Accountable Distribution of Machine-Checked Correctness Evidence

A Transparency Model and the Lean Transparency Log

Olaf Horvath
Olaf.Horvath@zkdefi.org ORCID 0009-0004-8008-5805

July 2026
Revised: August 2026 — v0.10

Abstract

Formal verification produces machine-checkable evidence, but consuming that evidence usually requires the original prover, dependency graph, source checkout, and substantial replay time. This paper studies a distinct cryptographic problem: how can a lightweight consumer obtain precise and accountable assurance about a deterministic proof replay without executing the verifier and without reducing the result to an opaque provider label?

We define *accountable replay attestation*. A specialized operator performs an expensive replay once and publishes a structured observation through a signed append-only log. Consumers verify a signed tree head and logarithmic inclusion proof, pin history, and apply their own policy to the exact reported axiom-name sets for each theorem. The construction does not prove that the operator's observation is true. It makes the claim immutable within a signed view, comparable across consumers, and attributable when incompatible signed views are compared.

We instantiate the model as the Lean Transparency Log (LTL), using Lean 4 replay attestations and an RFC 9162 Merkle tree. We give explicit collision-extracting arguments for inclusion and consistency, lift them to scheme-level accountability games with a composition theorem, and evaluate a live deployment over four production Ed25519 codebases. The public log contains thirteen leaves; the thirteenth attests a Lean mechanization of the accumulator's own security arguments (61 human-reviewed certificates with one project-specific uninterpreted SHA-256 boundary axiom). The mechanization effort also exposed, via differential testing, a nontrivial implementation boundary — the deployed iterative consistency verifier is not extensionally equal to the recursive model on malformed size claims — and the leaf records this limitation explicitly. The contribution is a cryptographic distribution model for machine-checked correctness evidence, with an end-to-end deployed instantiation that carries scoped proofs about its own accountability machinery.

1 Introduction

Formal verification has made it possible to connect production cryptographic software to machine-checked mathematics. Systems such as HACL*, EverCrypt, Fiat-Crypto, and Aeneas demonstrate different routes from implementation to proof [13, 14, 15, 11]. Yet the standard assurance story quietly assumes a capable consumer: one that can retrieve the exact source, reconstruct the verifier environment, resolve dependencies, and spend minutes or hours replaying a corpus.

That assumption is often false. A package resolver selecting a cryptographic backend, a deployment controller enforcing a proof requirement, or an autonomous custody agent may have milliseconds and a small trusted computing base, not a theorem prover and thirty minutes of kernel time per candidate. This creates a problem that is logically downstream of proof construction:

How can a consumer that cannot execute the prover obtain precise, accountable evidence about a proof replay, without collapsing the result into an opaque provider verdict?

A detached signature on the word "verified" authenticates an issuer but does not bind an ordered history, expose silent replacement, or give consumers a compact state that can be pinned and required to grow. Shipping the complete proof and checker preserves direct verification but may defeat the cost and portability objective. Committees distribute trust but do not themselves fix the semantics of the attested result. Succinct proofs of verifier execution would provide validity rather than mere accountability, but require a circuit or verified-VM representation of the prover and are not yet the deployment assumption of the artifacts studied here.

We therefore study a narrower primitive: *accountable delegation of deterministic proof replay*. The operator still observes the replay. The cryptographic layer does not make that observation true. Instead, it makes the observation exact, persistent, attributable, and locally policy-checkable. Independent replay remains the mechanism for challenging a fabricated observation.

Central thesis. The contribution is not a new Merkle tree and not a new theorem prover. It is a trust decomposition for distributing machine-checked correctness evidence:

> **Expensive deterministic verification produces an observation. Transparency makes that observation accountable. Consumer-local policy decides whether the observation is acceptable.**

The Lean Transparency Log (LTL)[1] is the complete instantiation evaluated in this paper. Its subjects are four Rust Ed25519 codebases with Lean 4 [12] certificates against extracted models. Its thirteenth public leaf attests the Lean corpus that mechanizes the log's own accumulator arguments. Thus the paper's central claim survives replacement of Lean, Ed25519, or RFC 9162 by other components; what is essential is the distribution and accountability model.

Contributions.

1. **A distribution model for machine-checked evidence.** We define replay attestations, distinguish observation from verdict, and state what a lightweight consumer learns without executing Lean.
2. **A cryptographic accountability layer.** Using the RFC 9162 tree unchanged, we define signed views, inclusion receipts, local history pinning, and transferable same-size fork evidence. We give explicit collision-extracting soundness arguments specialized to the consumer algorithms, and lift them to scheme level: concrete games for position binding, history binding, and fork evidence, discharged by explicit reductions (§5.4).
3. **Boundary-conformance policy.** Each leaf records the exact axiom names reported by Lean. Consumers compare those observations with their own policy; operator labels can veto but cannot grant acceptance. We state clearly that axiom-name equality is not semantic identity of theorem statements.
4. **A deployed cryptographic case study.** The log contains twelve historical replay leaves for four verified Ed25519 codebases and a thirteenth leaf for the accumulator's own Lean corpus. The entry-13 corpus carries an environment-derived audit inventory of 222 compiled constants, 61 human-reviewed certificate cones, and a single uninterpreted SHA-256 axiom.

[1] The acronym collides with linear temporal logic [20]; we note the collision once and rely on context.

5. **A negative deployment result.** Differential testing found that the deployed iterative RFC-style consistency verifier and the recursive model proved in Lean are not extensionally equal: there are malformed size/root combinations accepted only by the deployed verifier. We characterize 3,867 divergences in 73,573 pinned boundary tests — every one deployed-accepts-only — and scope the public attestation accordingly. (Post-submission closure, July 2026: the divergence was traced to the deployed verifier omitting RFC 9162 §2.1.4.2 Step 7's terminal $sn = 0$ condition; restoring that one conjunct removes every divergence in the pinned family, confirmed by a three-way regression against an independent faithful RFC transliteration.)

Non-claims. LTL does not prove that the operator honestly reported a kernel run; independent replay remains the way to detect a fabricated observation. It does not prove binary correspondence, compiler correctness, extraction faithfulness, side-channel resistance, SHA-512 correctness, or execution provenance of the signing binary. The present leaf schema identifies theorem declarations by repository commit and name, not by a canonical digest of their elaborated Lean types. These are explicit boundaries, not hidden qualifications.

2 The distribution problem

2.1 Three evidence modes

Let a subject repository at commit g contain theorem declarations $T_1, \ldots, T_q$. A deterministic verifier execution produces an observation O_g containing success/failure and the reported assumption cone of each T_i — the set of axioms the checked proof of T_i ultimately rests on. There are three natural ways to consume this result.

Direct replay. The consumer reconstructs the verifier environment and checks O_g itself. This gives the strongest provenance, but has high operational cost.

Detached attestation. A provider signs O_g. This is cheap to consume, but provides no append-only history and no common value for clients to pin.

Transparent attestation. The provider signs a tree head committing O_g as one leaf among an ordered history. A consumer verifies inclusion and persists a head. Incompatible views become attributable when compared.

The third mode is useful precisely when replay is expensive but the result is stable and deterministic. It does not dominate direct replay: it replaces local computation with a narrower trust in the replay provider.

2.2 Why transparency rather than a signature database?

Suppose an operator signs every replay result independently. Authenticity of an individual record follows from signature verification, but four properties are absent:

1. no signed value commits to the ordered set of all records;
2. the signer commits to no complete ordered history, so omission or replacement is not detectable by a fresh consumer and carries no compact consistency proof;
3. two consumers cannot compare a single compact view identifier;
4. a consumer cannot demand that its previously accepted history only grow.

An append-only Merkle tree supplies these missing interfaces. At the current deployment size, logarithmic proof size is not the decisive benefit; *history binding* is.

Mechanism	Consumer cost	Semantic checker	History accountability	Main residual cost
Local replay	high	consumer	local only	prover/toolchain deployment
Proof transport / PCC	medium–high	consumer checker	optional	proof/checker portability
Detached signed result	low	provider	statement-level only	replaceable history
Committee replay	low	committee	none without an additional log	membership trust
Succinct proof of replay	low	circuit/VM verifier	optional	proving the prover
LTL	low	provider observes; consumer applies policy	signed append-only views	observation honesty

Table 1: Evidence-distribution alternatives. LTL targets low-cost consumers while retaining an attributable history; it does not remove trust in the replay observation.

2.3 Design alternatives

Table 1 places the construction among the natural alternatives, read as a design taxonomy rather than an empirical comparison. Local replay and proof transport keep semantic checking with the consumer at high operational cost; detached signatures and committees lower consumer cost but commit to no ordered history (a committee distributes trust in the observation; it does not by itself make the record's history accountable); succinct proofs of replay would upgrade accountability to validity at the price of proving the prover. LTL occupies the low-consumer-cost point that still binds an ordered, signed, pinnable history — and deliberately does not buy validity of the observation itself.

