctly. Proof depth is 16 levels (512 bytes per proof). A wrong-leaf test confirms that an incorrect

leaf fails verification. The construction's domain separation (0x00 leaf prefix, 0x01 internal prefix) prevents second-preimage attacks at the tested scale.

Test 7: Receipt forgery resistance.

A legitimate receipt is verified, then six forgery attacks are attempted: (a) wrong public key, (b) wrong signature, (c) wrong signing message, (d) swapped signature families (Dilithium signature presented as FALCON), (e) truncated signature, and (f) bit-flip in the verification hash. All six are rejected. The receipt's verification_hash commits to the exact signing message, the exact public keys, and the exact signatures via a domain-tagged, length-prefixed SHA3-256 digest; any modification to any component produces a different hash.

Test 8: Constant-time verification.

The receipt verification path is timed over 10,000 iterations for both valid and invalid inputs. Valid verification: 2,155 microseconds per iteration. Invalid verification (wrong signing message): 2,162 microseconds per iteration. The ratio is 1.003 — within 0.3% — confirming that the verification path does not leak timing information correlated with input correctness. The implementation uses `subtle::ConstantTimeEq` for all hash comparisons to prevent side-channel attacks.

Test 9: Signature size validation against NIST specifications.

ML-DSA-65 (FIPS 204): 3,309 bytes. FALCON-512 (Draft FIPS 206): bounded by 666 bytes. SLH-DSA-SHA2-128f (FIPS 205): 17,088 bytes. Total three-family bundle: 21,063 bytes. Distillation ratio: 284.6:1 (21,063 ephemeral → 74 persistent). All sizes match the published NIST parameter specifications.

Test 10: Size invariance.

Six inputs ranging from 1 byte to 1 megabyte, across six computation types, all produce primitives of exactly 58 bytes, signing messages of exactly 32 bytes, and compact receipts of exactly 42 bytes. The output size is independent of the input size and the computation type. This is the defining property that makes H33-74 viable in storage-constrained environments: the attestation cost is O(1) in data volume.

7.10. Live Bitcoin Anchor Test

The live Bitcoin mainnet anchor test is presented in §3.2 as a construction proof. The transaction ID is `7f8d9ef2d5625d7e3acbc269daac21087ce6b7d77f8e4ec369aabdcdb028b4a7` and is independently verifiable on any Bitcoin full node or block explorer.

7.11. NIST KAT Validation

We report the results of 18 Known Answer Test (KAT) validations against the three post-quantum signature families used in H33-74's three-family bundle. These tests verify that the pqcrypto crate implementations — which wrap the PQClean C reference implementations — produce correct keypairs, valid signatures, and proper rejection of tampered or misattributed signatures. All 18 tests pass. Combined with the 5 functional proofs (§7.8), 10 stress tests (§7.9), and the live mainnet anchor (§3.2), the reference implementation passes 118 tests with zero failures across the full crate.

FAMILY	NIST STANDARD	PUBLIC KEY	SIGNATURE	SIGN/VERIFY	TAMPER REJECT	WRONG KEY
ML-DSA-65	FIPS 204	1,952 B	3,309 B	PASS	PASS	PASS
FALCON-512	Draft FIPS 206	897 B	649–661 B (max 666)	PASS	PASS	PASS
SLH-DSA-SHA2-128f	FIPS 205	32 B	17,088 B	PASS	PASS	PASS

Per-family results:

ML-DSA-65 (FIPS 204): Keygen produces 1,952-byte public keys matching the NIST specification. Detached signatures are exactly 3,309 bytes. Sign/verify round-trip passes. Tampered signatures (single bit flip) are rejected. Signatures under a different keypair are rejected. Signatures verified against a different message are rejected. Six tests, all pass.

FALCON-512 (Draft FIPS 206 / FN-DSA): Keygen produces 897-byte public keys matching the NIST specification. Detached signatures are variable-length, bounded by 666 bytes. Across 100 signatures on distinct messages, observed signature sizes ranged from 649 to 661 bytes — all within the NIST-specified bound. Sign/verify round-trip passes. Tampered signatures are rejected. Wrong-key signatures are rejected. Six tests, all pass.

SLH-DSA-SHA2-128f (FIPS 205): Keygen produces 32-byte public keys matching the NIST specification. Detached signatures are exactly 17,088 bytes. Sign/verify round-trip passes.

Tampered signatures (bit flip at byte offset 100) are rejected. Wrong-key signatures are rejected. Five tests, all pass.

Three-family independence: The same 32-byte signing message was signed by all three families independently. Each signature verifies only under its own public key. Total ephemeral bundle: 21,051 bytes (3,309 + 654 + 17,088). Distillation to 74 bytes: 284.5:1. The three families are cryptographically independent — no signature from one family is accepted by any other family's verifier.

The test source is published in the reference implementation crate at `tests/nist_kat.rs`. All public key sizes, signature sizes, and verification behaviors match the NIST-published specifications for each algorithm family.

8. Commercial Deployment and Licensing Model (NON-NORMATIVE)

This section describes the reference commercial implementation and is non-normative. The cryptographic construction of Sections 3–6 is independent of the deployment model described here.

H33-74, as a primitive, is deployment-agnostic. This section describes the deployment model adopted by the H33.ai reference commercial implementation, not because it is the only possible deployment model, but because it is the model that admits the best composition of H33-74's security properties with the commercial licensing and billing structure required to sustain the reference implementation over time. The model is intentionally structured so that the licensing enforcement does not contradict H33-74's trustlessness properties, and so that customers who adopt the commercial reference implementation are not operationally dependent on infrastructure operated by the commercial vendor.

8.1. Customer-Hosted Execution

In the commercial deployment, the signing binary runs entirely on customer-owned infrastructure. No primitive operation — minting, signing, verification, or storage — requires a network call to infrastructure operated by the commercial reference implementer (H33.ai, Inc.). An exception exists at initial deployment: the TEE binary performs a one-time key provisioning handshake with the

vendor's attestation service to unlock the signing keypair. After this one-time provisioning event, no further vendor contact is required for any primitive operation. The customer installs the licensed binary on their own hardware, typically inside a Trusted Execution Environment (AWS Nitro Enclave, Intel SGX, AMD SEV-SNP, or equivalent), and operates it under their own security posture. License revocation is enforced via a revocation primitive published from a well-known vendor address; binaries check periodically and fail closed if their fingerprint appears in the revocation tree.

This architectural choice is motivated by three requirements that recur across commercial and regulated deployments. First, many customers subject to data-residency regulations (GLBA, HIPAA, GDPR, ITAR, FedRAMP, PCI-DSS) cannot legally send sensitive data to a third-party service, regardless of the security guarantees the third party might offer. A self-hosted deployment removes this objection entirely: the customer's data never leaves the customer's perimeter. Second, customers with performance-sensitive workloads prefer to minimize network round-trips by running the signer adjacent to the computation being attested, which is possible only with self-hosted execution. Third, the self-hosted model removes the commercial vendor as an operational dependency — customers whose primitives are minted on their own hardware continue to function even if the vendor experiences an outage, is acquired, is sued, or ceases operations entirely.

We note that this deployment model is how most commercial cryptographic infrastructure actually ships in the enterprise market. Hardware Security Modules (Thales Luna, AWS CloudHSM physical appliances, Fortanix DSM, Utimaco), specialized cryptographic appliances, compliance-grade key management systems, and most enterprise cryptographic signing products ship as on-premise or customer-cloud-hosted artifacts with vendor licensing and support, not as hosted SaaS APIs. The primitive's commercial deployment follows the same pattern.

8.2. Trusted Execution Environment Protection

The licensed binary is distributed as an encrypted image that can only be decrypted inside a supported TEE. At first launch, the TEE attests its measurement chain to the commercial vendor's release signer, receives a per-customer decryption key provisioned via the TEE's remote attestation protocol (using ML-KEM-768 [@fips203] for the key encapsulation), and unlocks the binary for execution. The three-family signing keypair for the customer's deployment is generated inside the TEE at first launch and never leaves it; the customer's own operators cannot extract it through any software interface, and the commercial vendor cannot extract it without the TEE's cooperation.

