§ WORKING PAPER

receipt: verifiable custody of agent-produced records

Max Ghenis

Axiom Foundation

Revision 14 · 2026-09-10

Abstract

An auditor who holds a clone of a rule corpus, and a few trust pins committed in their own repository, can decide offline whether the records in it are the ones a pinned key signed and pinned witnesses saw, and whether any release object seen at first contact has since been changed or deleted. receipt renders that verdict. Four inputs fix it: the tree object a named commit carries, a base tree when the auditor supplies one, the specification the auditor pinned, and the trust material the auditor owns; the verifier's own clock bounds how far ahead a token may sit. Every record byte the verdict speaks for comes from the repository's object store and is rehashed against its name; no record byte comes from a working directory. A pass establishes integrity of the release chain, authorship by the pinned key, existence no later than the witnessed times, closed-world binding of the corpus to its journal, and, with a base, preservation of every release object the base held. It does not establish how the bytes were produced, that the history is the only one the producer keeps, or that a chain regenerated before first contact is the original. Two repositories pin the package on their default branch, one of them in an optional extra, and call it from their verification paths; a third pins it on a branch short of its default, and a fourth shows its output.

STATUS

Working paper. It describes `receipt` 0.6.1 on PyPI (2026-09-06). Its source differs from 0.6.0 (2026-09-05) in the signing module and the version string alone: two refusals the review below surfaced, noted where they apply. Code: github.com/TheAxiomFoundation/receipt. API reference: axiom.org/receipt/api.

1 Two paths

An encoded rule turns out to be wrong: a rate reads 0.15 where the statute says 0.17. The fix can arrive two ways. The pipeline path re-encodes. The encoder runs again, and the corrected file lands in a fresh release, signed and witnessed, with a journal row behind it. The shortcut edits the published file by hand and commits the edit. Both paths produce the same YAML. Run from a clone, `receipt verify` tells them apart after the fact (Figure 1). The re-encoded fix passes as release 0002 with its journal row, and the hand edit refuses, because the tree carries bytes the witnessed journal never bound. The command reads the commit's tree from the object store, never the checkout, and its verdict names the commit and the tree it judged.

The command cannot tell an encoder run from bytes a producer wrote by hand and then journaled, signed and witnessed. That distinction needs a record of the run itself, and the corpora make it separately (Section 5). Nor does it see an edit that was never committed: the checkout is outside its subject.