3 Model and trust decomposition

3.1 Roles and objects

The system has three logical roles.

Subject maintainer. Publishes source and proof artifacts at a commit.

Replay operator. Executes the declared verifier procedure, constructs an attestation, appends it to the log, and signs tree heads.

Consumer. Holds the log public key and a local policy; verifies receipts and optionally persists a previous head.

A replay attestation a contains at least

$$(\mathsf{repo}, g, \mathsf{toolchain}, \mathsf{environment}, [(N_i, s_i, A_i)]_{i=1}^{q}),$$

where N_i is a declaration name, s_i is replay status, and A_i is the observed axiom-name set. The deployed schema additionally carries diagnostics, resource controls, scope, and exclusions.

Definition 1 (Attestation-transparency scheme)**.** *An attestation-transparency scheme is a tuple*

$$\Pi = (\mathsf{KeyGen}, \mathsf{Append}, \mathsf{ProveIncl}, \mathsf{VerifyIncl}, \mathsf{ProveCons}, \mathsf{VerifyCons}, \mathsf{Verdict})$$

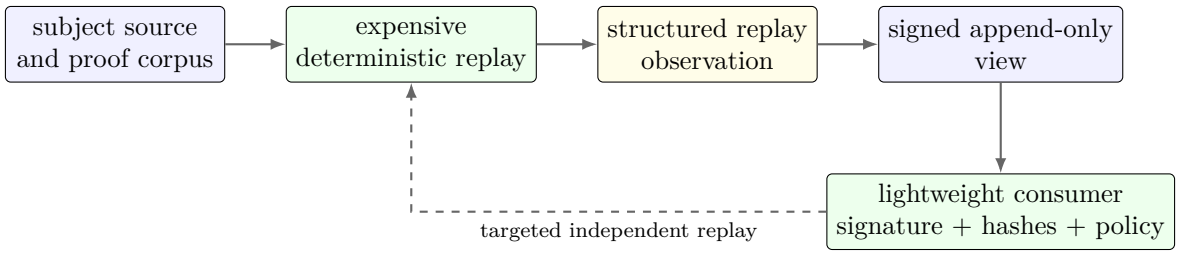

Figure 1: Trust decomposition. The log authenticates and orders the replay operator's observation; it does not replace the theorem prover or make the observation true.

over a hash function and signature scheme. Append *commits the canonical serialization of an attestation as the next leaf and returns a signed tree head.* Verdict *is parameterized by consumer-local policy and does not consume an operator verdict as positive evidence.*

Definition 2 (Accountable replay distribution)**.** *Fix an operator public key and a consumer that persists accepted signed heads. A replay-distribution scheme is* accountable *if the following hold: (i) every accepted attestation has an authentic signed view and a uniquely determined leaf value at its claimed position; (ii) a consumer accepts a later view only as the same view or a verified extension; (iii) two valid equal-size heads with unequal roots, in one log and protocol context, form transferable evidence that the key holder signed incompatible views; and (iv) positive acceptance of a theorem boundary is a function of recorded observations and consumer-local policy, not of an operator verdict.*

The definition is intentionally an accountability property, not a validity property. It says when conflicting claims become attributable; it does not cryptographically prove that the replay observation was honestly produced. Throughout, "accountability" means signed-view and history accountability; observation validity remains external to the mechanism. Section 5.4 states clauses (i)–(iv) as games and explicit reductions and proves the construction satisfies them.

3.2 What is and is not transferred

A verified receipt establishes a statement of the form:

> The holder of public key *pk* signed a tree head committing, at position m, to a leaf in which the operator reports that the named declarations at commit g replayed with the recorded axiom-name sets.

It does not establish that the operator's report is true. Nor does it identify the semantics of a theorem from its name alone. This separation is central:

Layer	What it contributes
Lean kernel	validity of a checked term relative to declarations and axioms
Replay pipeline	binding of source, toolchain, theorem names, and observations
Attestation signature	attribution of one replay statement
Merkle log	position binding and append-only view commitments
Consumer policy	acceptability of the recorded boundary
Independent replay	detection of fabricated operator observations

3.3 Adversary model

The network adversary may replay, delay, suppress, or substitute messages. The operator may be malicious: it may construct arbitrary leaves, sign arbitrary heads, label results arbitrarily,

and present different signed views to different consumers. We assume collision resistance of SHA-256 for the log, EUF-CMA security of the head-signature scheme, and correct initial acquisition of the operator public key.

The model deliberately does not cryptographically exclude fabricated kernel observations; the same holds when the operator's replay harness is defective rather than dishonest. That is a statement about a physical execution on the operator's machine. The mechanism instead makes the claimed execution target precise enough for a third party to replay.

3.4 Consumer goals

Each goal below is formalized as a game in Section 5.4.

G1: Authentic position binding. If a consumer accepts leaf d at index m under signed head (n, r): the head was issued by the key holder except under signature forgery, and no leaf distinct from d can also be opened at position m under that head except under hash collision.

G2: Local append-only history. A consumer that persists (n, r) accepts a later view only if it is the same view or a verified extension. Two valid heads of equal size and unequal roots, in one log context, are transferable evidence that the key holder signed incompatible views. Unequal-size forks require retained history, gossip, or a witness. The transition discipline itself is syntactic, enforced by the pin rule by construction; the semantic content — an opened position cannot change value across accepted views — is a theorem (§5.4).

G3: Policy separation. The operator's positive label cannot make a certificate acceptable. The consumer recomputes boundary conformance from observations and local policy. Operator failure labels may be treated as a conservative veto.

3.5 Boundary conformance, not semantic identity

For certificate c, let $\mathsf{Obs}_a(c)$ be the axiom-name set recorded in leaf a, and let $\mathsf{Policy}(c)$ be the consumer's allowed set. Define

$$\mathsf{boundary\_ok}_a(c) \iff \mathsf{Obs}_a(c) = \mathsf{Policy}(c).$$

Exact equality detects both additional assumptions and drift in the declared assurance interface. A missing expected axiom is not automatically a logical defect: it may indicate a strengthened theorem, a changed statement, a bypassed abstraction boundary, or stale policy. The consumer therefore rejects or requires review rather than interpreting the drift.

This predicate is intentionally narrower than "the intended theorem was proved." The deployed system identifies a declaration by repository commit and name. A stronger future schema should include canonical digests of the elaborated theorem type and of the types of declarations in its axiom cone.

4 Construction

4.1 RFC 9162 tree

Let H be SHA-256. For byte string d and 32-byte values x, y define

$$\mathsf{h}_{\mathrm{leaf}}(d) = \mathsf{H}(\texttt{0x00} \parallel d), \qquad \mathsf{h}_{\mathrm{node}}(x, y) = \mathsf{H}(\texttt{0x01} \parallel x \parallel y).$$

For leaf list $D = [d_0, \ldots, d_{n-1}]$:

$$\mathsf{MTH}([]) = \mathsf{H}(\epsilon), \qquad \mathsf{MTH}([d]) = \mathsf{h}_{\mathrm{leaf}}(d),$$

and for $n > 1$,

$$\mathsf{MTH}(D) = \mathsf{h}_{\mathrm{node}}(\mathsf{MTH}(D[0{:}k]), \mathsf{MTH}(D[k{:}n])),$$

where k is the largest power of two strictly smaller than n. Inclusion and consistency proofs are the RFC 9162 algorithms [2].

4.2 Signed tree heads

A tree head contains schema-version and type tags, a log identifier, tree size, root hash, timestamp, and hash-algorithm identifier. The canonical JSON serialization of those fields is signed with Ed25519. The log identifier and version tag prevent cross-log and cross-protocol replay.

The current implementation records signing-backend provenance alongside the signature, but that provenance is not execution attestation: an Ed25519 signature does not identify the program that produced it. The public system therefore treats the claimed signing implementation as operator-reported context, not as a property proved by the signature.

4.3 Receipts and pinning

A receipt contains the leaf index, sibling path, and signed head. A consumer first verifies the head signature and then reconstructs the root. For history, a local pin $(n_{\mathrm{pin}}, r_{\mathrm{pin}})$ evolves as follows:

- same size: accept iff roots match; otherwise retain both signed heads as same-size fork evidence;
- larger size: accept iff a consistency proof verifies, then update;
- smaller size: reject as rollback.

Freshness is an external availability policy. A persisted pin detects rollback relative to local history; it does not prove that a client sees the globally latest signed head.