The TEE protection serves two distinct purposes. The first is standard key custody: the signing keys are protected from extraction by root-level access to the host machine, defending against insider threats, supply-chain attacks on the operating system, and compromise of the customer's own

system administrators. The second is license enforcement: the binary verifies, at startup and periodically during operation, that the current license is valid and has not been revoked, and refuses to run if either condition fails.

We note explicitly that TEE protection is not cryptographically unconditional. Intel SGX has been compromised multiple times by side-channel attacks (Foreshadow, Plundervolt, SGAxe, and several other published breaks), and AMD SEV-SNP has had its own history of partial breaks. The primitive's overall security posture does not rely on the TEE as a trust root — the three-family signature bundle described in §4 remains valid even if the TEE is compromised, because the signing operation itself is performed by standard post-quantum algorithms whose security does not depend on the TEE. The TEE protects the signing keys from extraction and the license state from tampering; it does not provide the cryptographic guarantees of the attestation itself. Primitive verification by a third party is independent of TEE status — a verified primitive remains verified whether or not the original signer's TEE was subsequently compromised.

8.3. License Fingerprint Embedding

At commercial agreement signing time, the commercial vendor generates a per-customer license fingerprint derived from the customer identifier, the license epoch (the date the agreement became active), and the SHA-256 hash of the executed commercial agreement document. The fingerprint is an 8-byte value computed as HMAC-SHA256(k_{vendor}, customer_id || epoch || agreement_hash), truncated to 8 bytes, where k_{vendor} is a symmetric key held only by the vendor's licensing infrastructure. The 8-byte (64-bit) truncation is sufficient for customer attribution within realistic deployment populations. At 2^{16} (~65,000) simultaneous active customers — well beyond any near-term deployment scenario — the birthday-bound collision probability is approximately 2^{-33}, which is negligible. At larger populations, fingerprint collisions would affect only billing attribution (not cryptographic security), and the vendor can resolve ambiguities using the customer_id and epoch inputs available in the licensing database.

The fingerprint is embedded into the encrypted binary at shipment time. Every primitive minted by the customer's deployment encodes the fingerprint into the first 8 bytes of H33-74's 16-byte nonce field; the remaining 8 bytes are filled with fresh random bytes from the cryptographically secure random source. To external observers examining a single primitive, the fingerprinted nonce is computationally indistinguishable from a uniformly random nonce (under the standard PRF assumption on HMAC-SHA256, the truncated output is indistinguishable from uniform random bytes); repeated primitives from the same deployment share the same 8-byte prefix, which is observable by an adversary comparing multiple primitives but does not reveal the customer's

identity without the vendor's HMAC key. The fingerprint reveals nothing about the customer's identity or the commercial terms of the license to any party not holding k_{vendor}. We note, however, that an observer with access to the public primitive store who compares multiple primitives can cluster them by deployment — grouping all primitives that share the same 8-byte nonce prefix — without being able to identify which customer produced the cluster. This is a clustering-without-identification property: the observer learns that a set of primitives came from the same deployment, but not whose deployment it is. Customers with linkability concerns can mitigate this by rotating deployments or requesting periodic fingerprint refresh from the vendor. To the vendor holding k_{vendor}, the fingerprint is deterministically computable from the customer's identifying data, and primitives bearing the fingerprint are attributable to the corresponding customer.

FIELD	DATA	LOCATION	ON-CHAIN EXPOSURE	THIRD-PARTY EXPOSURE	NOTES
customer_id	Customer identifier string	Vendor licensing DB only	None	None	Never embedded in primitive directly
license_epoch	Date agreement became active	Vendor licensing DB only	None	None	Input to HMAC, not stored in primitive
agreement_hash	SHA2-256 of executed agreement document	Vendor licensing DB only	None	None	Input to HMAC, not stored in primitive
k_{vendor}	Vendor HMAC symmetric key	Vendor licensing infrastructure only	None	None	Never leaves vendor; used to compute and verify fingerprints

FIELD	DATA	LOCATION	ON-CHAIN EXPOSURE	THIRD-PARTY EXPOSURE	NOTES
fingerprint	HMAC-SHA256(k_{vendor}, customer_id ‖ epoch ‖ agreement_hash), truncated to 8 bytes	TEE only at customer site; vendor DB	None directly — embedded in nonce field	Not recoverable without k_{vendor}	Indistinguishable from random in single-primitive observation
nonce field (16 bytes)	Bytes 0–7: fingerprint. Bytes 8–15: fresh CSPRNG random	Inside primitive S (customer TEE + primitive store)	Not directly — only SHA3-256(S) is on-chain	Observable in primitive store but not decomposable without k_{vendor}	External observer sees only the hash of the full primitive, not the nonce itself
signing message m(S)	SHA3-256(full 58-byte primitive)	On-chain permanently	32 bytes — fully public	Fully public	This is what goes into OP_RETURN or Taproot tweak
primitive S	Full 58-byte tuple	Customer TEE + public primitive store	Not directly — only m(S) is on-chain	Public via primitive store	Retrievable by anyone with m(S) as lookup key
compact receipt R	42-byte receipt	Off-chain receipt store	None	Public via receipt store	Indexed by R.c — the verification digest
signature bundle σ	~21,063 bytes (ML-DSA + FALCON + SLH-DSA)	Off-chain ephemeral signature store	None	Public via signature store indexed by R.c	May be destroyed after verification at signer's discretion

FIELD	DATA	LOCATION	ON-CHAIN EXPOSURE	THIRD-PARTY EXPOSURE	NOTES
billing count	Count of matched primitives per customer per period	Vendor chain scanner + credit ledger	Derivable by vendor from on-chain data	Not directly accessible	Both parties derive independently from same public ledger
credit balance	Prepaid mints remaining	Customer TEE + vendor credit ledger	None	None	Decremented at mint time by licensed binary

8.4. Bitcoin Chain Metering

Every primitive produced by a customer's deployment is anchored to the Bitcoin blockchain — via Taproot key-path tweak or OP_RETURN — for reasons intrinsic to the primitive protocol (discussed in §9.1 on Bitcoin UTXO Attestation) rather than for billing reasons. The anchoring serves the customer's own integrity workflow: it establishes an immutable public timestamp for the attestation that can be independently verified by any third party with Bitcoin network access.

The commercial vendor operates a Bitcoin chain scanner that reads Taproot commitment tweaks (or OP_RETURN outputs) from Bitcoin full nodes, resolves each observed signing message m(S) against the public primitive store to retrieve the full 58-byte primitive S, extracts the fingerprint from the nonce field of S, and tests it against the license fingerprints of the active customer population. Every match is a primitive attributed to the corresponding customer. The chain scanner serves as the canonical usage meter — the authoritative record of how many primitives a customer has minted. Commercially, customers acquire credits in advance, and the licensed binary decrements credits at mint time. The chain scanner reconciles the credit ledger against the on-chain record at the end of each billing period, ensuring that the credit-based prepayment and the chain-observed usage agree.

This billing mechanism has two structurally important properties that distinguish it from conventional commercial API metering.

First, the billing ledger is the Bitcoin blockchain itself. The commercial vendor and the customer compute the billing count from the same public data, using the same deterministic fingerprint matching function, and must always agree on the result. Billing disputes are structurally minimized: if the two counts disagree, the discrepancy can be traced to the customer's internal accounting or

the vendor's fingerprint matching, because both sides derive their counts from the same authoritative source. The Bitcoin blockchain's immutability guarantees that historical billing counts cannot be retroactively altered by either party.

Second, the customer cannot under-report usage without simultaneously failing to anchor their own primitives, which defeats the customer's own integrity workflow. The anchoring to Bitcoin is the customer's independent reason for using H33-74 in the first place; the same anchoring action that provides the customer with their integrity guarantees also constitutes the billing record. The customer's self-interest is therefore aligned with the billing integrity — they cannot cheat the vendor without simultaneously cheating themselves of the protection they are paying for.

