SECURITY & TRUST

From Action to Forensic Evidence

Every security-relevant action becomes a record you can prove unchanged — and a family of signed artifacts you can check with a verifier you run yourself.

September 2026 · v1.0 · 18 min read

01 The thesis: tamper-evident, not immutable
02 How an action is recorded
03 How the record shows tampering
04 A family of signed evidence, not one log
05 What a customer or auditor can verify
06 Tamper-evidence and erasure, reconciled
07 Making cryptographic evidence affordable

Executive summary

Most audit stories end in a promise that the log is immutable. This paper makes a different, checkable claim: a record can be proven unchanged. Any in-place modification, gap or reorder of the recorded events is detectable by a verifier the customer can run, against published keys and an external anchor. The distinction matters most where it is least comfortable — a governed database has to be able to erase, and an immutable log cannot forget.

The paper follows one action from occurrence to evidence. It is recorded as an identity-bound event from a catalogued vocabulary, on the action's own path rather than as sampled telemetry. It joins two complementary tamper-evidence surfaces — a signed hash chain over custody operations and a per-row change trail sealed by periodic signed anchors. And it belongs to a family of distinct, domain-separated signed artifacts, each of which is checked by its signature and its type, never by one verifier applied to everything.

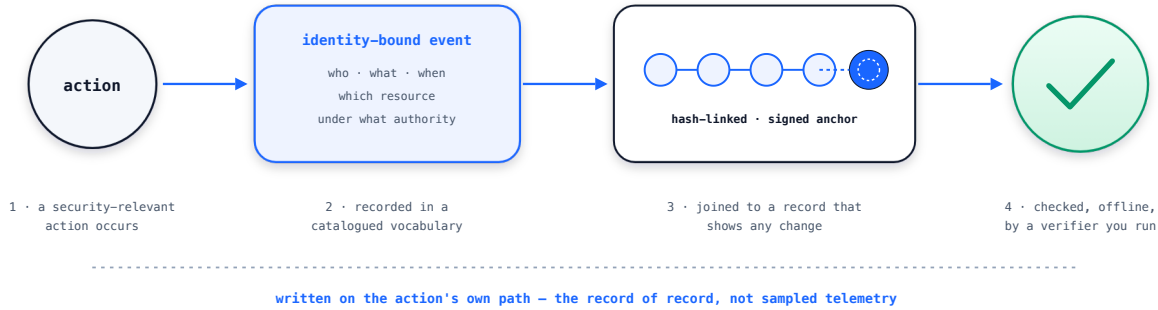
It closes with how erasure is itself a signed, evidenced operation rather than a hole in the record, how the design scales by amortizing signatures across segments instead of signing every row, and — as in every paper in this series — exactly where each claim ends.

01 The thesis: tamper-evident, not immutable

The usual promise is that the audit log is immutable. For a governed database it is the wrong promise, for two reasons. An immutable log cannot forget, and a system that has to honour erasure must be able to erase. And “impossible to change” is a claim no one outside the vendor can check.

The claim this paper makes is the checkable one: the record can be proven unchanged. Any in-place modification, gap or reorder of the recorded events is detectable — by a verifier a customer can run, against published keys and an external anchor. Tamper-evident, not immutable. Every claim that follows resolves to “you can detect and prove tampering,” never to “tampering is impossible.”

The second idea is that Kilimanjaro does not produce a log; it produces a family of signed evidence. Custody-chain entries, transparency anchors, audit-export commitments, decision receipts, erasure receipts and mint authorizations are each a distinct, signed, independently checkable class, and each is domain-separated so that one class can never be presented as another. Two audit surfaces stay deliberately distinct throughout: a tamper-evident chained record of custody operations, and an identity-aware record of every security-significant action. Collapsing them into a single chain would claim more than the design provides.