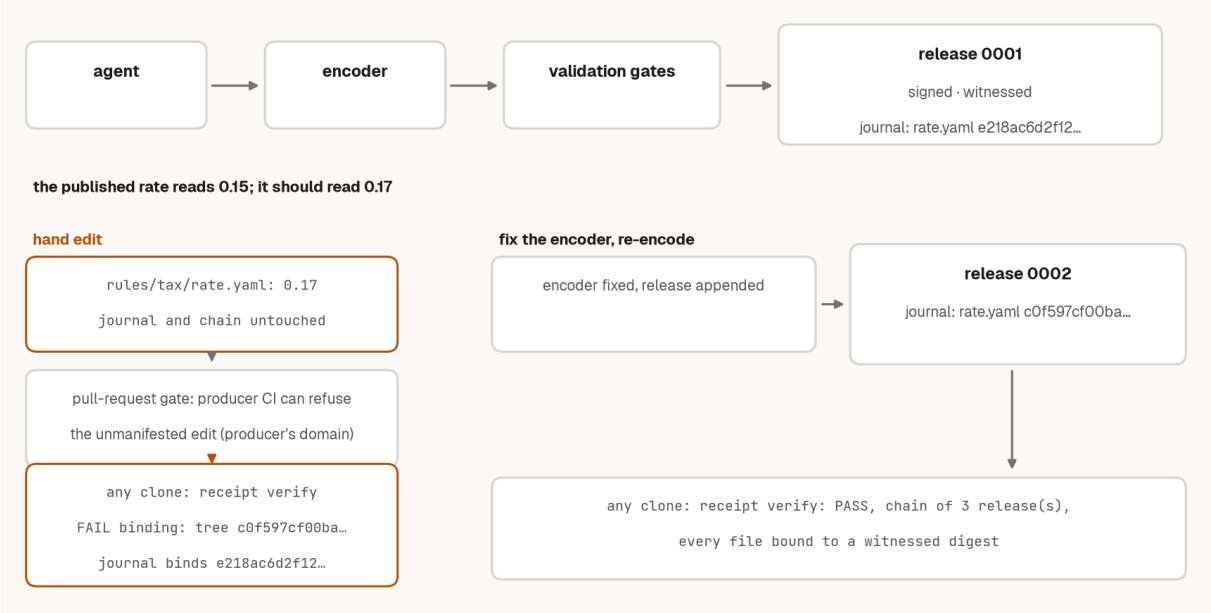

Figure 1: One correction, two paths. Both leave a rate of 0.17 in the same file. The pipeline path leaves a third release in the chain and a journal row that binds the new bytes, and the verdict passes. The hand edit leaves bytes that match no journal row; the producer's own pull-request gate can refuse it first, in the producer's domain, and the verdict refuses it in the binding pass, naming the file and both digests. Either verdict is anyone's, offline, from a clone.

The records that need this check are the ones agents produce faster than people can witness them. The Axiom Foundation's software agents encode statutes into executable rules, and a shared validation workflow refuses a change under the guarded rule roots that no signed manifest covers. A signed corpus, the release history behind it, and the gate declarations the producer recorded are the record set, and the receipt package checks them. The question descends from the audit-log literature, where secure logs protect past entries after compromise and make append-only consistency auditable (Schneier and Kelsey 1999; Crosby and Wallach 2009). This setting asks the consumer's version: what an outside verifier can conclude about a complete record corpus from a clone, against pins the producer cannot reach.

2 What a verdict establishes

Four inputs fix the verdict. The first is the tree object the named commit carries: the verifier resolves the commit once, authenticates its payload and its root tree, and rehashes every object it reads against the object name Git gave it. The second is a base tree, when the auditor names an ancestor commit; the verifier walks the candidate's parents to reach it, so the base must be an ancestor. The third is the specification, a Python module the auditor reviews and pins by digest: the loader hashes the source bytes once, compares the digest before compiling, and refuses to run mismatching code. The fourth is the trust material that specification pins: the producer's key fingerprint and, for each timestamp authority, the anchor file's name and digest, the policy identifier, and digests of the signer certificate and signer key. The anchor bytes themselves come from the tree; the anchor-set digest below binds them. No record byte comes from the checkout or the index. One more thing enters: the verifier's clock. A token more than 300 seconds ahead of it refuses, so a verdict holds for the clock it was rendered against.

The verifier resolves the candidate, enters the base when one was supplied and proves it an ancestor,

and optionally checks the whole object store, which refuses as custody. The passes then run in a fixed order and stop at the first failure: history when a base was supplied, then custody, binding and declaration completeness. A closing re-audit of the repository's control files and configuration follows; its failure voids custody, binding and declaration, keeps a history pass already recorded, and is reported as a custody refusal. A pass requires all three of custody, binding and declaration to have succeeded. An unexpected exception or a failure to render the result counts as a failure. The exit code carries the verdict: zero for a pass (and for help), one for a failed or aborted verification, two for a usage error. A specification that raises or exits on load refuses at the specification stage with the usage status, not the failure status.

Table 1 states what a pass establishes and what it does not.

Table 1: What a passing verdict is and is not evidence of.

A pass establishes	A pass does not establish
Chain. A contiguous, content-addressed sequence of release manifests from genesis to head, each signed by the pinned producer key over its exact bytes, each carrying one token from every configured timestamp authority.	That this chain is the only one the producer keeps. One clone cannot expose a split view.
Time. Each release existed no later than each authority's signed time, checked against the pinned root, policy, signer certificate and signer key without any network call.	When a release was created. Nothing bounds the signed time from above relative to the manifest's recorded creation time, so a chain regenerated, re-signed and re-witnessed before first contact passes.
Preservation. With a base, every object under the release root that the base held keeps its mode and its object name in the candidate.	Preservation of anything added after the base, or that the candidate is the newest history.
Binding. Within the configured content roots, every file with a configured suffix is listed in the journal and hashes to its listed digest; an unlisted file, a missing file, a rewritten byte, a symlink where a file was recorded, and two sibling names anywhere in the tree that collide under ASCII case folding each refuse; under any of the five pinned chain paths, or in a directory above one, the colliding pair refuses in custody first.	How the bytes were produced, or that they read the statute correctly.
Declarations. Every gate the specification requires is declared in the journal, whose bytes hash to the value the head manifest records.	That any gate ran or passed. The verifier does not rerun them.
Identity. The commit and tree that were judged, the specification's path and digest, the head manifest's digest, the resolved base, and a digest of the anchor bytes custody consumed.	That any checkout equals the judged tree, or that the anchors the tree carries are trusted, unless the auditor pinned them.