We consider these two properties to be the primary structural reason that the self-hosted commercial model is viable at scale, though we acknowledge that operational edge cases (customer key rotation, TEE migration, network partitions during anchor broadcast) require additional engineering beyond what the structural argument alone provides.

In the event the commercial vendor ceases operations, customers retain primitive verification capability through the open-source h33-74-verifier crate (Apache 2.0), provided that the public keys of the original signer remain available and the verifier crate remains accessible. The on-chain anchors (whether Taproot tweaks or OP_RETURN outputs) remain on the Bitcoin chain permanently. The three-family signing keys were generated inside the customer's own TEE and never left it; the customer's deployed binary continues to function. Billing simply stops. No primitive signed by a surviving deployment becomes unverifiable solely due to the vendor's absence, provided the signer's public keys and the open-source verifier remain available.

8.5. Module Breakout

The commercial deployment ships as a set of separately-licensed modules, each with its own enforcement surface. Four modules comprise the minimal deployment footprint:

h33-74-core

The canonical commitment construction itself, including the wire-format assembly, signing message derivation, nonce generation, computation type registry enforcement, Merkle aggregation for batched mode, and CompactReceipt assembly. Subject to the per-mint

h33-3key-signer

The three-family post-quantum signer described in §4, including the parallel signing implementation, the CompactReceipt distillation, the ephemeral signature store indexing, and the key rotation policy engine. Typically deployed together with h33-74-core as a bundled pair. Commercial license required.

economic model described in Appendix B. Commercial license required.

h33-cachee-core

The in-process two-tier cache described in §7, including the CacheeLFU admission sketch and observability instrumentation. Subject to a separate per-operation royalty structure documented in the commercial license agreement. Commercial license required.

h33-74-verifier

A minimal primitive verifier crate released under the Apache License 2.0. Any party may verify any primitive locally, using only public keys and local cryptography, without acquiring a commercial license. The verifier is open source because verification is a pure function of public data, and because the trustlessness of verification is a defining property of H33-74.

The verifier's open-source release establishes that verification of any primitive, past or future, is independent of the commercial vendor's continued existence. A customer holding primitives that were minted under a commercial license retains the ability to verify those primitives indefinitely, even in the absence of the commercial vendor, using only the open-source verifier and the publicly-distributed public key set of the original signer.

9. Applications

H33-74 is a general-purpose primitive and admits applications in any domain that requires a post-quantum, fixed-width, batch-friendly attestation of a computation result. We describe nine applications we have implemented, and enumerate what the computation type byte is set to in each.

The application we spend the most space on is post-quantum Bitcoin UTXO attestation, because this paper is addressed in part to the Bitcoin developer community and because the contrast with script-layer constructions like QSB is most instructive in the Bitcoin context.

9.1. Bitcoin UTXO Attestation (t = 0x06)

Motivation. A UTXO is produced by an off-chain computation — say, a privacy-preserving fraud score, a regulated financial settlement, or a proof-of-reserves computation — and the holder wishes to prove, at any point in the future, that the UTXO was produced by a specific computation whose result r is known and whose canonical encoding hashes to a specific digest H(r). A Bitcoin on-chain commitment (Taproot tweak or OP_RETURN) can timestamp the UTXO but does not by itself prove

computation provenance. The primitive provides the provenance certificate by binding the result digest to a three-family post-quantum signature at a specific timestamp.

Construction. The computation producer runs Mint(0x06, r) → S where r is the canonical encoding of the computation's result, then Sign(S, ...) → (R, σ), and commits the 32-byte signing message m(S) = SHA3-256(S) to a Bitcoin transaction — either as a Taproot key-path tweak on the output ($Q = P + H(P \| m(S)) \cdot G$, adding zero marginal transaction weight) or as a standard OP_RETURN output. The transaction is broadcast through the standard Bitcoin mempool and does not require a private mempool service. The holder retains (r, S, R, σ) off-chain.

Verification. A third party who wishes to verify the provenance of the UTXO obtains (r, S, R) from the holder (or from the signer's public primitive store), obtains the raw signature bundle σ from the signer's ephemeral signature store indexed by R.c, and runs Verify. In addition to the primitive verification, the third party fetches the Bitcoin transaction and verifies the on-chain commitment — either by recomputing the Taproot output key from the internal key and the signing message, or by checking that SHA3-256(S) matches the OP_RETURN payload and that the transaction is confirmed in the block height claimed by the holder. A successful verification proves, under the three-family security argument of Section 6 composed with the security of the Bitcoin ledger, that the UTXO was anchored to the Bitcoin chain at the claimed block height as the output of a computation of type 0x06 with computation result r, whose canonical encoding hashes to the attested content digest h = H(r).

Relationship to Script-Layer Post-Quantum Constructions. The Bitcoin script locking the UTXO is orthogonal to the primitive attestation and can be any script the holder chooses. If the holder wishes to additionally protect the UTXO against script-layer signature forgery by a quantum adversary, they may use a QSB or BINOHASH locking script. The two constructions compose freely and neither interferes with the other. We discuss this composition in Section 10.

Cost. The primitive attestation contributes a 32-byte commitment to the Bitcoin transaction — zero marginal bytes via Taproot key-path tweak, or 34 bytes via OP_RETURN. At the global aggregate rate (Appendix B), the attestation fee paid to the commercial reference implementation is $0.025 per mint at launch (decreasing as global adoption grows) — illustratively several orders of magnitude lower at launch than the published QSB per-transaction cost estimates (noting that QSB economics may evolve as the construction matures). The Bitcoin transaction itself is standard and costs whatever the current Bitcoin mempool fee market sets for a standard OP_RETURN spend.

We note that this application requires no coordination with Bitcoin miners, developers, or node operators. H33-74 is already deployable on Taproot-enabled Bitcoin today, using only existing

transaction formats and existing relay policy. Any Bitcoin holder with access to the reference signing binary can begin producing post-quantum-attested UTXO commitments immediately.

9.2. HTTP API Response Attestation (t = 0x0C)

An HTTP server produces a response body and runs Mint(0x0C, response body) → S, then Sign(S, ...) → (R, σ). The server installs four HTTP response headers: X-H33-Primitive containing the 32-byte signing message, X-H33-Receipt containing the 42-byte compact receipt, X-H33-Algorithms containing the FIPS canonical algorithm identifiers of the three families, and X-H33-Primitive-Ts containing the millisecond timestamp. A client verifies the response body locally against the primitive and performs the three signature checks against the server's published public keys. The verification performs no network calls beyond the initial HTTP GET.

9.3. AI Inference Provenance (t = 0x0D)

A machine-learning inference engine produces an output — a classification, a prediction, a generated text — together with the model identifier, the model weight hash, the input hash, the output hash, and the inference timestamp. The inference engine runs Mint(0x0D, concatenation of those fields) → S and signs it. The resulting attestation proves that the named output was produced by the named model on the named input at the named time. An output lacking an AI-inference-domain primitive attestation is, by the converse, provably not an attested inference from any system implementing this application.

9.4. Capture-Time Media Authenticity (t = 0x0E)

A digital media capture device (camera, microphone, CCTV) is provisioned at manufacture time with a three-family post-quantum signing key set. At capture time, the device's firmware computes the SHA3-256 of the raw sensor data before any post-processing, runs Mint(0x0E, sensor data), signs the primitive, and embeds the 32-byte signing message and 42-byte compact receipt into the media file's metadata or a sidecar file. Any downstream verifier can check that the media was produced by the named device class at the named time and has not been modified since capture. Media artifacts lacking a capture-time media primitive are provably not capture-time originals from any device implementing this application. We note in passing that this is an answer to the content-authenticity problem that has recently gained public attention under the name "deepfake detection."

9.5. Legal Evidence Chain of Custody (t = 0x0F)

A digital artifact enters legal evidence. Each access, transfer, or transformation of the artifact is recorded as a primitive of type 0x0F, whose h field binds to the event details and whose prior-primitive reference (via a primitive-referencing-primitive construction) binds the event to the preceding event in the custody chain. The complete custody chain from creation to courtroom presentation is cryptographically provable from the directed acyclic graph of primitive commitments alone, without reliance on the integrity of any centralized custody database.