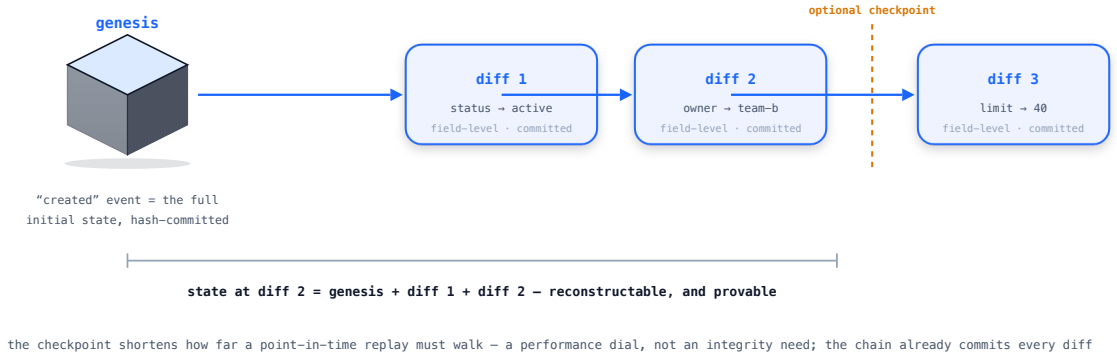


An action becomes an identity-bound event, joins a record in which any change shows, and ends as evidence that is checked offline with a verifier the customer runs. The record is written on the action's own path — it is the forensic record of record, not sampled telemetry.

02 How an action is recorded

Every security-significant action is recorded as an identity-bound audit event: who acted, what they did, when, on which resource, and under what authority. Events are drawn from a catalogued, versioned vocabulary rather than written as ad-hoc log lines, so an auditor is reading a defined schema, not free text. These events are the forensic record of record, written on the action's own path — they are not sampled metrics, and they are not a best-effort side channel that can fall behind the action it describes.

A change to a governed record is captured as a field-level diff from a hash-committed genesis. The “created” event carries the full initial state; every later change is a committed diff. Because each diff is committed against what came before, any point-in-time state can be reconstructed by replaying the genesis and the diffs up to that point — and proven, because the commitments must still line up. Optional snapshot checkpoints bound how far a replay has to walk.



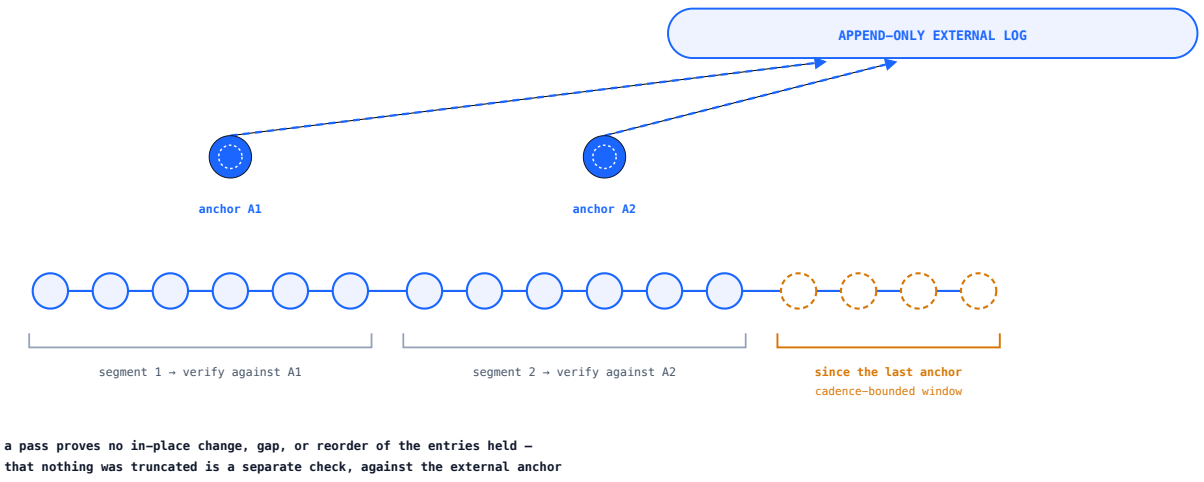
A governed record begins with a genesis — the “created” event, holding the full initial state and hash-committed — and every later change is a committed field-level diff. The state at any point is the genesis plus the diffs up to it, so any past state is reconstructable and provable. Optional checkpoints shorten replay; they are a performance dial, not an integrity need.

One boundary belongs here, because it governs how the rest of the paper should be read: not every event is in the cryptographic hash chain. The chain is a subset. Routine, high-volume operations are fully audited and identity-bound but are not individually chained — a deliberate scale choice, discussed in the final section, not a gap in coverage.