The right-hand column shrinks over time through one recipe. At first contact the auditor records the head manifest's digest and the commit, and pins the specification's digest and the anchor-set digest the verdict printed. Every later run supplies that commit as the base, expects the current candidate by name, and re-supplies the specification and anchor-set digests it pinned. From then on, a release object present at first contact cannot be rewritten or deleted without a refusal. A swapped key or authority cannot pass as the pinned one, and the specification cannot change under the auditor. The

recipe touches the time and preservation rows, and two limits stay even there. The specification's digest names the source bytes the loader read, not the files that source may read in turn, so an auditor reviews it as code once. And the recipe closes the regeneration window from first contact forward, never backward.

The anchor discipline has one more step. The anchor-set digest is computed from the anchor files the verified tree carries and compared with the auditor's expectation before any cryptographic call, and the bytes custody consumed must match that digest afterward. Without a pinned expectation the verdict still passes, and says in its scope that trust in the anchors and in the specification code was not established.

Three callers reach the directory verifier through a private materialization of the selected tree: the composed command, the append gate and the base-chain verifier. A caller who runs the directory verifier on a directory of their own gets a verdict about that directory as it was read, with the exposure to a concurrent writer that implies.

3 Why the shape follows from the threat

The producer controls the repository, its CI, its key settings and its monitoring. The verifier controls one commit of its own. Each design rule keeps the verdict on the verifier's side of that line.

Trust anchors live in the verifier's committed code. Key fingerprints, timestamp-authority anchor names and digests, content roots, suffixes and required declarations sit in a specification the auditor commits and reviews. The package ships no trust anchor of its own and reads none from the environment. The chain and corpus specification objects check their trust values at construction: a fingerprint that is not a digest, an empty authority set, or a malformed release path refuses before any verification runs. The producer-key and append specifications carry no validator, and the key specification checks the scheme label alone. The keyring specification checks a nonempty key set, distinct identifiers and fingerprints, and a threshold between one and the key count, which a NaN threshold passed in 0.6.0, and nothing downstream closed that gap: the threshold check compared the satisfied count against the threshold and returned. 0.6.1 refuses any threshold that is not an integer at construction. The defaults that exist are mechanics, not trust. The candidate is `HEAD` unless named, names are held to the portable repertoire, and a receipt may precede its manifest's recorded creation time by at most 300 seconds.

Rotation is loud. The release-chain lane pins one producer key by its fingerprint, and rotating it is an edit to that pin. The keyring API the encoder calls for its own signatures names current keys, a threshold and retired keys. The threshold call says whether retired keys may vouch: excluded, any retired key or signature presented refuses outright; allowed, they count toward the same threshold. The any-generation call, for a threshold of one, tries the current keys and then the retired ones; 0.6.1 lets the caller exclude the retired ones, as the threshold call already could. A malformed or mispinned public key is fatal in either call. In the threshold call, a missing signature or a missing public key is reported as absent, and a malformed signature whose key is present as failed; neither votes, so the check still passes when enough other keys satisfy the threshold. The any-generation call refuses a missing key or a malformed signature outright. There is no time window in which either generation silently works. A timestamp authority rotates in one of two ways. In the release-chain lane the specification pins each authority's root, policy, signer certificate and signer key, and rotation is an edit to those pins. In the witness lane an authority rotates through a new immutable trust bundle with its own pinned identities, activated by the caller.

Verification runs offline on commodity tools. A full SHA-1 clone, Python 3.11 with the cryptography library at 42 or later, OpenSSL 3 and Git 2.36 suffice for the chain, time, signature and binding checks. A shallow, grafted, partial or SHA-256 clone refuses in custody. The verifier queries no authority. The workflow-attestation check is the one exception. It asks GitHub, through the GitHub CLI, whether the

attestation service accepted the canonical subject for a commit under a workflow named by path and ref (Newman et al. 2022). That check trusts GitHub as a third party, pins the workflow by name rather than by its bytes, and takes the CLI's exit status as its answer. The package supplies the pieces of a history sweep: the list of commits touching the protected tree, the enforcement epoch, the in-scope test and the repository slug; the caller composes them.

Specified gaps refuse. A missing token, a malformed row, an unknown key, a time out of range, a path that escapes its root, a name outside the repertoire: each refuses with a typed error. The verification libraries write nothing to stdout or stderr; the command renders acceptance and refusal, and exit codes carry the verdict. On the routes the differential harnesses cover (Section 4), the refusal texts are the production verifiers' own, and a consumer's own differential harness matches on them. Two base-ref routes match because the harness maps the package's snapshot diagnostic onto the upstream wording, and the tree-subject checks add refusals upstream never emitted.