Notation summary. For reference across the security analysis:

H; $\mathsf{h}_{\mathrm{leaf}}(d)$; $\mathsf{h}_{\mathrm{node}}(x, y)$	SHA-256; leaf hash $\mathsf{H}(\texttt{0x00} \Vert d)$; node hash $\mathsf{H}(\texttt{0x01} \Vert x \Vert y)$
D, d, m, n	leaf list; leaf bytes; leaf index; tree size
$\mathsf{MTH}(D)$; k	Merkle root; split point (largest power of two below n)
$\mathsf{Path}(m, D)$; $\mathsf{Root}(v, m, n, P)$	inclusion path (leaf to root); path refold
$\mathsf{Open}(d, m, n, P, r)$	accepting opening: $m < n$ and $\mathsf{Root}(\mathsf{h}_{\mathrm{leaf}}(d), m, n, P) = r$
$\mathsf{ConsRec}$; Ext	recursive consistency verifier; pin-rule transition
$\mathsf{Obs}_a(c)$; $\mathsf{Policy}(c)$	axiom names recorded in leaf a; consumer's allowed set
χ_{enc}; $\chi = (\chi_{\mathrm{enc}}, pk)$	payload-encoded head context; full context with the key
$h = (n, r, t; \sigma)$; Vf_{pk}	signed head (size, root, timestamp); signature check
Ev_{χ}	same-context, equal-size, unequal-root evidence pair
$\mathcal{B}_{\mathrm{pb}}, \mathcal{B}_{\mathrm{hist}}, \mathcal{B}_{\mathrm{ha}}, \mathcal{B}_{\mathrm{fr}}$	the named explicit reductions of §5.4
Q; **Adv**	signing-oracle query set; winning probability (keyed games)

5 Security analysis

This section states the consumer-facing arguments in the form used by the Lean mechanization. The proofs are elementary but explicit: successful false openings yield concrete SHA-256 collisions rather than appealing to an informal "Merkle trees are secure" statement. The explicitness is load-bearing: over a fixed-width hash a bare "some collision exists" is vacuously true by counting, so each soundness statement is about a named extractor function, and the corpus pins a machine-checked non-vacuity guard for every extractor.

5.1 Inclusion

Let $\mathsf{Root}(v, m, n, P)$ recursively fold value v at position m through proof path P using the same largest-power-of-two decomposition as MTH. Both Root and $\mathsf{ConsRec}$ are partial (the mechanization's `Option`): malformed shapes return a distinguished rejection value, and an equation such as $\mathsf{Root}(\cdot) = \mathsf{MTH}(D)$ asserts acceptance with that output.

Lemma 1 (Domain separation)**.** *For all byte strings d and 32-byte values x, y,* `0x00` $\| d \neq$ `0x01` $\| x \| y$.

Proof. The first byte differs. □

Theorem 1 (Inclusion completeness)**.** *For every non-empty D, every $m < |D|$,*

$$\mathsf{Root}(\mathsf{h}_{\mathrm{leaf}}(D[m]), m, |D|, \mathsf{Path}(m, D)) = \mathsf{MTH}(D).$$

Proof. By structural induction on $|D|$. The singleton case is immediate. For a split $D = L \| R$, the honest path is the recursive path inside the side containing m followed by the other side's root. The induction hypothesis reconstructs the selected child root, and the final node hash reconstructs $\mathsf{MTH}(D)$. □

Theorem 2 (Inclusion soundness: position binding)**.** *There is an explicit extractor $\mathcal{E}_{\mathrm{incl}}$ such that, given D, $m < |D|$, $d \neq D[m]$, and a path P satisfying*

$$\mathsf{Root}(\mathsf{h}_{\mathrm{leaf}}(d), m, |D|, P) = \mathsf{MTH}(D),$$

$\mathcal{E}_{\mathrm{incl}}(D, m, d, P)$ returns two distinct SHA-256 preimages with the same digest.

Proof. Recompute the honest tree and replay the accepting reconstruction from the root downward. At each visited internal node, compare the adversarial and honest node preimages. If they differ while their digests agree, output the collision. Otherwise both child values agree and descent continues along the path. At the terminal leaf, equality of digests with $d \neq D[m]$ yields a leaf-hash collision. Domain separation rules out interpreting a leaf preimage as a node preimage without already producing a collision. □

5.2 Consistency

The recursive verifier $\mathsf{ConsRec}(n_0, n_1, C, b, r_0)$ reconstructs an old and new root from proof C, a flag b indicating whether the old root is implicit, and pinned value r_0. Malformed shapes return rejection.

Theorem 3 (Consistency soundness of the recursive model)**.** *There is an explicit extractor $\mathcal{E}_{\mathrm{cons}}$ such that, whenever $|D_0| = n_0 \leq n_1 = |D_1|$, $D_0 \neq D_1[0{:}n_0]$, and*

$$\mathsf{ConsRec}(n_0, n_1, C, \top, \mathsf{MTH}(D_0)) = (\mathsf{MTH}(D_0), \mathsf{MTH}(D_1)),$$

$\mathcal{E}_{\mathrm{cons}}(D_0, D_1, C)$ returns a SHA-256 collision.

Proof. The new-root component is a hash fold over the shape of the n_1 tree. Compare it with the honest D_1 tree. Either the first differing preimage gives a collision, or every consumed proof node equals the corresponding honest node. In the latter case, the old-root component is the canonical fold of the honest prefix $D_1[0{:}n_0]$, so acceptance implies $\mathsf{MTH}(D_0) = \mathsf{MTH}(D_1[0{:}n_0])$. The two equal-sized leaf lists differ; descend through their identically shaped honest trees to extract the first hash collision. □

The theorem's hypothesis pins the honest old root $\mathsf{MTH}(D_0)$. The deployed flow has no mechanized supplier of that pin; the next subsection measures what the iterative verifier does when size claims alone steer its walk.

Proposition 1 (Pin-store safety)**.** *Assume EUF-CMA security of the head signature and collision resistance of SHA-256. A consumer following the pin transition accepts only a nondecreasing sequence of sizes whose exhibited leaf lists are prefix-related. Two accepted heads under the same key, in one log context, with equal size and unequal roots are transferable evidence that the key holder signed incompatible views.*

Proof. Rollback is rejected syntactically. At equal size the transition is accepted only with equal roots; if the two exhibited equal-length leaf lists differed, whole-tree binding would extract a SHA-256 collision, so under collision resistance the lists are equal. A larger head is accepted only after a consistency proof, so non-prefix acceptance yields a collision by the previous theorem. Equal-size unequal roots in one log context, with valid signatures, are two conflicting statements attributable to the key holder, except under signature forgery. □

Proposition 2 (Policy separation)**.** *For fixed local policy* Policy*, the boundary-conformance result for every certificate is a function only of* $\mathsf{Obs}_a(c)$ *and* $\mathsf{Policy}(c)$*. An operator label cannot change a nonconforming observation into a conforming one.*

Proof. The comparison is set equality and takes no positive operator verdict as input. A deployment may conservatively treat an operator failure label as a veto, but a veto cannot grant acceptance. □

5.3 Scope of the deployed consistency claim

The Lean theorem covers the recursive predicate above. The deployed iterative verifier follows the familiar RFC bit-navigation algorithm. Differential testing discovered that the two verifiers are not extensionally equal on malformed size claims, every observed divergence being accepted only by the iterative verifier: for example, a valid proof for a $2 \to 3$ transition can be accepted under the false old-size claim $1 \to 3$ when paired with the size-2 root. The mechanism is elementary: the iterative algorithm seeds its reconstruction with the supplied old root and consults the size claims only as bit-navigation state, so several distinct old-size claims navigate one proof identically. In 73,573 lied-size boundary cases, 3,867 divergences were observed; all were one-sided (deployed accepts, recursive model rejects). The root cause was later identified and closed: the deployed loop omitted RFC 9162 §2.1.4.2 Step 7's terminal $sn = 0$ condition; with the conjunct restored the pinned family shows zero divergences (three-way regression: deployed verifier, recursive model, independent RFC transliteration).

The intended consumer flow binds (n_0, r_0) in local persistent state and binds (n_1, r_1) together in a signed head. The present corpus does not prove a refinement theorem from that operational invariant to the recursive predicate. Consequently the public attestation says exactly this: the model is proved; deployment correspondence is finite-tested and relies on an unmechanized authentic-size/root invariant.

5.4 Scheme-level games and a composition theorem

The theorems above bind single artifacts to a reference leaf list. This subsection lifts them to the scheme. Readers content with the component theorems can skim the statements — the four games, Definition 3, Theorem 8 — and the closing mapping paragraph; the proofs add explicit reductions but no new assumptions.