We note an important limitation. H33-74 attests the computation result, not input authenticity. A primitive anchored to a forged document is a cryptographically valid attestation of a forged document. The primitive's value in legal contexts is establishing chain of custody and detecting post-attestation modification — it does not substitute for the identity verification and source authentication that establish the authenticity of the underlying document before attestation.

To mitigate this limitation, a document attestation primitive may incorporate both a cryptographic commitment to the document content and a cryptographic commitment to a verified identity credential of the submitting party — for example, a ZK-KYC proof or a government-issued identity attestation — via the primitive-referencing-primitive construction described above. In this configuration, a forged deed has a verified identity cryptographically linked to its submission event. The forger is now bound to the forgery by the same three-family post-quantum signature that protects the attestation itself. Combined with Bitcoin anchoring, this produces a millisecond-precision, quantum-resistant, immutable record of who submitted the document for attestation and when — providing cryptographic provenance evidence that composes with traditional notarization procedures to strengthen chain-of-custody arguments.

9.6. Federated Machine-Learning Mesh

A federation of machine-learning agents exchanges primitive content-address tags instead of the underlying data payloads. Each payload — model weights, context windows, retrieval chunks, gradient updates, tool invocation results — is bound to a primitive at the moment of production, and H33-74's 32-byte signing message functions as the network-level identifier for the payload. Receiving agents resolve tags against a local content-addressed store and fetch payloads only on demand. The aggregate per-agent bandwidth cost is decoupled from the aggregate data volume of the distributed learning workload. This application treats the primitive as a content-address tag rather than as a per-computation attestation, and we find the generalization useful in practice.

9.7. Authenticated Relay Messaging (t = 0x10)

Two systems that need to exchange authenticated messages — whether across organizational boundaries, between microservices in a distributed deployment, or between autonomous agents in a multi-agent pipeline — can use primitives as self-authenticating message envelopes. The sender mints a primitive of type 0x10 whose h field binds to the message payload, signs it under its three-family key set, and transmits the 74-byte anchored footprint (the 32-byte signing message plus the 42-byte compact receipt) alongside or in place of the payload itself. The receiver verifies the primitive locally, using only the sender's public key set and the signature bundle, and accepts the message only if verification succeeds.

The construction eliminates the need for a centralized message broker to vouch for message authenticity. No trusted intermediary holds a signing key or a shared secret; H33-74's three-family signature is the authentication credential, and it travels with the message. In a publish-subscribe topology, the publisher attests each message at production time and subscribers verify independently, without trusting the relay infrastructure. A compromised relay can delay or drop messages but cannot forge them, because forgery requires producing valid signatures under all three post-quantum families simultaneously.

We note that H33-74 in this application functions as a network-layer primitive rather than a storage-layer primitive. The 74-byte footprint is small enough to fit in a single UDP datagram header extension, an MQTT payload prefix, or a gRPC metadata field. The receiver does not need to contact Bitcoin or any ledger to verify the message; verification is purely local and purely offline. When the application additionally requires non-repudiation or global ordering, the signing message can optionally be anchored to Bitcoin in the same manner described in Section 9.1, but the anchoring step is not required for the authenticated-relay use case.

The authenticated relay additionally functions as a network-layer routing primitive whose computation type byte and signing message can be parsed by intermediate network nodes to guide application-layer routing and reference storage decisions — directing the application layer to the appropriate off-chain content store without transmitting the underlying payload. This construction is distinct from unauthenticated blockchain-based coordination channels, which have been exploited as command-and-control infrastructure for botnet operations. Because every relay primitive requires a valid three-family post-quantum signature under a registered computation type, unauthenticated coordination instructions are rejected cryptographically at the network layer. Cross-domain injection — using a relay primitive to deliver instructions intended for a different computation domain — is prevented by the domain separation registry, which makes cross-type replay cryptographically impossible.

Abuse surface: authenticated command-and-control. We acknowledge a limitation that the preceding paragraph does not fully address. H33-74 prevents *unauthenticated* abuse of the relay channel, but it does not prevent *authenticated* abuse. A holder of a valid three-family signing key can attest arbitrary byte sequences, including sequences that encode executable instructions, configuration payloads, or coordination signals for malicious software. The three-family signature makes such payloads *attributable* and *non-repudiable* — the signer is cryptographically bound to every instruction they issue — but it does not prevent the issuance itself. More broadly, the primitive-referencing-primitive construction that we describe as a feature in §9.5 and §9.8 is, from a command-and-control perspective, a chaining mechanism whose ergonomics could be exploited by a sophisticated adversary with access to a signing key. We consider the design of friction mechanisms that make reflexive primitive chaining operationally costly without degrading legitimate relay throughput to be an open design question, and we flag it here so that reviewers evaluating H33-74 for deployment in adversarial environments can weigh this surface explicitly.

9.8. Identity Credential Attestation (t = 0x11)

A verified identity — the output of a KYC check, a biometric enrollment, an organizational role assignment, a device registration, or any other identity-establishing event — can be bound to a primitive of type 0x11 whose h field commits to the canonical encoding of the identity claim. The resulting primitive is a signed, verifiable, post-quantum-secure credential: a 74-byte object that proves that a specific identity claim was attested at a specific time by the holder of a specific three-family key set, under three independent hardness assumptions.

The credential is self-contained. A verifier who possesses the primitive, the compact receipt, and the signer's public key set can confirm the credential's integrity without contacting the issuer, without accessing a centralized identity directory, and without any network connectivity. This property is particularly valuable in environments where connectivity is intermittent (field operations, disaster response, air-gapped facilities) or where the verifier does not trust the network path to the issuer.

The primitive-referencing-primitive construction introduced in Section 9.5 enables credential composition: a document attestation primitive (type 0x0F) can reference an identity credential primitive (type 0x11) by including the identity primitive's signing message in its computation result. The composed construction binds who presented a document to what was presented and when, under the same three-family post-quantum security as the individual attestations. We have implemented this composition in our legal evidence application and in our KYC verification flow, and we find it to be a natural and useful pattern.

We emphasize a limitation. H33-74 attests that an identity claim was made and signed; it does not independently verify the truth of the underlying claim. A fraudulent identity provider who controls a signing key can issue primitives for fabricated identities. The primitive's value is in binding the claim to a specific signer, a specific time, and a specific three-family key set — creating a cryptographic audit trail that holds the issuer accountable for every credential it produces. The accountability is post-quantum-durable: even after a quantum computer breaks classical signature schemes, the three-family primitive binding remains intact.

9.9. Post-Quantum Proof of Life (t = 0x12)

Motivation. Approximately 1.8 million BTC reside in Pay-to-Public-Key (P2PK) outputs whose secp256k1 public keys are exposed on the blockchain. A sufficiently powerful quantum adversary running Shor's algorithm could derive the corresponding private keys and spend those outputs. The holder of such an output may wish to prove, cryptographically, that they still control the private key — and to bind that proof of control to a post-quantum key set — without revealing the private key itself and without moving the coins.

Construction. The holder generates a ZK-STARK proof of discrete logarithm knowledge: "I know a scalar sk such that pk = sk · G on secp256k1," where G is the standard generator and pk is the exposed public key. The STARK proof is generated over a Goldilocks field ($p = 2^{64} - 2^{32} + 1$) using non-native field arithmetic, representing each 256-bit secp256k1 base field element as eight 32-bit limbs. The 32-bit limb width ensures that every limb product fits exactly within the Goldilocks field without wraparound, yielding unconditional arithmetic soundness — the algebraic constraints faithfully represent the intended secp256k1 arithmetic without requiring auxiliary range proofs on intermediate carry values.

The execution trace consists of 256 rows (one per bit of the scalar), each implementing a projective-coordinate point doubling and conditional point addition. The Algebraic Intermediate Representation (AIR) enforces 836 constraints per transition row: 448 for projective point doubling (11 non-native field multiplications), 336 for projective point addition (8 non-native field multiplications), 48 for infinity-aware and scalar-bit multiplexing, and 4 for control flow (binary enforcement and step counter). The total constraint surface is approximately 213,000 Goldilocks field equations across 255 transitions, plus boundary constraints at the first and last rows that bind the accumulator's initial state to the group identity and its final state to the claimed public key.