4 Evidence

Two kinds of evidence are checkable: the verdicts the released package renders on six scenarios over its own signed fixture, and the harnesses that hold the verifier to the production code it came from.

The package ships a corpus fixture that builds a fresh Git repository, an Ed25519 producer key, and two local RFC 3161 authorities with their own roots. It publishes two releases of a corpus of three content files and one attested file. The auditor's specification lives outside the corpus, so a mutation of the corpus cannot change what the auditor expects. Four scenarios commit one mutation of a private clone; one mutates nothing, and one publishes a separate corpus from its own genesis. Each runs three ways. At first contact the command names `--commit HEAD`. With a base but no candidate pin it refuses as a usage error, because a base requires an expected commit. Pinned, it names `--commit`, `--expect-commit` and `--base-ref` at that scenario's genesis. Table 2 gives the verdicts.

Table 2: The six scenarios and their verdicts under receipt 0.6.1, from the package's own fixture. The base-only runs, the ones this table omits, refuse as usage errors before any pass runs.

Scenario	Change	First contact	Pinned to its genesis
pristine	none	PASS	PASS
hand edit	`rules/tax/rate.yaml` rewritten to 0.17 and committed; journal, manifests, signatures and tokens untouched	FAIL, binding: `content file 'rules/tax/rate.yaml' does not match its witnessed digest: …`	FAIL, binding
re-encode	the same change appended as a third signed, witnessed release with its journal row	PASS	PASS
swapped key	`releases/anchors/producer-ed25519.pub` replaced by another key	FAIL, custody: `producer public-key SPKI is not code-pinned: …`	FAIL, history: existing release file bytes changed relative to the base

Scenario	Change	First contact	Pinned to its genesis
regenerated	every manifest, signature and token rebuilt with the original key and authorities; content and journal unchanged	PASS	FAIL, history: `existing release file was deleted relative to` the base, naming the genesis token
dropped gate	published from its own genesis without ever declaring `rulespec/compile`	FAIL, declaration: `the witnessed journal does not declare a gate the pinned spec requires: …`	FAIL, declaration

The hand edit and the re-encode leave the same rate in the same file, and only the journal tells them apart. The regenerated chain passes at first contact: custody for two releases, binding for the files, declarations for the gates, all genuine, all dated to the regeneration. It refuses only against a base, and the refusal names the first object the base held that the candidate lacks. The dropped gate passes custody and binding and refuses on the declaration the specification requires. The page at axiom.org/receipt shows these eighteen runs, and the capture recipe in the site's repository regenerates them from the fixture.

The verifier has its own checks. The canonical-JSON module is PolicyEngine's ledger file byte for byte (PolicyEngine, n.d.); a test pins its SHA-256. The release-chain and append-gate surfaces came from that ledger, and the witness and attestation surfaces from a forecasting repository's record chain (Ghenis, n.d.), behind differential harnesses in the tradition of differential testing (McKeeman 1998). Each harness pins the upstream source files by SHA-256 and runs the unmodified upstream verifier as an oracle on real fixtures. The package must reproduce the oracle's verdicts, accepted and refused alike, to the byte after two disclosed normalizations: surrounding whitespace, and the volatile identifiers OpenSSL prints on error lines. At v0.6.1 the four harnesses collect 108 cases: 43 release-chain, 28 append-gate, 17 witness and 20 attestation. Of those, 86 compare the package's verdict against the oracle's; the other 22 authenticate the oracle sources, exercise the commit subject where the directory oracle has no counterpart, or require the two to diverge. The witness fixture carries 53 snapshots, 52 available witnesses and real timestamp tokens; the attestation harness compares command construction against a local stub and claims no live-service equivalence. The 0.6 move from checkout to tree subject added cases the oracle cannot share. One rewrites a committed ledger row in the checkout only; the directory oracle refuses while the package accepts the unchanged commit. The signing module's retired-key semantics match the release-key rotation procedure the statute corpus repository adopted (the Axiom Foundation, n.d.-b); they entered with direct tests and no oracle. The package's remaining 1,471 tests pass at the tag.

5 Adoption

On 2026-09-10, two repositories pin `receipt==0.6.1` on their default branch and call it from their verification paths, a third pins it on a branch that has not reached its default branch, and a fourth shows its output (Table 3).

Table 3: Adoption at one date.