Fix once, for the entire subsection, an *encoded context*

$$\chi_{\mathrm{enc}} = (\text{log identifier},\ \text{schema and type tags},\ \text{hash-algorithm identifier});$$

every head below is required to encode χ_{enc} in its canonical payload, matching the deployed head format of §4.2. The verification key is deliberately *not* part of the payload: it is the external verification parameter, and we write $\chi = (\chi_{\text{enc}}, pk)$ for the full context once a key exists — the keyed games fix χ_{enc}, run KeyGen, and then set χ.

The results come in two levels. The theorems are unconditional: explicit algorithms turn any winning transcript into a concrete SHA-256 collision, and the signature reductions lose exactly one EUF-CMA forgery. Hardness enters only at the end: SHA-256 is a fixed, unkeyed function, so following the human-ignorance treatment [21], Corollary 1 states the constructive consequence — an explicit winner yields an explicit collision finder — and labels the security reading as the engineering judgment it is. (A keyed-family restatement is routine and omitted.)

The position-binding and history games are non-interactive: they are universal statements over accepted transcripts, independent of how a transcript was obtained, so adaptive interaction with a proof-issuing service collapses into the adversary's final output. Signatures constrain two other parties — an outsider forging an ordinary head (HEAD), and a third party fabricating equivocation evidence (FORK); those games have a secret and a signing oracle, and their advantage is the probability, over key generation and the adversary's coins, of winning. The adversary may query the signing oracle adaptively on context-valid payloads; Q denotes the set of exact queried payload bytes.

The games at a glance.

Game	Adversary	Secrets	Wins by exhibiting	Consequence
PB	operator (holds key)	none	two openings, one position, $d \neq d'$	collision (Thm. 4)
HIST	operator	none	accepted chain, changed opened value	collision (Thm. 5)
HEAD	outsider	signing oracle	valid head never issued	forgery (Thm. 6)
FORK	third party	signing oracle	evidence pair not fully issued	forgery (Thm. 7)

Policy separation is deliberately not a game: it is a deterministic property of the verdict algorithm (Lemma 5).

Accepted-artifact syntax. A head is $h = (n, r, t; \sigma)$, with t a timestamp; its canonical payload is $\mathsf{EncodeHead}_{\chi_{\text{enc}}}(n, r, t)$ as in §4.2, and $\mathsf{Vf}_{pk}(h) = 1$ iff its Ed25519 signature verifies. An *opening* of leaf d at index m under (n, r) is a path P with $m < n$ and $\mathsf{Root}(\mathsf{h}_{\text{leaf}}(d), m, n, P) = r$ (acceptance in the Option sense of §5.1); write $\mathsf{Open}(d, m, n, P, r) = 1$. An *extension* from (n_0, r_0) to (n_1, r_1) by proof C is exactly a pin-rule transition: either $n_0 = n_1$, $r_0 = r_1$, and C is empty, or $n_0 < n_1$ and $\mathsf{ConsRec}(n_0, n_1, C, \top, r_0) = (r_0, r_1)$; write $\mathsf{Ext}(n_0, r_0, n_1, r_1, C) = 1$.

Lemma 2 (Monotone extensions)**.** *If* $\mathsf{Ext}(n_0, r_0, n_1, r_1, C) = 1$ *then* $n_0 \leq n_1$*; accepted chains of extensions have nondecreasing sizes.*

Proof. Immediate from the two disjuncts of Ext. □

Lemma 3 (Payload injectivity)**.** *For fixed* χ_{enc}, $\mathsf{EncodeHead}_{\chi_{\text{enc}}}$ *is injective on* (n, r, t)*; in particular, distinct* (n, r) *pairs yield distinct payload byte strings.*

Proof. The canonical serialization emits a fixed set of keys in sorted order with fixed separators; the size is a decimal integer, the root a fixed-length lowercase hex string, and the timestamp a JSON string with injective escaping, all under distinct fixed keys, so the encoding parses back uniquely. This is injectivity of the specified serializer over the restricted head schema, not a claim about arbitrary JSON. □

Game PB (position binding). $\mathcal{A}$ outputs (n, r, m, d, P, d', P') and wins iff $d \neq d'$ and

$$\mathsf{Open}(d, m, n, P, r) = \mathsf{Open}(d', m, n, P', r) = 1.$$

Theorem 4 (Scheme position binding)**.** *There is an explicit algorithm $\mathcal{B}_{\mathrm{pb}}$ that, whenever $\mathcal{A}$ wins* PB*, outputs two distinct byte strings with equal SHA-256 digests, using at most the $2(\lceil \log_2 n \rceil + 1)$ hash evaluations of replaying the two openings, with their intermediate values retained.*

Proof. Both accepting folds have the shape determined by (m, n) and output the same root r. Walk from the root downward along the path of m, maintaining that the two transcripts agree on the current node's value. At an internal node the transcripts present preimages $\mathtt{0x01}\|a\|b$ and $\mathtt{0x01}\|a'\|b'$ with equal digests; since child values have fixed 32-byte width, the argument pairs are recoverable from the preimages, so unequal pairs are a collision and equal pairs propagate agreement one level down. If no disagreement occurs, the leaf presents $\mathtt{0x00}\|d$ and $\mathtt{0x00}\|d'$ with equal digests and $d \neq d'$ — a collision. Since the shapes coincide, every comparison is leaf-to-leaf or node-to-node; domain separation would in addition make any cross-type coincidence itself a collision of distinct strings. □

The transport algorithm. For the history theorem we need to move an opening backward through an accepted extension. Recall the recursive verifiers (§4.1, the mechanized form; malformed shapes reject); Figure 2 shows the assembly in a small instance. With k the largest power of two below n:

$$\mathsf{Root}(v, m, n, P) = \begin{cases} v & n = 1,\ P \text{ exhausted} \\ \mathsf{h}_{\mathrm{node}}(\mathsf{Root}(v, m, k, P'),\ s) & m < k \\ \mathsf{h}_{\mathrm{node}}(s,\ \mathsf{Root}(v, m{-}k, n{-}k, P')) & m \geq k, \end{cases}$$

where s is the sibling P supplies for the current level, and

$$\mathsf{ConsRec}(n_0, n, C, b, r) = \begin{cases} (v, v), \quad v = r \text{ if } b \text{ else the next value of } C & n_0 = n \\ (x,\ \mathsf{h}_{\mathrm{node}}(y, s)) & n_0 \leq k \\ (\mathsf{h}_{\mathrm{node}}(s, x),\ \mathsf{h}_{\mathrm{node}}(s, y)) & n_0 > k, \end{cases}$$

where in the second branch $(x, y) = \mathsf{ConsRec}(n_0, k, C, b, r)$ and s is the next value of C, and in the third branch s is the next value of C and $(x, y) = \mathsf{ConsRec}(n_0{-}k, n{-}k, C, \bot, r)$. Both recursions' shapes are determined by their integer arguments, not by the adversary, so two computations at the same arguments traverse the same nodes and there are no mismatched stopping points.

Lemma 4 (Prefix transport)**.** *Suppose $m < n_0 \leq n_1$ and*

$$\mathsf{Ext}(n_0, r_0, n_1, r_1, C) = 1, \qquad \mathsf{Open}(d, m, n_1, P, r_1) = 1.$$

There is an explicit algorithm Transport *returning either two distinct byte strings with equal SHA-256 digests, or a path P_0 with $\mathsf{Open}(d, m, n_0, P_0, r_0) = 1$, using at most the hash evaluations of replaying the two accepted transcripts.*

Proof. If $n_0 = n_1$ then $r_0 = r_1$ and $P_0 = P$. Otherwise $\mathsf{ConsRec}(n_0, n_1, C, \top, r_0) = (r_0, r_1)$, and we prove the following claim by strong induction on n — both sub-calls strictly decrease it — for every sub-call arising in the accepted transcript:

Claim. If $\mathsf{ConsRec}(n_0', n, C', b, \cdot)$ accepts with output (x, y), and $\mathsf{Open}(d, m', n, P', \rho) = 1$ with $m' < n_0'$ and $\rho = y$, then either an explicit collision is output, or a path P_0' with $\mathsf{Open}(d, m', n_0', P_0', x) = 1$.