03 How the record shows tampering

There are two complementary tamper-evidence surfaces. The first is a signed hash chain over custody operations — the key lifecycle: rotation, revocation, custody-mode changes and erasure. Each entry links to the previous entry by hash and carries a gap-free sequence number, so a missing entry shows as a sequence gap. The platform-managed regional custody chain is rooted in a FIPS 140-3 Level 3 Managed-HSM key at the Secure Key Release (SKR) custody tier, with the floor tier's root held as a software-protected sovereign vault secret. The second surface is a Governed-Database change-tracking trail: a per-row hash chain over the change history of your data, sealed by periodic signed transparency anchors.

The hash chain makes any in-place modification, gap or reorder of the recorded events detectable. The transparency anchors extend that beyond the platform's own word: each anchor is a signed commitment over a segment of the change trail, published to an append-only external log, so a customer can confirm after the fact that the platform did not retroactively modify the ledger by comparing an anchor to their own re-derivation of the segment.



Entries are hash-linked; periodic signed anchors seal each segment and are published to an append-only external log. An auditor verifies a bounded segment against its anchor. Entries since the last anchor form a cadence-bounded window that is not yet sealed — and a passing check proves no in-place change, gap or reorder of the entries held, not that nothing was truncated.

Three boundaries are material. Detection has a bounded window: events since the last anchor are not yet anchored, and the anchoring cadence is tunable per project — there is no zero-latency claim. Tamper-evidence rests on signing-key custody, which is why the claim is tamper-evident rather than impossible to forge. And intact is not complete: a passing verification proves that there was no in-place tamper, gap or reorder of the events held, while proof that nothing was truncated from the end is a separate check, made against the external anchor.

04 A family of signed evidence, not one log

The organizing principle is domain separation. Every signed artifact carries an issuer and a type in its signed header. Different evidence classes may share a signing key, or an issuer, but the type keeps them cryptographically distinct — so a receipt of one class can never be verified, or replayed, as another. The verification rule is the same for every class below: check the signature and the type.

The classes fall into two kinds. Customer-verifiable evidence is something you download or receive and check offline. Platform-enforced evidence is a signed gate the platform itself requires: you see its effect, and its existence — failing closed when it is absent — is the guarantee.



Distinct evidence classes can share a signing key or issuer, yet each carries its own type in the signed header. A verifier checks the signature and the type; a receipt of one class presented as another fails the type check. Public keys for every class are published at a well-known key-set endpoint.

Class	What it evidences	Who checks it
Custody-ledger entry	Each key-lifecycle operation, as a signed entry in a hash chain with a gap-free sequence number	Customer — chain-walk verifier
Transparency anchor	A periodic signed commitment over the change-tracking trail, published to an append-only external log	Customer — signature + type
Audit-export commitment	A signed commitment over an exported audit range: the export you hold is the export produced	Customer — signature + type
Warden decision receipt	A standalone signature over a policy decision	Customer — signature + type
Erasure receipts	The signed KEY_PURGED custody entry, and a minimised erasure receipt with a random reference	Customer — chain walk + signature
BYOS mint authorization	A ledger-signed authorization that must exist before a credential is minted into your own storage	Platform — fails closed without it
Exec-capability receipt	A ledger-signed capability the execution plane must present in order to act	Platform — fails closed without it
Attestation-tier evidence	The serving node's declared trust tier with a verified bit, adjudicated against your trust-tier policy; hardware-backed on confidential-computing nodes	Platform — signed response field

The evidence classes. The first five are checked offline by the customer; the last three are signed gates the platform enforces and fails closed on.

05 What a customer or auditor can verify

Your Governed-Database audit trail is verified by a formally-proven verifier across three namespaces (project, org and platform); the producer reproduces the verified bytes, confirmed by a cross-language conformance test and real-export validation. A customer can download the ledger and verify it client-side with that verifier, and can use the external transparency anchor to confirm the platform did not retroactively modify it.

Every Warden decision returns a receipt — { ledgerId, signature } — where signature is a standard JWS signature over the decision, produced by the platform's ledger key (EdDSA or RS256, per the receipt's alg header), and ledgerId correlates to the decision's entry in your project's audit stream.

Signing keys are published as an RFC 7517 JWKS at /.well-known/ledger-jwks (the platform's ledger_anchor key family); verify a receipt offline by resolving its kid against that set with any standard JOSE/JWS verifier, and confirm the receipt's typ is k9o.warden.gate-receipt.v2. Any change to a receipt breaks its signature. Warden receipts are standalone signed commitments — they are not entries in the hash chain, which covers custody-lifecycle key operations, and the chain-walk verifier does not apply to them.