Repository	Pin	Surface called
TheAxiomFoundation/axiom-encode	`receipt==0.6.1`	threshold keyring check on apply manifests, through `receipt.sign`
ThesisInstitute/thesis	`receipt==0.6.1` (custody extra)	signature primitives in the snapshot signer and the chain verifier; producer signing armed
PolicyEngine/chronicle, branch `codex/thesis-ledger-facts`	`receipt==0.6.1`	release-chain and append-gate shims over `receipt.release_chain` and `receipt.append_gate`, whose shim headers require a fresh byte-equivalence proof against pinned legacy oracles before each receipt upgrade
TheAxiomFoundation/axiom.org	none in a dependency manifest; the capture recipe installs 0.6.1	the receipt page shows eighteen fixture runs, byte-identical to the maintainer's regeneration at v0.6.1 after masking the run-minted object ids, digests, times and temporary paths

The New Zealand pilot (the Axiom Foundation, n.d.-a) ships, in its open pull request, an auditor guide, a specification and a witnessed journal. The guide's recipe is written for `receipt>=0.5` and its worked reproduction pins 0.5.0. Its base-ref recipe predates the 0.6 commit contract, which requires an expected commit alongside a base. The specification carries the custody key, two timestamp-authority anchors, one content root and six required gate identifiers. It commits a witnessed journal of 104 rows: 80 content files, 7 attested context paths and 17 gate declarations, every digest matching its committed blob.

A producer proposes a specification by shipping it in the repository. An auditor who verifies against it as found establishes consistency with a policy the producer shipped. Independent custody begins when the auditor reads the specification once, pins its digest and the anchor-set digest in the auditor's own records, and supplies both on every later run. What a gate row does and does not establish is in Table 1.

A package pin fixes the verifier's bytes; custody comes from the trust pins the auditor supplies. What an encoder run produced is a separate claim, and the foundation's corpora make it separately. The encoder signs an apply manifest binding source, tool, generated-output digest and applied-file digests, and the shared validation workflow refuses a change to a guarded rule file that no manifest covers.[1] That check runs in the producer's trust domain, and it supplies what the custody verdict cannot (Table 1).

[1] rulespec-us at `d58cc0ce` calls the shared workflow at `7a1eb243` with the generated-file guard on and `guard-programs-root` off (134 files under `programs/` unguarded); manifest schema `axiom-encode/applied-rulespec/v5`, fields at `src/axiom_encode/cli.py:503` of `29b30fb7`. Observed 2026-09-05.

6 Related systems

receipt assembles per-consumer trust policy in committed code, multi-anchor timestamp composition, record-level history checks and closed-world corpus binding under one fail-closed verdict that names its non-conclusions. Each neighbor solves an adjacent problem (Table 4).

Table 4: Neighboring systems by trust root, setting, time and history. The receipt row describes the custody and binding path; its workflow-attestation check is the online exception (Section 3).

system	trust root	verification setting	witnessed time	history and binding
receipt	consumer's committed code	offline, from a clone, no service	multi-anchor RFC 3161, pinned roots and signers	append-only release chain; closed-world corpus binding; gates stay declarations
OpenTimestamps	blockchain consensus	proof against the chain	blockchain anchoring	existence proofs per document
CT / Go checksum DB	log operators; client-pinned key (Go)	log service plus monitors; gossip outside the RFC (CT), designed in (Go)	SCT timestamps (CT); not specified in the design proposal (Go)	public append-only log with consistency proofs
SCITT	transparency service	online registration; receipts verify offline with the service's key	the service's signed endorsement of its log state	registry of signed statements
TUF	client-side root metadata, thresholds	updater fetching current metadata	expiry-based freshness	current snapshot only
Sigstore (keyless)	OIDC identity federation	bundles verify offline; log for inclusion	log entry timestamp, or RFC 3161	signature transparency, per artifact
Signed Git commits	locally configured signers	offline, from a clone	none	commit-object authentication over the DAG
in-toto	owner-signed layout	offline against the layout	none	step materials and products; steps attested by authorized functionaries
Agent event receipts	in-process, daemon or HSM keys (Obsigna); an external witness (halo-record)	offline, against a saved chain or receipt database; Obsigna resolves the issuer key by DID	optional RFC 3161 timestamps	per-event chains

Timestamping. Haber and Stornetta (Haber and Stornetta 1991) make back-dating and forward-dating infeasible even against the service; RFC 3161 (Adams et al. 2001), as updated by RFC 5816 (Santesson and Pope 2010), defines the token receipt builds on, and the package composes saved

tokens from the consumer's authorities under pinned roots, policies and signers. OpenTimestamps (Todd 2016) anchors existence in a blockchain instead, a different trust root for witnessed time. Evidence Record Syntax (Gondrom et al. 2007; Blazic et al. 2011) keeps such evidence valid across hash and certificate obsolescence; receipt has no renewal protocol (Section 7).