The SHA3-256 hash of the STARK proof's polynomial commitments — specifically, the Merkle roots of the trace commitment and the constraint commitment produced during the FRI protocol — serves as the proof commitment. This commitment is bound to an H33-74 primitive of type 0x12

(PostQuantumMigration) via a domain-tagged binding digest: SHA3-256(`h33:proof-of-life:v1:` || proof_commitment || pk_compressed || pq_pk_hash || nonce). The binding digest becomes the content field of the H33-74 primitive, which is then signed under all three post-quantum families.

What it proves. A verified proof-of-life primitive establishes: (1) the holder knows the private key corresponding to a specific secp256k1 public key, without revealing the private key; (2) that knowledge is cryptographically bound to a specific three-family post-quantum key set; and (3) the binding is attested via H33-74 with a timestamp and, optionally, a Bitcoin anchor. This is the first application in this paper that addresses the limitation identified in §1.2 — that H33-74 attests results but does not verify computation correctness. The STARK proof verifies the computation (discrete logarithm) and the primitive attests the verification.

Relationship to Script-Layer Constructions. Proof of life is complementary to, and independent of, script-layer post-quantum spending constructions such as QSB. QSB protects a UTXO against quantum-assisted forgery of a spending transaction. Proof of life proves that the holder of a classical key still controls it and has migrated that control to a post-quantum key set. The two constructions serve different purposes and compose freely: a holder might prove life (establishing post-quantum key binding) and subsequently spend via a QSB-locked output (protecting the spend itself). Neither construction requires the other.

Limitations. The STARK proof establishes knowledge of the private key. It does not establish that the prover is the "rightful" owner of the corresponding Bitcoin — classical Bitcoin has the same limitation, and proof of life inherits it. The proof is also not transferable: the binding to the prover's PQ key set means that a third party cannot replay the proof to claim control under a different PQ identity.

Performance. Cold proof generation completes in 495 ms on Apple Silicon (release build) and 813 ms on AWS Graviton4 (c8g.16xlarge, release build). The proof consists of two 32-byte Merkle roots (trace commitment and constraint commitment) plus FRI query responses; the proof commitment bound to the H33-74 primitive is a single 32-byte SHA3-256 hash. Verification of the H33-74 attestation (binding consistency plus three-family signature check) completes in approximately 800 µs, dominated by SLH-DSA-SHA2-128f signature verification. With Cachee-backed proof caching, repeat attestations for the same (pk, pq_key_set) binding skip the STARK pipeline entirely and complete in approximately 14.5 ms (34× speedup), dominated by the three-family PQ signing cost (SLH-DSA-SHA2-128f signing is the bottleneck at approximately 12 ms per invocation). Cached verification completes in sub-microsecond time. The STARK engine is a custom Goldilocks-field implementation with SHA3-256 Merkle commitments and a FRI-based polynomial commitment scheme; it does not use Winterfell, Plonky2, or any third-party proving framework.

Circuit Audit. The AIR constraints were validated against `k256` (RustCrypto's secp256k1 implementation) as a trusted oracle. Differential testing computed pk = sk · G independently in both the native implementation and k256 for 1,100 keys: 4 named test vectors (sk = 1, 2, 42, 0xDEADBEEF), 32 powers of 2, 5 edge cases (sk = n−1, byte-boundary values 255/256/257, the generator point, and sk > n), 100 pseudo-random keys, and 1,000 pseudo-random keys. All 1,100 key pairs produced identical (x, y) coordinates in both implementations. The AIR transition constraints (836 per row × 255 transitions) were verified to evaluate to zero for all 1,100 keys, plus 50 additional random keys tested at the constraint level. Boundary constraints at rows 0 and 255 were verified independently. Tamper detection was confirmed: modifying a single field element in the accumulator, flipping a single scalar bit, or corrupting the step counter each produced constraint violations detected by the verifier.

Bug Disclosure. Differential testing against k256 revealed a modular multiplication error in the native secp256k1 field arithmetic. The original implementation used a bit-by-bit shift-and-subtract reduction of the 512-bit product, processing 64 individual doublings per limb (512 total conditional subtractions per multiplication). This algorithm produced correct results for sk = 1 (which requires no point doubling beyond the generator lookup) but diverged from k256 for sk ≥ 2. The root cause was accumulated rounding in the iterative reduction path. The fix replaced the hand-rolled reduction with `num-bigint` (a widely audited arbitrary-precision integer library already present in the dependency tree). After the fix, all 1,100 differential test keys matched k256 exactly, and all AIR constraint checks passed with zero violations. We disclose this bug because it illustrates a class of error that is invisible to internal consistency testing — the AIR constraints were satisfied before and after the fix, because the constraints faithfully encoded the (incorrect) native arithmetic. Only differential testing against an independent trusted oracle detected the divergence. We consider this finding a strong argument for mandatory oracle-based differential testing of any ZK circuit that encodes non-native field arithmetic.

Implementation Status. The construction described above is built, tested, and deployed. A reference verifier for H33-74 attestations, including proof-of-life verification, is open-sourced at github.com/H33ai-postquantum/H33-74-Verifier. The proving engine and circuit constraints are proprietary to H33.ai, Inc.

9.10. What These Applications Have In Common

We list the nine applications above because they are what we have implemented. The common thread is that each application requires the same four properties, and H33-74 satisfies all four simultaneously:

1. **Fixed interface.** The primitive accepts an arbitrary computation type and an arbitrary computation result. It is application-agnostic: the same Mint/Sign/Verify interface attests a biometric match, a legal document, a financial settlement, an AI inference, or a proof of Bitcoin key control. No per-application customization is required.
2. **Fixed size.** The persistent footprint is 74 bytes (32 on-chain + 42 off-chain), regardless of whether the computation result is 1 byte or 10 megabytes, and regardless of which signature families are active. Size invariance is the property that makes H33-74 viable in storage-expensive environments (Bitcoin outputs, smart contract storage slots, embedded device memory).
3. **Composable.** Primitives can reference other primitives (via the primitive-referencing-primitive construction of §9.5), and batches of primitives can be aggregated under a single Merkle root (§5) with O(1/N) amortized cost. These two composition mechanisms — chaining and batching — are sufficient to encode arbitrary DAGs of attested computation events.
4. **Domain-separated.** The computation type byte t provides protocol-level separation between attestation domains. Cross-type replay is cryptographically impossible: an attestation minted as type 0x0D (AI Inference) cannot verify as type 0x06 (Bitcoin UTXO) because the type byte is included in the signing message hash. This is a registry-level property, not an application-level convention.

We do not claim novelty for any individual building block. SHA3 is not novel. Three-family signing bundles are not novel. Merkle aggregation is not novel. The claim is that no prior construction achieves all four properties simultaneously under post-quantum assumptions with a fixed-width output. The closest prior art — C2PA (variable-width, single-family), DSSE (no batching, no domain separation), Sigstore (not post-quantum, no fixed-width commitment) — satisfies subsets of these properties but not the conjunction. We are prepared to be corrected on this point and we welcome citations that predate our work.

The proof-of-life application (§9.9) is distinctive in that it composes the primitive's attestation with a ZK-STARK proof of computation correctness — the first instance in this paper where H33-74 attests not merely a result but the correctness of the computation that produced it.

10. Related Work

10.1. Script-Layer Post-Quantum Bitcoin

The closest related work is the recent line of script-layer post-quantum Bitcoin constructions: Heilman's Lamport-in-script demonstration [@heilman2024], Linus's BINOHASH [@binohash2026], and Levy's QSB [@qsb2026]. These constructions attack a specific, well-defined threat: an adversary with Shor's algorithm who attempts to forge or rewrite a spending transaction. They achieve quantum-safe spending within existing Bitcoin consensus rules, without a soft fork. They are excellent works of cryptographic engineering and we consider them complementary to the present primitive rather than alternatives.