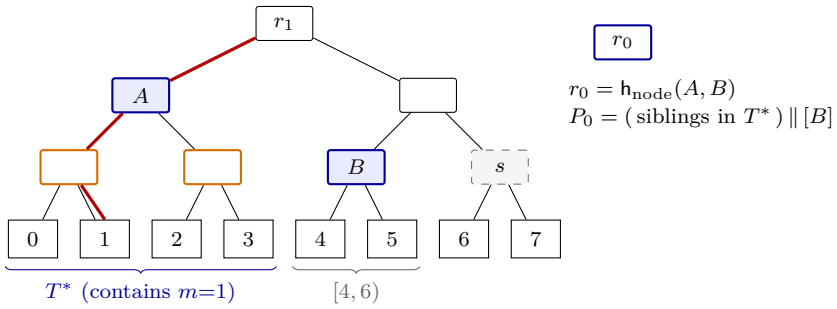

Figure 2: Prefix transport in the $6 \to 8$ instance, opening at index $m = 1$. The accepted consistency transcript pins the frontier values A, B covering $[0, 6)$ (solid blue) and consumes the proof value s covering $[6, 8)$ (dashed). Comparing the opening's fold (red path) with the transcript fixes the opening's value at the frontier subtree T^* containing m. Below T^* the opening keeps its own siblings (orange); above it, the old-root fold $r_0 = \mathsf{h}_{\mathrm{node}}(A, B)$ supplies the one remaining sibling B. The assembled P_0 is the opening's inner path with the new tree's top sibling replaced by B.

Base ($n'_0 = n$): the transcript gives $x = y = \rho$, and $P'_0 = P'$.

Case $n'_0 \le k$, k the split of n: the transcript's second component is $y = \mathsf{h}_{\mathrm{node}}(y_L, s)$ with (x, y_L) the left sub-call's output. Since $m' < n'_0 \le k$, the opening's top step is $\rho = \mathsf{h}_{\mathrm{node}}(u, s_P)$ with $u = \mathsf{Root}(\mathsf{h}_{\mathrm{leaf}}(d), m', k, \cdot)$ accepted on the opening's remaining path. The two preimages of $\rho = y$ are $\mathtt{0x01} \| y_L \| s$ and $\mathtt{0x01} \| u \| s_P$: if the pairs differ, output the collision; otherwise $u = y_L$, and the inductive hypothesis applied to the left sub-call (output (x, y_L)) and the sub-opening (root value $u = y_L$) yields a collision or P'_0 with $\mathsf{Open}(d, m', n'_0, P'_0, x) = 1$, which is the claim since the first component passes through this branch unchanged.

Case $n'_0 > k$: the transcript consumed s and the right sub-call returned (x_R, y_R), so $x = \mathsf{h}_{\mathrm{node}}(s, x_R)$ and $y = \mathsf{h}_{\mathrm{node}}(s, y_R)$. Because $k < n'_0 \le n$ and k is the largest power of two below n, k is also the largest power of two below n'_0 (there is no power of two strictly between k and n; mechanized in the corpus as `kbelow_prefix_eq`), so the n'_0-tree splits at k as well and $x = \mathsf{h}_{\mathrm{node}}(s, x_R)$ is precisely its root form. *If* $m' < k$: the opening's top step is $\rho = \mathsf{h}_{\mathrm{node}}(u, s_P)$ with $u = \mathsf{Root}(\mathsf{h}_{\mathrm{leaf}}(d), m', k, \cdot)$ accepted on the remaining path P'_{in}. Compare preimages of $\rho = y$: unequal pairs are a collision; otherwise $u = s$ and $s_P = y_R$, so the remaining path opens d at m' in the left subtree with root value s, and $P'_0 := P'_{\mathrm{in}} \,\|\, [x_R]$ satisfies $\mathsf{Root}(\mathsf{h}_{\mathrm{leaf}}(d), m', n'_0, P'_0) = \mathsf{h}_{\mathrm{node}}(s, x_R) = x$: an accepting opening, assembled from the opening's own inner path and the transcript's right first-component. *If* $m' \ge k$: the opening's top step is $\rho = \mathsf{h}_{\mathrm{node}}(s_P, u)$ with u accepted at $m' - k$ in the right subtree; comparing preimages of $\rho = y$ either yields a collision or $s_P = s$ and $u = y_R$, and the inductive hypothesis on the right sub-call (output (x_R, y_R), sizes $n'_0 - k \le n - k$, index $m' - k < n'_0 - k$) gives a collision or P''_0 opening d at $m' - k$ under x_R; then $P'_0 := P''_0 \,\|\, [s]$ opens d at m' under $x = \mathsf{h}_{\mathrm{node}}(s, x_R)$.

The top-level instance of the claim has $\rho = r_1 = y$ and $x = r_0$ by acceptance, which is the lemma. Every comparison is between two explicit 32-byte-child node preimages, so each disagreement is a concrete collision; every assembled path entry is either an entry of P, an entry of C, or a sub-call output value, all present in the replayed transcripts. □

In the smallest growth case $2 \to 3$ — the log's own transition in §5.3 — n_0 is a power of two: the frontier is the whole old tree and P_0 is simply the opening's within-prefix tail.

Game HIST (local history binding). $\mathcal{A}$ outputs a chain of head values $h_0, \ldots, h_k$, transition proofs $C_1, \ldots, C_k$ with $\mathsf{Ext}(n_{i-1}, r_{i-1}, n_i, r_i, C_i) = 1$ for every $1 \le i \le k$, indices $0 \le a < b \le k$,

an index $m < n_a$, and openings with $\mathsf{Open}(d, m, n_a, P, r_a) = \mathsf{Open}(d', m, n_b, P', r_b) = 1$ and $d \neq d'$. $\mathcal{A}$ wins iff everything verifies. (The chain is the Merkle-level state sequence a consumer's pin traverses; authentication is HEAD's job, and the binding properties quantify over accepted transcripts regardless of provenance.)

Theorem 5 (History binding)**.** *There is an explicit algorithm $\mathcal{B}_{\text{hist}}$ that, whenever $\mathcal{A}$ wins* HIST*, outputs a SHA-256 collision, using $O(k \log n_k)$ hash evaluations.*

Proof. By Lemma 2, $m < n_a \leq n_i$ for all $i \geq a$. Walk t from b down to $a{+}1$, maintaining an accepting opening of d' at m under (n_t, r_t). If $n_{t-1} = n_t$ then Ext forces $r_{t-1} = r_t$ and the opening carries over unchanged; if $n_{t-1} < n_t$, apply Lemma 4 to C_t and the current opening, obtaining a collision (done) or an accepting opening under (n_{t-1}, r_{t-1}). Arriving at h_a yields two accepting openings of $d \neq d'$ at m under (n_a, r_a), and Theorem 4 extracts the collision. Accepted transcripts have their RFC-determined logarithmic length — malformed lengths reject — so the walk costs at most the evaluations of replaying the k transition transcripts and the two openings. □

Game HEAD **(head authenticity).** A challenger runs KeyGen and signs, on the operator's behalf, the canonical payloads the operator issues in context χ_{enc} (query set Q). The adversary, without the key, outputs a head h and wins iff $\mathsf{Vf}_{pk}(h) = 1$, h encodes χ_{enc}, and h's payload is not in Q.

Theorem 6 (Head authenticity)**.** *For every $\mathcal{A}$ there is an explicit $\mathcal{B}_{\text{ha}}$ with $\mathbf{Adv}^{\mathsf{HEAD}}(\mathcal{A}) \leq \mathbf{Adv}^{\textit{euf-cma}}(\mathcal{B}_{\text{ha}})$: a winning head's exact payload bytes were never queried, so its valid signature is an existential forgery, which $\mathcal{B}_{\text{ha}}$ outputs.*

Game FORK **(fork evidence).** Define the context-scoped evidence predicate: $\mathsf{Ev}_\chi(h, h') = 1$ iff both signatures verify under pk, both heads encode the same χ_{enc}, the tree sizes are equal, and the roots differ. Heads of different logs, schema versions, or hash algorithms never form evidence — one key legitimately operating two logs must not be classifiable as equivocating. *Completeness* is by construction: if the key holder signs two equal-size, unequal-root heads in one context, the pair itself satisfies Ev_χ; producing it requires retention and comparison, not cooperation. *Frame resistance* is the game: the challenger signs the operator's issued payloads in χ_{enc} (query set Q); the adversary, without the key, outputs (h, h') and wins iff $\mathsf{Ev}_\chi(h, h') = 1$ and at least one of the two payloads is not in Q. This is an *issued-message attribution* game: a valid evidence pair proves the key holder signed both conflicting payloads, except with forgery probability.