Transparency logs. Certificate Transparency (Laurie et al. 2021) and the Go checksum database (Cox and Valsorda 2019) put records in public append-only logs with consistency proofs and third-party monitors. A log that shows different views to different clients defeats a lone client in both; CT's own RFC leaves gossip out of scope, and the Go design builds it in. A single clone has the same gap and no log to gossip about. SCITT (Birkholz et al. 2026; Microsoft 2026) registers signed statements with a transparency service that returns signed inclusion proofs, which its architecture calls receipts. Registration puts a service in the loop, and a receipt then verifies offline against the service's key; this package checks a locally held history with no service at either stage. PeerReview (Haeberlen et al. 2007) holds nodes accountable by checking each other's tamper-evident logs of their own actions; receipt never judges the agent's behavior, only the custody of what it wrote.

Update and signing systems. The Update Framework (Cappos et al. 2026; Samuel et al. 2010) roots trust client-side with thresholds and offline root keys, as an updater protocol for fetching current metadata; receipt checks the history of a clone already obtained. Sigstore's keyless flow (Newman et al. 2022) binds signatures to OIDC identities through short-lived certificates and a transparency log. Its bundles carry what a verifier needs offline (Sigstore project 2026). receipt's signature and witness surfaces use no identity federation and no log, while its workflow-provenance check rides GitHub's Sigstore-backed attestation service. Signed Git commits (*gitformat-signature: Git Cryptographic Signature Formats* 2023) authenticate commit objects under locally configured signers; receipt adds the release chain, witnessed time, historical bytes and modes, and corpus binding.

Process attestation. in-toto (Torres-Arias et al. 2019) verifies that supply-chain steps matched an owner-signed layout and records their materials and products; receipt verifies custody of the records and treats gates strictly as declarations it never reruns. SLSA (SLSA Community 2025) specifies leveled requirements rather than one verifier. Reproducible builds (Reproducible Builds project, n.d.) compare rebuilt artifacts byte for byte; receipt's harnesses compare verifier verdicts on shared fixtures, a related and distinct discipline.

Agent-record neighbors, and the word "receipt". Obsigna's agent receipts (Jongerius and Obsigna contributors 2026) sign and chain runtime actions under a key held in process, by a signing daemon, or in a hardware module, with optional RFC 3161 timestamps. halo-record (Kuan 2026) holds no signing key at all; it hash-chains the agent's event stream to checkpoints an external witness keeps, with optional RFC 3161 timestamps on the checkpoint. Basu's tool receipts (Basu 2026) authenticate one tool observation inside a live runtime, and C2PA's repository receipt (Coalition for Content Provenance and Authenticity 2026) binds asset ingestion under its X.509 ecosystem. Each authenticates machine-produced evidence at the event or asset level, in or beside a runtime; this package verifies the durable repository history those events land in, under trust policy each consumer commits. Agent provenance schemas (Souza et al. 2025), research-object packaging (Soiland-Reyes et al. 2022) and proofs of agent computation (Wang et al. 2026) describe, package and prove the computation; each combines with custody of the resulting records.

7 Limits

A pass establishes custody, not truth (Table 1). Freshness needs an expectation from outside the clone, and comparing head digests out of band is how two auditors learn that their views diverge. The base recipe in Section 2 reaches forward from first contact only. Later content may still be superseded through the journal, by design: the base protects release objects, not the current binding.

The consumer chooses the authorities and the verifier enforces the configured set; it cannot show

that authorities fail independently. The release-chain lane requires one token from every configured authority for every release, so a release missing one authority's token cannot pass until the consumer revises policy. The witness lane accepts a declared outage with a reason as a declaration and can verify a record with zero tokens. A consumer who needs a floor reads the returned status and token list and enforces it in their own code; the specification has no field for it. Certificates are validated as of the token's signed time through OpenSSL without revocation checking, and no renewal protocol carries aging evidence forward (Gondrom et al. 2007).