The relationship is the following. QSB and BINOHASH operate inside Bitcoin script, at the layer where the Bitcoin full-node verifier runs. They protect the UTXO's spendability. They do not — and should not — attempt to attest the off-chain computation that produced the UTXO, because Bitcoin script is bounded to 201 non-push opcodes and 10,000 bytes for the explicit purpose of keeping full-node verification work bounded. H33-74 operates at the application layer, outside Bitcoin script. It does not — and should not — attempt to solve the script-layer forgery problem, because that problem is better solved by constructions like QSB and BINOHASH that live at the script layer and can take advantage of Bitcoin-specific primitives like FindAndDelete.

PROPERTY	QSB / BINOHASH	H33-74
Kind of object	Protocol construction	Primitive
Scope	Single-UTXO Bitcoin script	General-purpose attestation
Lives at	Bitcoin script layer	Application layer
Protects	UTXO spendability	Computation-result attestation
Hardness bet	One assumption (hash pre-image)	Three independent assumptions
Chain dependence	Bitcoin legacy script	None (chain-agnostic)
Proves	Transaction integrity under quantum adversary	Computation-result attestation
Softfork required	No	No

The table is not a ranking. Each column describes a different kind of object. A production deployment that cares about both script-layer quantum safety for a specific UTXO and application-layer quantum safety for the computation that produced the UTXO would deploy both constructions, and we recommend that deployment pattern for institutions that require both.

We note one small operational observation. A QSB transaction broadcast must be conducted via a non-standard mempool service (the paper references Marathon Slipstream) because QSB transactions exceed the default Bitcoin Core relay policy. A primitive attestation anchored via Taproot key-path tweak is indistinguishable from any other Taproot spend and adds zero marginal transaction weight; alternatively, an OP_RETURN anchor fits within standard transaction format. Both relay through any standard Bitcoin mempool. This is a deployment convenience, not a security property.

10.2. Single-Family Attestation Schemes

RFC 9421: HTTP Message Signatures [@rfc9421] defines a wire format for signing HTTP message components with a single signature algorithm. A post-quantum deployment of RFC 9421 would pick one post-quantum signature family and sign each response independently. H33-74 differs in three ways: it provides three-family rather than single-family protection; it has a fixed 74-byte persistent footprint versus RFC 9421's raw signature payload of several kilobytes; and it admits batched Merkle aggregation for amortized cost, which RFC 9421 does not specify.

Certificate Transparency [@ct2013] publishes all issued X.509 certificates to a globally-shared append-only log with Merkle inclusion proofs. Certificate Transparency's Merkle construction directly influences H33-74's batched aggregation construction in Section 5, and we credit the original CT paper accordingly. H33-74 differs from Certificate Transparency in that it uses three-family post-quantum signing rather than single-family classical signing, in that it does not require a single globally-shared log, and in that it supports arbitrary computation types via the append-only computation type registry (T) rather than being specialized to X.509 certificates.

Sigstore [@sigstore2022] is a transparency-log-backed signing service for software artifacts using ephemeral keys and OIDC identity. H33-74 differs in its three-family post-quantum signing, its absence of dependence on an OIDC identity provider or a central log, and its generalization to arbitrary computation types.

C2PA [@c2pa2023] is a media-authentication format for associating cryptographic metadata with digital media under classical single-family signatures. The primitive's media-capture application (Section 9.4) addresses the same problem space with three-family post-quantum signatures computed in firmware before any post-processing of the captured data.

10.3. Content Addressing

IPFS [@ipfs2014] addresses data items by cryptographic hash, providing a content identifier that binds a name to the canonical bytes of the referenced data. The primitive's signing message field functions analogously as a content identifier, but additionally binds the content to a computation type, a timestamp, and a three-family signature. One may view H33-74's signing message as a content identifier with computational provenance.

11. Open Problems

We list several open questions concerning H33-74, in the hope of encouraging discussion and future work.

11.1. Independence of the Three Hardness Assumptions

The three-family security argument of Section 6 relies on the informal claim that MLWE, SIS over NTRU, and hash pre-image resistance are cryptanalytically independent. This claim is empirically supported by the current state of the NIST PQC program — each of the three families is believed to stand or fall on its own — but it is not a theorem. A formal separation result would strengthen the security argument considerably.

We offer the following structural rationale for believing the independence claim is reasonable, without claiming it constitutes a proof. MLWE and NTRU-SIS are both lattice problems, but they operate over different algebraic objects with different geometric structure. MLWE hardness relies on the difficulty of distinguishing noisy inner products over module lattices; the best known attacks reduce to the Block Korkin-Zolotarev (BKZ) lattice basis reduction algorithm with complexity exponential in the lattice dimension. NTRU-SIS hardness relies on the difficulty of finding short vectors in lattices defined by the NTRU key structure $f \cdot g^{-1} \bmod q$; the best known attacks combine lattice reduction with NTRU-specific combinatorial methods that exploit the ring structure in ways that do not apply to generic MLWE modules. The two problems share a common ancestor (worst-case lattice hardness) but the practical attack algorithms diverge. Importantly, no known reduction shows that breaking MLWE at useful parameters implies breaking NTRU-SIS at useful parameters, or vice versa. If such a reduction existed at practical parameters, it would be one of the most significant results in lattice cryptanalysis.

SLH-DSA is categorically outside the lattice family. Its security reduces to the pre-image resistance of SHA2-256. A breakthrough in lattice algorithms — even one that simultaneously breaks both MLWE and NTRU-SIS — would have no bearing on SHA2-256 pre-image resistance, because hash

functions have no algebraic structure for lattice techniques to exploit. This structural separation between the two lattice-based families and the hash-based family is the strongest element of the independence argument.

Conversely, if a future cryptanalytic result reduces the effective independence of the first two families (for example, by exhibiting a reduction between MLWE and NTRU-SIS with useful parameters), then H33-74's security margin would narrow from three-family to approximately two-family (lattice + hash), and we would want to rotate one of the lattice families to a non-lattice replacement via the algorithm_flags mechanism of Definition 6. We consider formalizing this independence claim to be the most important open theoretical question concerning H33-74.

11.2. Post-Quantum Hash Function Choice

H33-74 uses SHA3-256 (Keccak sponge construction, NIST FIPS 202) for the content digest, the signing message, and the internal Merkle tree. We acknowledge that SHA3-256 is less widely deployed and less battle-tested in the Bitcoin ecosystem than SHA2-256 (Merkle-Damgård construction), which Bitcoin uses natively for block hashing, transaction identifiers, and address derivation. SHA3-256 was chosen for H33-74 because its sponge construction provides a structurally different security argument from SHA2-256, offering defense-in-depth if a future weakness is found in Merkle-Damgård constructions. This choice is a design decision, not a claim that SHA2-256 is insufficient.

We note that a content-addressable storage layer using SHA2-256 addresses — parallel to the SHA3-256 signing message — would provide native interoperability with Bitcoin full nodes and IPFS without requiring an adapter layer. This dual-hash approach (SHA3-256 for security properties, SHA2-256 for ecosystem addressing) is under consideration for a future version.

Grover's algorithm reduces the effective pre-image resistance of a 256-bit hash from 2^{256} to 2^{128} and the effective collision resistance from 2^{128} to $2^{85.3}$. Both figures remain well above any practical threshold, but an adversary with extraordinary quantum resources would shift the security margin. An upgrade to SHA3-384 or SHA3-512 is plausible, at the cost of increased primitive width. We have not committed to this upgrade and consider it an open design question.

We note that larger hash functions are a per-deployment option for chains that natively use larger digests. The upgrade path is not abstract — a deployment targeting a chain whose native hash is SHA3-384 or SHA2-512 can select the corresponding hash width at mint time, and the primitive's wire format accommodates the change by widening the content digest field. We observe that some chains claim to use larger hashes but in practice rely on secp256k1 ECDSA for transaction signing

(Hedera, for example, advertises SHA-384 but signs with secp256k1), so the effective post-quantum security of the chain's native hash is not always what it appears. Deployments should verify the actual hash used in the chain's consensus-critical path, not the marketing claim.