Theorem 7 (Frame resistance)**.** *For every $\mathcal{A}$ there is an explicit $\mathcal{B}_{\text{fr}}$ with $\mathbf{Adv}^{\mathsf{FORK}}(\mathcal{A}) \leq \mathbf{Adv}^{\textit{euf-cma}}(\mathcal{B}_{\text{fr}})$.*

Proof. A winning pair contains a head whose exact canonical payload bytes were never queried to the signing oracle; its valid signature is an existential forgery, which $\mathcal{B}_{\text{fr}}$ outputs. □

Lemma 5 (Policy separation)**.** *For every leaf a and certificate c, the verdict computed by* Verdict *equals $[\mathsf{Obs}_a(c) = \mathsf{Policy}(c)]$; it reads no operator label, and acceptance consults the operator's status only as a veto. This is a deterministic property of the* Verdict *algorithm, by construction (§3); it is not a hardness statement.*

Definition 3 (Collision-extractable accountability)**.** *A scheme with fixed context χ is* collision-extractably accountable *if there are explicit algorithms, running in time polynomial in the transcript size, that map every winning* PB *or* HIST *output to two distinct strings with equal hash digests; explicit reductions bounding $\mathbf{Adv}^{\mathsf{HEAD}}$ and $\mathbf{Adv}^{\mathsf{FORK}}$ each by one EUF-CMA advantage; and a* Verdict *satisfying policy separation.*

Theorem 8 (Collision-extractable accountability of the construction)**.** *The LTL construction — the RFC 9162 tree, the canonical signed heads of §4.2 in their fixed context χ, the pin rule of §4.3, and the policy verdict of §3 — is collision-extractably accountable, with $\mathcal{B}_{\mathrm{pb}}$ ($\leq 2(\lceil \log_2 n \rceil + 1)$ hash evaluations), $\mathcal{B}_{\mathrm{hist}}$ ($O(k \log n_k)$), and the one-forgery reductions $\mathcal{B}_{\mathrm{ha}}, \mathcal{B}_{\mathrm{fr}}$ of Theorems 4–7.*

Corollary 1 (Constructive security consequence)**.** *For every explicitly given feasible adversary that wins* PB *or* HIST*, the explicitly specified $\mathcal{B}_{\mathrm{pb}}$ and $\mathcal{B}_{\mathrm{hist}}$ constitute an explicitly given SHA-256 collision finder, feasible with the explicit overhead stated in Theorems 4 and 5. For every explicitly given feasible* HEAD *or* FORK *adversary, the stated black-box reductions give an Ed25519 EUF-CMA forger with no loss in success probability, under correct initial acquisition of pk and the fixed context. Under the human-ignorance reading of collision resistance [21] — no feasible SHA-256 collision finder is presently known — this yields the intended security interpretation; that reading is an engineering judgment stated as such, not a mathematical assumption discharged by this corollary.*

What the games do and do not formalize. Against Definition 2: clause (i) is delivered as head authenticity (HEAD) plus opening *uniqueness* (PB) — no distinct leaf can also be opened at an accepted position. Whether a root moreover commits a complete published leaf list is a system property, not a game property: the log publishes its leaves, and a consumer holding the mirror checks membership against the actual list. In short, PB proves leaf-value binding; mirror recomputation proves equality to the published list. Clause (ii) splits into a syntactic part — the pin rule accepts only same-view or verified-extension transitions, by construction — and the semantic part supplied by HIST: a position opened in two accepted views cannot change value without a collision. Clause (iii) is FORK completeness and frame resistance, scoped to χ. Clause (iv) is Lemma 5. The games adapt the established two-transcript secure-logging notions [4] to replay attestation — operator as first-class adversary, policy separation added.

Remark 1 (What is mechanized, what is not). The games are stated for the scheme's specified verifiers — the recursive model whose honest-reference specializations are kernel-checked in entry 13 (the named extractors and per-step pin safety). The two-transcript comparisons and the transport induction are paper-level proofs in the same discipline — the induction reuses the corpus's mechanized `kbelow_prefix_eq` fact — and are not part of the mechanized corpus; HIST supplies, at paper level, the multi-step closure the corpus leaves external. Applying any of these statements to the deployed iterative verifier inherits the refinement boundary of the previous subsection unchanged.

6 Lean and Ed25519 instantiation

6.1 Proof corpus

The initial subjects are upstream `curve25519-dalek/ed25519-dalek` (one implementation: the curve crate and the signature crate atop it) and three deployed forks: Solana/Anza, RISC Zero, and Betrusted — all implementations of Ed25519 [16, 17]. Aeneas provides a functional translation route from Rust to theorem-prover models; its design uses Rust ownership information to avoid explicit memory reasoning for a large class of safe Rust programs [11]. A recent independent experience report likewise applies a Rust-to-Lean pipeline to cryptographic code [22].

Each fork's corpus contained sixteen reviewed certificates at the historical leaves studied here (the corpora have since grown to forty-four per fork — the log records both generations as separate leaves), covering:

- five-limb field arithmetic over $\mathbb{F}_{2^{255}-19}$ with value and bound preservation;

- complete twisted-Edwards group operations [18, 19];
- scalar arithmetic modulo the Ed25519 group order;
- encoding, decoding, and constructive point decompression;
- a four-stage lifting ladder from byte-level verifier acceptance to a mathematical point equation.

The signature apex can be summarized as follows. Let k be the challenge scalar produced by an opaque SHA-512 boundary and let r_1 be the raw R bytes from the signature. The corpus separates:

T1: acceptance iff the verifier's recomputed compressed bytes equal r_1;

T2: those recomputed bytes are the canonical encoding of $[k](-A) + [s]B$;

T3: canonical encoding is injective on valid curve points;

T4: acceptance iff constructive decompression of R yields $[k](-A) + [s]B$.

The separation keeps residual assumptions visible. SHA-512 and selected wire-format interfaces are opaque boundaries at the apex; lower arithmetic and group certificates use the foundational Lean axioms observed in the corpus.

6.2 Replay attestation

For every certificate the operator records:

```
name
status
observed_axioms
expected_axioms          # audit trail; consumer policy is local
axiom_status
diagnostics
```

The attestation also records repository URL and commit, Lean and Lake versions, replay diagnostics, and resource controls. Missing cones are `unverifiable`; they are never interpreted as empty.

6.3 Operational self-reference

The service reports that tree heads are generated using a binary built from the same Ed25519 source family whose verification-path certificates appear in the log. Before signing, the operator recomputes inclusion of the newest signing-library leaf in the tree. This is useful operational coherence, but not proof of execution provenance. The signature authenticates the tree-head payload; it does not reveal the program that produced it. Reproducible builds or execution attestation would be required to establish that stronger claim.

7 Deployment and evaluation

The evaluation asks four questions: (E1) can substantial proof-replay evidence be consumed without deploying Lean; (E2) does the public history retain failed and superseded observations rather than silently replacing them; (E3) can the accumulator's own arguments be placed under the same attestation discipline; and (E4) does mechanization expose mismatches between the proved model and the deployed verifier?

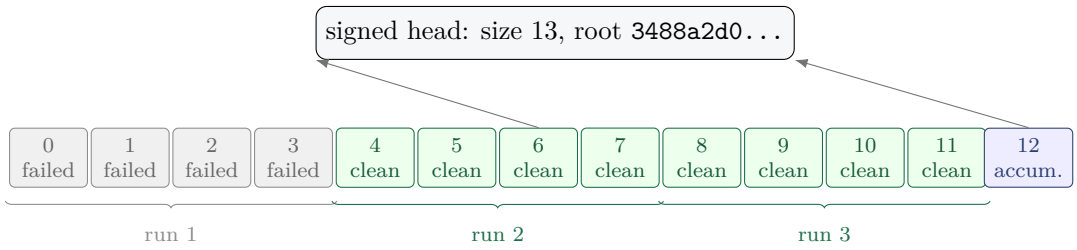

Figure 3: The public 13-leaf deployment. Failure leaves are retained; entry 13 attests the accumulator corpus itself, scoped to the recursive model.

7.1 Public state

As of 16 July 2026, the public log contains thirteen leaves and current root

`3488a2d0ff9f00415bb561d61b01a420e3ca2e0f7b29351ec9ebb3f57319da0d`.

Every signed head issued since public mirroring began is retained — six heads, at tree sizes 8 through 13 — together with every leaf and receipt, in an append-only Git mirror; a clone re-verifies the entire log offline with the repository's standalone verifier. The first twelve leaves are three four-fork replay generations. Leaves 0–3 record a failed audit run and remain permanently visible. Leaves 4–7 record a clean replay. Leaves 8–11 re-attest rewritten repository histories rather than replacing the old leaves. A leaf whose pinned commit ceases to be distributed decays from a replayable claim to a historical record; consumers act only on attestations whose subjects they can retrieve.

Leaf 12 (the thirteenth entry) attests the accumulator corpus at commit

`172a1d0653f489d5b7cb73ac7942a57cbb496532`

It records 61/61 reviewed certificates as proven with exact expected/observed cones. The corpus audit also inventories 222 compiled environment constants and permits exactly one boundary axiom, `LTLAcc.sha256`.