The differential harnesses cover finite batteries, and each test file carries its pins, commands and exclusions, which is also its rerun recipe. The package inherits the assumptions of its primitives: signature schemes and hash functions unbroken, authorities honest about time. Ordinary object reads rehash with plain SHA-1; optional store-wide verification hands collision detection to Git's own detector and requires Git 2.50. Key compromise before rotation defeats the signature layer for that generation.

The package verifies and supplies signing primitives; it does not produce manifests, witnesses or journals, which come from the producers' own tooling. The package supplies no producer-side reference implementation and no inert specification format. A contributed producer-side evidence-record type, non-authorizing by design, is open (Storey 2026). The package is Python; producers on other stacks interoperate at the artifact layer, in canonical JSON, manifests, tokens and signatures.

8 Availability

`receipt` is on PyPI (MIT), with source, port diffs, review records and differential harnesses at github.com/TheAxiomFoundation/receipt, the package page at axiom.org/receipt, and the API reference at axiom.org/receipt/api.

Acknowledgements

Nathan Storey's review and his contributed evidence-record proposal shaped the account of the producer side in Section 7.

References

Adams, C., P. Cain, D. Pinkas, and R. Zuccherato. 2001. *Internet X.509 Public Key Infrastructure Time-Stamp Protocol (TSP)*. No. 3161. Request for Comments. RFC 3161; RFC Editor. https://doi.org/10.17487/RFC3161.

Basu, Abhinaba. 2026. *Tool Receipts, Not Zero-Knowledge Proofs: Practical Hallucination Detection for AI Agents*. https://doi.org/10.48550/arXiv.2603.10060.

Birkholz, Henk, Antoine Delignat-Lavaud, Cédric Fournet, Yogesh Deshpande, and Steve Lasker. 2026. *An Architecture for Trustworthy and Transparent Digital Supply Chains*. No. 9943. Request for Comments. RFC 9943; RFC Editor. https://doi.org/10.17487/RFC9943.

Blazic, A. Jerman, S. Saljic, and T. Gondrom. 2011. *Extensible Markup Language Evidence Record Syntax (XMLERS)*. Request for Comments No. 6283. RFC Editor. https://doi.org/10.17487/RFC6283.

Cappos, Justin, Trishank Karthik Kuppusamy, Joshua Lock, Marina Moore, and Lukas Pühringer, eds. 2026. *The Update Framework Specification*. Specification 1.0.36. The Update Framework. https://theupdateframework.github.io/specification/v1.0.36/.

Coalition for Content Provenance and Authenticity. 2026. *Content Credentials: C2PA Technical Specification*. Technical Specification 2.4. Coalition for Content Provenance; Authenticity. https://spec.c2pa.org/specifications/specifications/2.4/specs/C2PA_Specification.html.

Cox, Russ, and Filippo Valsorda. 2019. *Proposal: Secure the Public Go Module Ecosystem*. The Go Project. https://go.googlesource.com/proposal/+/master/design/25530-sumdb.md.

Crosby, Scott A., and Dan S. Wallach. 2009. "Efficient Data Structures for Tamper-Evident Logging." *18th USENIX Security Symposium (USENIX Security 09)* (Montreal, Quebec), August, 317–34. https://www.usenix.org/conference/usenixsecurity09/technical-sessions/presentation/efficient-data-structures-tamper-evident.

Ghenis, Max. n.d. *Brier: Pre-Registered Forecast Records*. Https://github.com/MaxGhenis/brier.

gitformat-signature: Git Cryptographic Signature Formats. 2023. The Git Project. https://git-scm.com/docs/gitformat-signature.

Gondrom, T., R. Brandner, and U. Pordesch. 2007. *Evidence Record Syntax (ERS)*. Request for Comments No. 4998. RFC Editor. https://doi.org/10.17487/RFC4998.

Haber, Stuart, and W. Scott Stornetta. 1991. "How to Time-Stamp a Digital Document." *Journal of Cryptology* 3 (2): 99–111. https://doi.org/10.1007/BF00196791.

Haeberlen, Andreas, Petr Kouznetsov, and Peter Druschel. 2007. "PeerReview: Practical Accountability for Distributed Systems." *Proceedings of the Twenty-First ACM SIGOPS Symposium on Operating Systems Principles* (New York, NY, USA), SOSP '07, October, 175–88. https://doi.org/10.1145/1294261.1294279.

Jongerius, Otto, and the Obsigna contributors. 2026. *Obsigna and the Agent Receipts Protocol*. Protocol specification draft v0.5.0 (2026-06-09); implementation tag obsigna/v0.31.0. https://github.com/agent-receipts/obsigna.