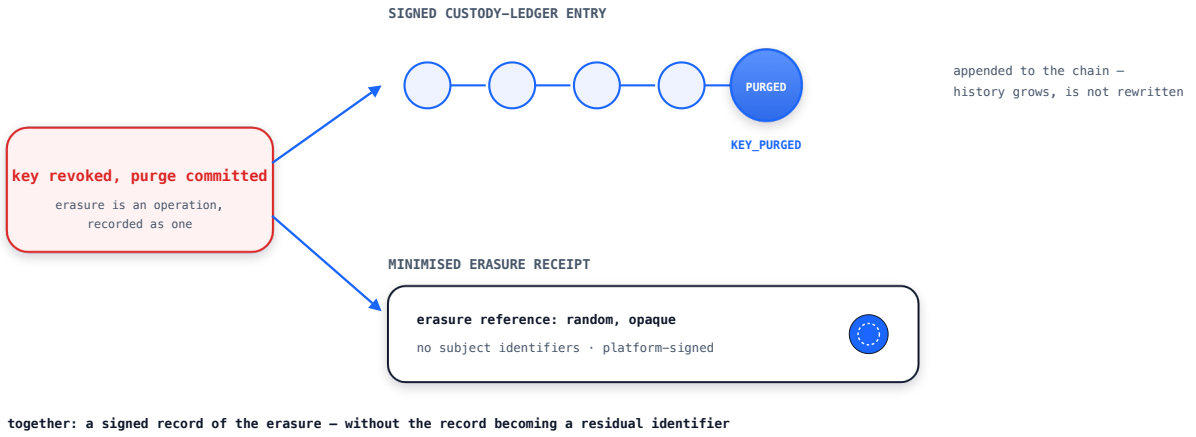
Two verifiers, never conflated. A formally proven chain-walk verifier validates the custody chain and the change trail across three namespaces. A standard JOSE/JWS verifier, plus a type check, validates the standalone signed artifacts. Platform-enforced gates are signed and fail closed, but they are not something the customer re-verifies offline.

The boundaries are as important as the claims. The chain-walk verifier is formally proven; the canonicalization and the hash function beneath it are trusted under cross-language test, and this paper does not describe them as formally proven. Resistance to the platform itself needs the external anchor — a purely local verification is made against the platform's own signed anchor.

06 Tamper-evidence and erasure, reconciled

The two properties look opposed: a record that shows every change, and a system that has to forget. They are reconciled by making erasure an operation the record itself evidences. Erasure produces platform-signed evidence in two forms — the signed KEY_PURGED entry appended to the custody ledger, which is the signed record that the platform revoked the key and committed to its purge, and a minimised erasure receipt carrying a random erasure reference and no subject identifiers.

For your declared-encrypted footprint — the fields and tokenised values you have designated for encryption — Kilimanjaro provides cryptographic erasure by destroying the platform-held key, rendering those encrypted values computationally unrecoverable through the platform in both live storage and applicable backups, within the 28-day erasure SLA, with completion evidenced by platform-signed shred receipts.



Erasure is itself a signed, evidenced operation. The custody chain gains an appended KEY_PURGED entry — history grows, it is not rewritten — and a minimised receipt carrying only a random erasure reference gives a customer a signed record of the erasure without the record itself becoming a residual identifier.

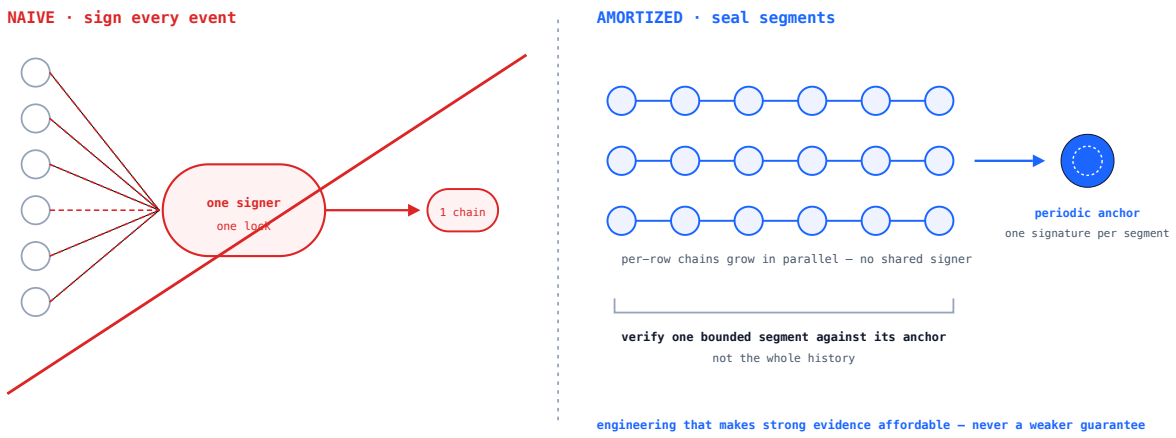
Personal data you leave unencrypted is erased through deletion, with copies retained solely in backup systems being removed in accordance with the applicable backup-retention cycle, within the same 28-day SLA. Where your personal data also appears as a reference inside another record owner's data — including in derived context or embeddings — that cross-owner occurrence is a disclosed limitation of record-owner-scoped erasure: the occurrence is addressed through the applicable custodian-scoped erasure process, but is not necessarily independently attributable to or selectively erasable from the other owner's record.