7.2 Mechanization coverage

Entry 13 is not a claim that the whole service is formally verified. The Lean corpus covers the recursive Merkle model, inclusion completeness and collision-extracting soundness, the consistency extractor, and the Merkle-layer share of pin-store safety. The abstract root-binding lemma from the paper is mechanized through the specializations needed by the extractors rather than as one quantified hash-fold theorem. Signature unforgeability, execution provenance, the full signed-head state machine, asymptotic cost, and the refinement from the deployed iterative consistency verifier remain outside the corpus.

Layer	Mechanized evidence	Explicit boundary
Merkle definitions	MTH, Root, Path, recursive ConsRec	single SHA-256 boundary axiom
Inclusion	completeness and named collision extractor	collision resistance interpreted externally
Consistency	recursive-model soundness and extractor	no general consistency-completeness theorem
Pinning	per-step monotonicity and prefix correctness	signature layer and multi-step closure external
Deployment refinement	finite differential harness	no theorem for iterative verifier under authentic-pair invariant
Policy separation	deterministic tooling logic and regression tests	not mechanized in the entry-13 corpus
Scheme-level games (§5.4)	paper-level explicit reductions	two-transcript comparisons and prefix transport not mechanized

7.3 Cost and reproducibility

A replay of one Ed25519 fork requires approximately 30 minutes of end-to-end guarded replay time under the pinned environment, a figure corroborated by the inter-leaf issuance spacing visible in the published log. Receipt verification requires one Ed25519 signature and a logarithmic number of SHA-256 node computations. The accumulator corpus is independently reviewable with a pinned public Lean release; an environment-derived inventory fails closed on added, removed, or axiom-smuggling declarations.

The fidelity harness compares the Lean-definition transliteration with the deployed Python algorithms over pinned finite families:

Family	Cases	Divergences	Interpretation
Inclusion	230,271	0	baseline and out-of-range families
Consistency baseline	230,016	0	honest and mutation families
Lied-size consistency	73,573	3,867	all deployed-accepts-only

Finite testing is not a proof of extensional equality. Here it served a more valuable purpose: it falsified an overbroad equivalence claim and supplied a stable regression boundary.

Remark 2 (Model/deployment seam). For malformed size claims, the deployed iterative verifier and the recursive model are not extensionally equal (figures are the pre-closure measurement; the $sn = 0$ restoration reduces the divergence count in this family to zero). In all 3,867 divergences observed across the pinned families the deployed verifier accepted and the model rejected; the reverse direction was not observed, and no global inclusion relation between the two acceptance sets is claimed. The public attestation therefore scopes soundness to the recursive model and states the additional operational assumption: roots and sizes must be bound by the authenticated pin-store and signed-head flow. This invariant is not mechanized in the present corpus.

7.4 Consumer prototypes and version exactness

The implemented internal consumer is a quorum-custody signing prototype: its inbound boundary accepts a log-derived statement only when independently attested verifier backends agree, and its policy consumes recorded observations, never operator labels. Separately, an informal check found a production codebase whose vendored Ed25519 dependency matches an attested subject at family level but not at the attested version; the model treats a family-level match as conferring nothing, because attestations are version-exact by construction. Neither observation is an evaluation claim; both indicate how the policy boundary is consumed in practice.

7.5 Proof portability across forks

Pure mathematical lemmas are largely reusable, while extraction-facing scripts diverge where code structure and generated names diverge. In the deployed corpora, the RISC Zero and Betrusted signature-layer proof files differ by 27 changed lines (tracking one fork's optimization barrier and the forks' differing operation order), other extraction-facing files differ by tens to hundreds of lines, and pure carry and field lemmas remain byte-identical. This supports a practical conclusion: verification is portable above the representation boundary and target-specific where implementation structure actually differs.

8 Related work

Transparency. Certificate Transparency introduced publicly auditable append-only logs for certificate issuance [1, 2]; Crosby and Wallach developed efficient tamper-evident history trees [3]; Dowling et al. formalized security notions for secure logging and CT [4] — the games of §5.4 adapt that two-transcript style to replay attestation, with the operator as first-class adversary and policy separation as a deterministic functionality. CONIKS applies transparency to key directories [7]. LTL reuses the authenticated data structure but changes the payload and trust semantics: a leaf is an observation of a proof replay, not an issuance event or key binding.

Software supply-chain attestations. In-toto expresses supply-chain steps and link metadata [6]. Sigstore combines ephemeral signing, identity, and transparency to reduce software-signing adoption barriers [5]. LTL is complementary: it concerns what a theorem prover reportedly accepted and which assumptions remained, not who built or signed a binary. A complete assurance chain should eventually combine both.

Proof transport and verified cryptography. Proof-carrying code ships a proof to a consumer-side checker [8]. LTL serves consumers that cannot deploy that checker and therefore accepts a different trust trade. HACL*, EverCrypt, and Fiat-Crypto demonstrate verified cryptographic implementation pipelines [13, 14, 15]; Computer-aided frameworks such as EasyCrypt address scheme-level security proofs [10]; Aeneas targets functional verification of Rust through translation [11]. LTL does not compete with those systems: it distributes accountable statements about their replay.

Verification of transparency protocols. Cheval et al. mechanize transparency-protocol reasoning [9]. The entry-13 corpus approaches the composition from the opposite direction: it mechanizes accumulator arguments and then logs that replay result. The remaining refinement from the deployed state machine to the recursive model is explicitly open.

9 Limitations and research agenda

The subject corpus maintains a numbered ledger of fifteen known gaps together with their closure options; this section groups the load-bearing ones.

Operator observation trust. A malicious operator can fabricate a replay report. Signatures and Merkle proofs make the lie attributable and persistent; they do not make it true. Targeted independent replay is the corrective mechanism.

Replay-harness integrity. A wrong observation needs no malice: a defective replay harness — a bug in the audit driver, a fail-open guard, a truncated transcript — produces the same evidentiary damage as a dishonest operator, with the same accountability answer (the record is attributable and persistent; independent replay corrects it). The subject corpus's adversarial gate self-tests exist for exactly this reason and reduce, but cannot eliminate, the exposure.

Theorem identity. Names and repository commits are not canonical semantic identifiers, and commit identifiers are SHA-1-based — a weaker binding than the log's own SHA-256 tree. A future schema should commit to elaborated theorem-type digests, axiom declaration-type digests, and an environment or replay-manifest digest.

Source-to-binary gap. The evidence concerns source models at pinned commits. Reproducible builds, compiler validation, binary measurement, and side-channel evidence are outside the present result.

Witnessing and key distribution. The deployment has one operator and trust-on-first-use key distribution. A Git mirror gives retaining observers a common public view, but does not force all isolated clients to receive that view. Independent witnesses or gossip are the natural next deployment step.

Consistency refinement. The recursive model is proved; the iterative deployment diverges from it on malformed inputs, every observed divergence being deployed-accepts-only. The strongest closure is either to deploy ConsRec-equivalent semantics or to mechanize the signed-head and pin-store flow and prove the authentic-pair refinement theorem.

Signed provenance. Signing-backend metadata is operator-provided context and should be committed inside the signed tree-head payload. Even then it would remain an assertion, not execution proof.

From accountable replay to cryptographic proof of replay. A longer-term direction is a succinct proof that a fixed proof-checker binary accepted a fixed corpus. Such a system could reduce operator-observation trust, but would introduce a new verified-execution stack. LTL supplies an intermediate accountability layer and a public corpus against which that future system can be evaluated.

10 Conclusion

Formal verification solves the production of correctness evidence; it does not by itself solve distribution to consumers that cannot execute the verifier. This paper isolates that second problem and gives a cryptographic answer based on accountable replay attestation. The operator's observation remains trusted, but its content is structured, its history is signed and append-only, its assumption boundary is subject to consumer-local policy, and incompatible views become attributable when compared.

The Lean Transparency Log demonstrates the complete construction. It amortizes expensive replay over lightweight consumers, retains failed and superseded observations, and carries a scoped attestation of the accumulator's own Lean corpus as entry 13. Just as importantly, the mechanization and differential harness exposed a mismatch between the recursive model and the deployed consistency verifier. Recording that mismatch in the public leaf is not a failure of the method; it is evidence that the trust decomposition is doing useful scientific work.

The next step is not to claim trustlessness. It is to close specific boundaries: canonical theorem-type commitments, reproducible source-to-binary linkage, independent witnesses, signed provenance commitments, and a proved refinement between the deployed signed-head flow and the recursive model. Accountable replay attestation provides an immediate infrastructure layer while those stronger validity mechanisms are developed.