Kuan, Brian. 2026. *Halo-Record: Tamper-Evident Runtime Records for AI Agents*. Software, version 0.2.42 on PyPI (2026-09-02); repository tag v0.2.32. https://github.com/bkuan001/halo-record.

Laurie, B., E. Messeri, and R. Stradling. 2021. *Certificate Transparency Version 2.0*. No. 9162. Request for Comments. RFC 9162; RFC Editor. https://doi.org/10.17487/RFC9162.

McKeeman, William M. 1998. "Differential Testing for Software." *Digital Technical Journal* 10 (1): 100–107. https://www.cs.tufts.edu/comp/150FP/archive/bill-mckeeman/DifferentailTesting.pdf.

Microsoft. 2026. *Pyscitt and the SCITT CCF Ledger*. Software, pyscitt version 0.14.2 (2026-08-24). https://github.com/microsoft/scitt-ccf-ledger.

Newman, Zachary, John Speed Meyers, and Santiago Torres-Arias. 2022. "Sigstore: Software Signing for Everybody." *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security*, CCS '22, November, 2353–67. https://doi.org/10.1145/3548606.3560596.

PolicyEngine. n.d. *Chronicle (Formerly Ledger): The Observation Ledger and Release Chain*. Https://github.com/PolicyEngine/chronicle.

Reproducible Builds project. n.d. *Definitions: When Is a Build Reproducible?* Reproducible-builds.org. https://reproducible-builds.org/docs/definition/.

Samuel, Justin, Nick Mathewson, Justin Cappos, and Roger Dingledine. 2010. "Survivable Key Compromise in Software Update Systems." *Proceedings of the 17th ACM Conference on Computer and Communications Security*, CCS '10, October, 61–72. https://doi.org/10.1145/1866307.1866315.

Santesson, Stefan, and Nick Pope. 2010. *ESSCertIDv2 Update for RFC 3161*. No. 5816. Request for Comments. RFC 5816; RFC Editor. https://doi.org/10.17487/RFC5816.

Schneier, Bruce, and John Kelsey. 1999. "Secure Audit Logs to Support Computer Forensics." *ACM Transactions on Information and System Security* 2 (2): 159–76. https://doi.org/10.1145/317087.317089.

Sigstore project. 2026. *Sigstore Bundle Format*. Project documentation. https://docs.sigstore.dev/about/bundle/.

SLSA Community. 2025. *SLSA Specification*. Approved Specification 1.2. Open Source Security Foundation. https://slsa.dev/spec/v1.2/.

Soiland-Reyes, Stian, Peter Sefton, Mercè Crosas, et al. 2022. "Packaging Research Artefacts with RO-Crate." *Data Science* 5 (2): 97–138. https://doi.org/10.3233/DS-210053.

Souza, Renan, Amal Gueroudji, Stephen DeWitt, et al. 2025. "PROV-AGENT: Unified Provenance for Tracking AI Agent Interactions in Agentic Workflows." *2025 IEEE International Conference on eScience (eScience)*, 467–73. https://doi.org/10.1109/eScience65000.2025.00093.

Storey, Nathan. 2026. *Add Non-Authorizing Evidence Records (Receipt.evidence)*. Https://github.com/TheAxiomFoundation/receipt/pull/53.

the Axiom Foundation. n.d.-a. *Publish a Witnessed Corpus an Outside Auditor Can Verify Offline*. Https://github.com/TheAxiomFoundation/rulespec-nz/pull/104.

the Axiom Foundation. n.d.-b. *Support Release Signing Key Rotation*. Https://github.com/TheAxiomFoundation/axiom-corpus/pull/431.

Todd, Peter. 2016. *OpenTimestamps: Scalable, Trust-Minimized, Distributed Timestamping with Bitcoin*. https://petertodd.org/2016/opentimestamps-announcement.

Torres-Arias, Santiago, Hammad Afzali, Trishank Karthik Kuppusamy, Reza Curtmola, and Justin Cappos. 2019. "In-Toto: Providing Farm-to-Table Guarantees for Bits and Bytes." *28th USENIX Security Symposium (USENIX Security 19)* (Santa Clara, CA), August, 1393–410. https://www.usenix.org/conference/usenixsecurity19/presentation/torres-arias.

Wang, Lizheng, Hancheng Lou, Chongrong Li, Yu Yu, and Yuncong Hu. 2026. *zkAgent: Verifiable LLM Agent Execution via One-Shot Transcript Proofs*. Cryptology ePrint Archive, Paper 2026/199. https://eprint.iacr.org/2026/199.