11.3. Formal Verification of Batched Aggregation

The Merkle aggregation construction of Section 5 is straightforward but relies on correct domain separation between leaf and internal node hashing (leaves are prefixed with 0x00, internal nodes with 0x01). A formal proof of second-preimage resistance of the aggregation construction under a random-oracle model for SHA3-256 would be a useful addition and would improve our confidence in the construction at very large N.

11.4. Standardization

The primitive's wire format is a reference implementation, not a formal standard. A formal standardization effort through an appropriate body (we suspect IETF for the HTTP wire protocol aspects, and potentially an ISO or IEEE track for the computation type registry) is a direction we would welcome and for which we invite collaboration.

11.5. Integration with BitVM

BitVM [@bitvm2023] defines a framework for off-chain Bitcoin program execution with on-chain dispute resolution. The primitive's arbitrary-computation attestation would fit naturally as an attestation layer for BitVM execution traces, providing three-family post-quantum protection of each computation step. Integration work has not yet begun and is a direction we consider worth pursuing jointly with the BitVM authors.

11.6. Threshold-of-Two Verification During Family Rotation

The current construction requires all three signature families to verify (the algorithm_flags byte must indicate all three are present, and the Verify algorithm checks all three in steps 6-8). If one family suffers a surprise cryptanalytic weakness, the secure response is to rotate the weakened family out and replace it with a stronger alternative via the algorithm_flags mechanism.

During the rotation window, however, existing primitives signed under the old three-family bundle cannot be re-signed. A verifier encountering a pre-rotation primitive faces a choice: reject it (because one of the three families is now considered weak) or accept it under a two-of-three threshold (because the remaining two families are intact). The algorithm_flags field supports partial

verification mechanically, but the security implications of accepting two-of-three verification as a backwards-compatibility measure during a rotation event have not been analyzed.

We consider this the most deployment-relevant open problem. A formal treatment would specify: (a) the conditions under which two-of-three acceptance is safe, (b) the maximum duration of a rotation window during which the relaxed threshold applies, and (c) whether the compact receipt's commitment digest c should be recomputed over only the surviving signatures or left unchanged. We invite the community's input on this question, as the answer has direct implications for the operational procedures of any large-scale primitive deployment. We recommend that production deployments treat rotation semantics as a pre-launch requirement to specify internally, even if the formal cryptographic analysis is deferred. Operating without a specified rotation policy means that a surprise family compromise places every existing primitive in an ambiguous verification state — an operational risk that should be resolved before, not after, the event.

11.7. Identity-Bound Document Attestation

The legal evidence chain of custody application (Section 9.5) attests document integrity but not document authenticity. A forged document attested by a valid signing key produces a valid primitive. We identify the composition of a document attestation primitive with a verified identity credential primitive — via the primitive-referencing-primitive construction — as a direction worth formalizing. The resulting construction binds the submitter's verified identity to the document attestation event, creating a cryptographically verifiable record of who presented the document, when, and what they presented. This composition has direct application to title insurance, land registries, court filings, and regulated document intake systems. The formal security properties of the composed construction — particularly the conditions under which the identity binding is unforgeable independently of the document binding — are an open question.

11.8. Crypto-Agile Algorithm Slot Architecture

The algorithm_flags byte in the compact receipt (Definition 6) is an 8-bit bitmask that encodes which signature families are present in the attestation. The reference deployment uses 0x07 (binary 00000111), meaning slots 1, 2, and 3 are active — ML-DSA-65, FALCON-512, and SLH-DSA-SHA2-128f respectively. Slots 4 through 8 are reserved and unused.

This bitmask architecture is directly aligned with the NIST Crypto Agility framework, which mandates that cryptographic systems be capable of disabling compromised algorithms and enabling replacement algorithms without requiring software updates to deployed infrastructure. The H33-74 flags byte satisfies this requirement by construction: any valid subset of the 8 registered

algorithm slots can be activated or deactivated at mint time by changing the bitmask value. A verifier that encounters an unfamiliar bitmask value can determine which algorithms are active by inspecting the individual bits and verify only the families it recognizes, providing both forward and backward compatibility.

The operational implications are significant. If a cryptanalytic advance compromises one of the three active families — for example, a practical attack on NTRU-SIS that breaks FALCON-512 — the remediation is: (1) register a replacement algorithm in an unused slot (e.g., slot 4), (2) update the default bitmask from 0x07 to 0x1D (binary 00011101: slots 1, 3, 4, 5 active), and (3) the next mint produces a valid attestation under the new family composition. No binary recompilation. No software distribution. No flag-day cutover. The bitmask change propagates through the existing license fingerprint mechanism (§8.3) at the next billing-period reconciliation.

We consider formalizing the algorithm slot registry — including the registration criteria for new slots, the governance process for activating and deactivating families, and the verification semantics when a verifier encounters an attestation whose bitmask includes families it does not support — to be an important direction for future work. The structural foundation is already present in the wire format; the policy and governance layer is what remains to be specified.

11.9. Zero-Knowledge Proof of Computation Integrity

Section 1.2 identifies a fundamental limitation of H33-74: the primitive attests that a result was signed, not that it was correctly computed. The proof-of-life construction in §9.9 closes this gap for one specific computation — secp256k1 discrete logarithm knowledge — by composing a ZK-STARK proof of computation correctness with an H33-74 attestation of the proof's validity.

The natural generalization is to extend this pattern to other computation types. An AI inference attestation (type 0x0D) currently proves that the named model produced a signed output; it does not prove that the inference was performed correctly — that is, that the output is the result of applying the model weights to the input. A ZK-STARK proof of correct inference, whose commitment is bound to the H33-74 primitive, would close this gap. Similarly, a fraud-score computation (type 0x02) could be accompanied by a STARK proof that the score was computed according to a published scoring function, without revealing the input data.

The challenge is circuit complexity. The secp256k1 proof-of-life circuit requires approximately 213,000 Goldilocks field constraints for 256 scalar multiplication steps — a non-trivial but tractable construction. A proof of correct neural-network inference for even a modest model would require orders of magnitude more constraints, likely necessitating recursive proof composition or a different

proof system architecture. Whether ZK-STARK proofs of computation correctness can be made practical for the full range of computation types that H33-74 supports is an open question with significant implications for the primitive's long-term value proposition. We consider this the most technically ambitious open problem on this list.

12. Conclusion

We have described H33-74, a 74-byte post-quantum attestation primitive that binds arbitrary computation results to a three-family signature bundle (ML-DSA-65, FALCON-512, SLH-DSA-SHA2-128f) at a persistent footprint independent of computation data volume. H33-74 is chain-agnostic and transport-agnostic; we have described applications to post-quantum Bitcoin UTXO attestation, HTTP API response attestation, AI inference provenance, capture-time media authenticity, legal evidence chain-of-custody, federated machine-learning mesh, authenticated relay messaging, identity credential attestation, and post-quantum proof of life — the last of which composes a ZK-STARK proof of secp256k1 discrete logarithm knowledge with the primitive's three-family attestation, binding classical key control to a post-quantum identity without revealing the private key.

H33-74 is intended to complement, not replace, the excellent recent work on script-layer post-quantum Bitcoin constructions by Heilman, Linus, and Levy. Their constructions protect Bitcoin UTXOs against script-layer signature forgery. H33-74 attests the computations that produced those UTXOs. The two kinds of object are complementary, and a production deployment that cares about both should deploy both.

We offer H33-74 to the Bitcoin developer community and the wider post-quantum cryptography community for review, criticism, and improvement. We are particularly interested in feedback on the three-family independence argument of Section 6, on the practical deployment patterns discussed in Section 8, and on the open problems listed in Section 11. Corrections to any of the related-work characterizations in Section 10 are especially welcome and should be communicated to the contact address below.