Artifact availability

The live service is `https://ltl.zkdefi.org`. Entry 13 has leaf hash

```
8cb258d657f1fd00baaa9e0091e26c316cb69b591cb249a9543f51cade57c50a
```

and is included in the size-13 head with root

```
3488a2d0ff9f00415bb561d61b01a420e3ca2e0f7b29351ec9ebb3f57319da0d
```

The public artifacts are available at:

- append-only mirror: `saymrwulf/lean-transparency-log`;
- accumulator mechanization: `saymrwulf/ltl-accumulator-verified`;
- provider and consumer tooling: `saymrwulf/proof-aware-crypto-tooling-agent`.

A clone of the mirror re-verifies every numbered leaf, every published signed head, and every published receipt offline via `python3 verify.py --all` (Python standard library plus an `openssl` binary; the verifier fails closed if signature checking is unavailable, and its adversarial self-test ships beside it).

Acknowledgments

The author designed the system and is responsible for every claim. Claude (Anthropic) and GPT (OpenAI) were used as critical assistants in proof-corpus, tooling, and manuscript review. Their output was not accepted as evidence; claims were retained only after human review or reproducible artifact checks.

References

[1] B. Laurie, A. Langley, E. Käsper. Certificate Transparency. RFC 6962, 2013.

[2] B. Laurie, E. Messeri, R. Stradling. Certificate Transparency Version 2.0. RFC 9162, 2021.

[3] S. A. Crosby, D. S. Wallach. Efficient Data Structures for Tamper-Evident Logging. USENIX Security, pp. 317–334, 2009.

[4] B. Dowling, F. Günther, U. Herath, D. Stebila. Secure Logging Schemes and Certificate Transparency. ESORICS, LNCS 9879, pp. 140–158, 2016.

[5] Z. Newman, J. S. Meyers, S. Torres-Arias. Sigstore: Software Signing for Everybody. ACM CCS, pp. 2353–2367, 2022.

[6] S. Torres-Arias, H. Afzali, T. K. Kuppusamy, R. Curtmola, J. Cappos. in-toto: Providing farm-to-table guarantees for bits and bytes. USENIX Security, pp. 1393–1410, 2019.

[7] M. S. Melara, A. Blankstein, J. Bonneau, E. W. Felten, M. J. Freedman. CONIKS: Bringing Key Transparency to End Users. USENIX Security, pp. 383–398, 2015.

[8] G. C. Necula. Proof-Carrying Code. ACM POPL, pp. 106–119, 1997.

[9] V. Cheval, J. Moreira, M. Ryan. Automatic verification of transparency protocols. IEEE EuroS&P, pp. 107–121, 2023. doi:10.1109/EuroSP57164.2023.00016. arXiv:2303.04500.

[10] G. Barthe, B. Grégoire, S. Heraud, S. Zanella Béguelin. Computer-Aided Security Proofs for the Working Cryptographer. CRYPTO, LNCS 6841, pp. 71–90, 2011.

[11] S. Ho, J. Protzenko. Aeneas: Rust verification by functional translation. Proc. ACM Program. Lang. 6 (ICFP): 711–741, 2022.

[12] L. de Moura, S. Ullrich. The Lean 4 Theorem Prover and Programming Language. CADE-28, LNCS 12699, pp. 625–635, 2021.

[13] J.-K. Zinzindohoué, K. Bhargavan, J. Protzenko, B. Beurdouche. HACL*: A Verified Modern Cryptographic Library. ACM CCS, pp. 1789–1806, 2017.

[14] J. Protzenko et al. EverCrypt: A Fast, Verified, Cross-Platform Cryptographic Provider. IEEE S&P, pp. 983–1002, 2020.

[15] A. Erbsen, J. Philipoom, J. Gross, R. Sloan, A. Chlipala. Simple High-Level Code for Cryptographic Arithmetic—With Proofs, Without Compromises. IEEE S&P, pp. 1202–1219, 2019.

[16] D. J. Bernstein, N. Duif, T. Lange, P. Schwabe, B.-Y. Yang. High-speed high-security signatures. J. Cryptographic Engineering 2(2): 77–89, 2012.

[17] S. Josefsson, I. Liusvaara. Edwards-Curve Digital Signature Algorithm (EdDSA). RFC 8032, 2017.

[18] D. J. Bernstein, T. Lange. Faster addition and doubling on elliptic curves. ASIACRYPT, LNCS 4833, pp. 29–50, 2007.

[19] D. J. Bernstein, P. Birkner, M. Joye, T. Lange, C. Peters. Twisted Edwards curves. AFRICACRYPT, LNCS 5023, pp. 389–405, 2008.

[20] A. Pnueli. The temporal logic of programs. IEEE FOCS, pp. 46–57, 1977.

[21] P. Rogaway. Formalizing Human Ignorance: Collision-Resistant Hashing without the Keys. VIETCRYPT, LNCS 4341, pp. 211–228, 2006. doi:10.1007/11958239_14.

[22] N. Klaus, J. Conejero, P. Tolmach. A Rust-to-Lean Verification Pipeline with AI Provers: An Experience Report. arXiv:2605.30106, 2026.

A End-to-end claim matrix

Consumer conclusion	Established by	Remaining assumption
Leaf has an authentic opening with a position-bound leaf value at index m under head h	inclusion proof and signed head	SHA-256 collision resistance; correct public key; EUF-CMA of the head signature
Head root commits the published numbered leaf list	full-mirror recomputation (`verify.py --all`)	mirror availability and retention
Head was authorized by the log identity	Ed25519 verification	correct key acquisition; EUF-CMA
New pinned head extends old pinned head	consistency proof	SHA-256 collision resistance; recursive-model soundness; authentic size/root pairing for deployment
Equal-size unequal roots in one log context conflict	two valid signatures	correct public key; EUF-CMA; operationally, a retaining observer must compare the heads
Observed cone matches local boundary policy	exact set equality	semantic identity of named declarations
Operator claims the kernel produced the observation	attestation signature and leaf inclusion	correct provider key; EUF-CMA
Kernel actually produced the recorded observation	not cryptographically established; independently checkable by replay	operator and replay-pipeline honesty, or faithful independent replay
Recorded cone was produced by an audit that performed its checks	not established — the audit driver is itself part of the replay pipeline	audit-gate integrity; adversarial gate self-tests reduce this exposure, they do not eliminate it
Source corresponds to deployed binary	not established	reproducible build and compiler assurance
Claimed signer implementation produced STH	not established	execution provenance

B Deployed entry-13 scope

The thirteenth public leaf contains the following deployment constraint, quoted verbatim, in its machine-readable scope block:

> Attestation scope: this corpus kernel-checks the listed theorems about the mechanized recursive accumulator model. Correspondence with the deployed inclusion verifier is supported by finite differential testing over the pinned families. The deployed consistency verifier is not extensionally equal to the model; applying the mechanized soundness result to the deployed consumer flow additionally relies on an unmechanized authentic-size/root invariant (KNOWN-GAPS 14/15).

Its exclusions name SHA-256 collision resistance, deployed-verifier extensional equality, the signature/STH layer, and asymptotic cost claims.

C Compact receipt-verification core

The following code is only the Merkle inclusion core. A complete receipt verifier must additionally validate the signed tree head, log identifier, tree-size binding, public-key fingerprint or pinned key, leaf hash, and receipt schema. The published log-repository verifier implements that full binding list for every published receipt — the binding fields are required, never compare-if-present — and fails closed when signature checking is unavailable.

```
import hashlib

def H(data):
    return hashlib.sha256(data).digest()

def h_leaf(data):
    return H(b"\x00" + data)

def h_node(left, right):
    return H(b"\x01" + left + right)

def split_below(n):
    return 1 << ((n - 1).bit_length() - 1)

def take(path, used):
    if used >= len(path):
        raise ValueError("proof exhausted")
    return path[used]

def root(value, index, size, path, used=0):
    if size == 1:
        return value, used
    k = split_below(size)
    if index < k:
        left, used = root(value, index, k, path, used)
        return h_node(left, take(path, used)), used + 1
    right, used = root(value, index-k, size-k, path, used)
    return h_node(take(path, used), right), used + 1

def verify_inclusion(leaf, index, size, path, expected_root):
    if size <= 0 or index < 0 or index >= size:
        return False
    try:
        result, used = root(h_leaf(leaf), index, size, path)
    except ValueError:
        return False
    return used == len(path) and result == expected_root
```

D Four Ed25519 verification tiers

Tier	Meaning	Upstream Lean declaration
T1	byte-level acceptance equation	`verify_accepts_iff`
T2	canonical encoding lift	`verify_accepts_iff_point`
T3	injectivity / point equation	`verify_accepts_iff_point_eq`
T4	constructive decompression lift	`verify_accepts_iff_decompress`