The boundaries are part of the claim. Erasure is completed within a defined window, not at the moment of the request. Copies that have left the platform for an external sink are outside the platform's control — the customer is the controller of those — and the default-on erasure notice and the egress audit scope what each sink received. Storage owned by an extension is outside it.

07 Making cryptographic evidence affordable

The naive design signs every event into one serialized chain, and it does not scale: a signed, lock-serialized append on every high-volume key operation is an architectural failure, not a tuning problem. What was built instead keeps the guarantees and removes the serialization.

The high-volume change trail is a per-row hash chain sealed by periodic signed anchors, not a signature per row; verification is identical — a chain walk — but a customer verifies a bounded, anchored segment instead of walking billions of rows. Only the bounded, high-value custody-lifecycle operations are individually chained; routine high-volume key operations are fully audited and identity-bound without being chain-appended, because where per-event chaining does not earn its cost, the design does not pay it.



Signing every event into one serialized chain caps throughput at a single signer and lock. Here, per-row chains grow in parallel and a periodic anchor seals each segment with one signature; an auditor verifies a bounded segment against its anchor instead of re-walking all of history.

Around that core sit smaller decisions with the same purpose. Wrap, unwrap and data-key generation are idempotent on a request identifier, so a replay returns the prior result without performing the operation again and retries are cheap and safe. One formally proven verifier validates the project, org and platform chains unchanged, because the producer conforms to its exact byte format — there is no separate verifier, and no re-proof, per namespace. And outbound event bridges hold a persistent connection with periodic revocation polling rather than connecting and disconnecting per batch; the connection is re-secured on every open — re-guarded, re-pinned, credential re-resolved — and bounded by a hard maximum age.

Choice	What it buys
Periodic signed anchors over a per-row chain	One signature per segment, not per row; verification stays a chain walk
Bounded chaining	Chain only the high-value custody operations; audit the rest, identity-bound
Idempotent key operations	Replays return the prior result; retries are cheap and safe
Segment verification	Verify one anchored segment, not the whole history
One verifier, three chains	No separate verifier or re-proof per namespace
Snapshot checkpoints	Bound point-in-time replay length — a performance dial, not an integrity need
Durable bridge connection	Lower per-event overhead; re-secured on every open and age-bounded

The design choices that let evidence keep pace with data-plane volume.

What we disclose

This paper claims tamper-evidence, not immutability. The record can be proven unchanged; it is not presented as unchangeable. That evidence rests on the custody of the signing keys, which is why it is described as tamper-evident and never as impossible to forge.

Detection has a bounded window: events since the last anchor are not yet anchored, and the cadence is a tunable per-project setting. A passing verification means intact, not complete: it proves no in-place change, gap or reorder of the events held, and it does not by itself prove that nothing was truncated — that is the separate check against the external anchor.

Not every event is in the hash chain. The chain covers the bounded, high-value custody-lifecycle operations; routine high-volume operations are fully audited and identity-bound but not individually chained. The two verifier families are not interchangeable, and platform-enforced signed gates are not something a customer re-verifies offline.

Copies at an external destination are outside the platform's control, as are a reference to a person inside another record-owner's data and extension-owned storage.

See what the evidence would prove for your records

Talk to a Sovereign Engineer about which of your audit, custody and erasure obligations map to which class of signed evidence, and what your auditors can verify without asking us.

Talk to a Sovereign Engineer →

Read the erasure paper