We close with an observation about the composition of H33-74 with Bitcoin's existing infrastructure. H33-74's computation-agnostic design, when anchored through Bitcoin's existing Taproot key-path tweak or OP_RETURN mechanism, suggests an architectural property that we believe deserves explicit attention: Bitcoin's unmodified transaction infrastructure may be sufficient to serve as a post-quantum attestation anchor for arbitrary computation across a wide range of domains. H33-74

does not upgrade Bitcoin. It does not require a single line of Bitcoin consensus code to change. It composes with an existing capability — fixed-width data embedding in Taproot outputs — to produce what appears to be a general-purpose attestation anchor whose trust properties are inherited from the Bitcoin ledger's own consensus security. To our knowledge, H33-74 is the first primitive to exercise this composition systematically, though we do not claim it is the only way to do so.

The implications extend beyond the nine applications enumerated in Section 9. The same 74-byte primitive that attests a biometric match or a legal evidence chain can attest a tokenized real-world asset issuance, serve as a cryptographic checkpoint in a multi-stage FHE pipeline, or anchor autonomous AI agent delegation chains. Each reduces to the same object: a 58-byte primitive whose 32-byte hash is committed to Bitcoin via a standard Taproot tweak. No new opcodes. No consensus changes. No soft fork.

Post-quantum Bitcoin is a hard problem and none of us is going to solve it alone. We are grateful for the work that came before this paper and we look forward to the work that will come after it.

13. Acknowledgements

The author is grateful to Ethan Heilman for the original Lamport-in-script demonstration that opened this design space; to Robin Linus for BINOHASH, which established that a practical in-script post-quantum construction was achievable under existing consensus rules; and to Avihu Mordechai Levy for QSB, which closed a sharp vulnerability in BINOHASH's proof-of-work puzzle and set the current bar for script-layer post-quantum Bitcoin. This paper would not have the form it does without the prior work of those three authors, and any technical errors or misstatements about their constructions in Section 10 are the present author's alone.

The author also thanks the NIST Post-Quantum Cryptography standardization team for providing the three signature families on which this primitive rests, and the SHA-3 standardization team for providing the hash function on which everything else is built.

The author thanks Luke Dashjr for his public engagement on anchoring challenges, which contributed to the analysis that led to the Taproot key-path tweak as the preferred zero-weight anchoring method.

The author extends special thanks to Ian Smith, co-author of BIP-361, CEO of QuantumEVM (@IanSmith_HSA), whose rigorous technical review during the editing process saved the author

from several errors that would have distracted from the substance of this paper. His detailed technical feedback on SHA2/SHA3 disambiguation, the FALCON floating-point sampler concern and FALCON+ integer variant, the distinction between NIST security level and forgery resistance, Y2038 timestamp compliance, SHA2-256 storage addressing for Bitcoin/IPFS interoperability, and the crypto-agile algorithm slot architecture (§11.8) which resolved the family-composition question of Section 6.2 by identifying the algorithm_flags bitmask as a NIST-aligned crypto-agility mechanism — several corrections and one new open problem in this version of the paper are directly attributable to his review.

Appendix A. Benchmark Methodology

All performance figures in Section 7 are drawn from a single benchmark session conducted on 2026-04-13. This appendix documents the conditions under which those figures were produced.

Hardware. AWS c8g.metal-48xl bare-metal instance (192 vCPUs, 371 GiB RAM, Graviton4 ARM Neoverse V2), us-east-1. On-demand pricing at approximately $2.30/hr. The Nitro-virtualized comparison was measured on c8g.48xlarge (same physical hardware, Nitro hypervisor active).

Software. Proprietary Rust implementation compiled in release mode for aarch64. Implementation details are trade secrets of H33.ai, Inc.

Concurrency. 96 workers on the 192-vCPU metal instance, each running the full pipeline in a tight loop. Batches of 32 users per signing event.

Measurement. 30-second sustained measurement window after a 5-second warmup. Throughput computed as total completed authentications divided by elapsed wall-clock time. The 30-second window was chosen to capture steady-state behavior including cache warming.

Reproducibility. Raw benchmark logs, including per-second throughput samples and batch latency histograms, are retained at the author's site and available for independent review upon request.

Appendix B. Pricing and Licensing (NON-NORMATIVE)

This section describes the reference commercial implementation's pricing model and is non-normative. The cryptographic construction is independent of the economic model.

H33-74 is pricing-agnostic at the protocol level. The reference commercial implementation offers a flat $0.025 per-mint launch rate for all computation types, with global aggregate volume pricing that reduces the rate for all participants as adoption grows. A free developer tier (10,000 mints per month, no credit card) is available for prototyping and integration. Available license tiers, current pricing, and commercial terms are published at h33.ai/pricing.

The flat per-mint rate is chosen for simplicity and to remove any incentive to misclassify computation types. We acknowledge that the economic consequence of a mint varies significantly by application — a single primitive anchoring a Layer 2 batch settlement carries different economic weight than a single biometric attestation. Whether differentiated pricing by computation type or by downstream economic consequence is appropriate is an open question that the market will answer as adoption grows. The append-only computation type registry (§3, Definition 2) provides the technical foundation for differentiated pricing in future implementations without changes to the primitive construction itself.

14. References

[@shor1994] P. W. Shor. Algorithms for quantum computation: discrete logarithms and factoring. In Proceedings of the 35th Annual Symposium on Foundations of Computer Science, pages 124–134, 1994.

[@grover1996] L. K. Grover. A fast quantum mechanical algorithm for database search. In Proceedings of the 28th Annual ACM Symposium on Theory of Computing, pages 212–219, 1996.

[@merkle1979] R. C. Merkle. Secrecy, authentication, and public key systems. Ph.D. thesis, Stanford University, 1979.

[@lamport1979] L. Lamport. Constructing digital signatures from a one-way function. Technical Report SRI-CSL-98, SRI International, 1979.

[@bls2001] D. Boneh, B. Lynn, and H. Shacham. Short signatures from the Weil pairing. In Advances in Cryptology — ASIACRYPT 2001, pages 514–532, 2001.

[@kzg2010] A. Kate, G. M. Zaverucha, and I. Goldberg. Constant-size commitments to polynomials and their applications. In Advances in Cryptology — ASIACRYPT 2010, pages 177–194, 2010.

[@heilman2024] E. Heilman. Signing a Bitcoin transaction with Lamport signatures (no OP_CAT). Bitcoin Development Mailing List, 2024.

[@binohash2026] R. Linus. BINOHASH: Transaction introspection without soft forks. 2026.

[@qsb2026] A. M. Levy. Quantum-safe Bitcoin transactions without soft forks. StarkWare, April 2026.

[@fips203] National Institute of Standards and Technology. Module-lattice-based key-encapsulation mechanism standard. FIPS 203, 2024.

[@fips204] National Institute of Standards and Technology. Module-lattice-based digital signature standard. FIPS 204, 2024.

[@fips205] National Institute of Standards and Technology. Stateless hash-based digital signature standard. FIPS 205, 2024.

[@falcon2022] P.-A. Fouque et al. FALCON: Fast-Fourier lattice-based compact signatures over NTRU. NIST PQC Round 3 submission, 2022.

[@ct2013] B. Laurie, A. Langley, and E. Kasper. Certificate transparency. RFC 6962, 2013.

[@sigstore2022] Z. Newman, J. S. Meyers, and S. Torres-Arias. Sigstore: Software signing for everybody. In CCS, 2022.

[@c2pa2023] Coalition for Content Provenance and Authenticity. C2PA content credentials specification. 2023.

[@ipfs2014] J. Benet. IPFS — content addressed, versioned, peer-to-peer file system. arXiv:1407.3561, 2014.

[@bitvm2023] R. Linus. BitVM: Compute anything on Bitcoin. 2023.

[@rfc9421] A. Backman, J. Richer, and M. Sporny. HTTP message signatures. RFC 9421, 2024.

15. Contact

H33.ai, Inc.
Author: Eric Beans
Email: research@h33.ai

Correspondence welcome. Corrections to any technical statement in this paper will be incorporated in a revised version with attribution to the contributor at their option. Errata and pull requests:

github.com/H33ai-postquantum.

This document describes H33-74 specification version v = 1 (the version byte defined in Definition 3). Future specification revisions will increment v accordingly; the computation type registry and algorithm flags byte are designed for forward compatibility without version-byte changes.
