Notarizing 59 Repositories with Dual RFC 3161 Timestamps: An Offline-Verifiable Supply-Chain Record and Its Limits

yxpil

Abstract

We present and measure a supply-chain record that binds 59 source repositories, 6,713 file entries in total, to two independently operated RFC 3161 timestamp authorities, and that a third party can check without network access and without trusting the tool that produced it. Each repository is described by a canonical, byte-sorted manifest; each timestamp token commits to the SHA-256 of one manifest's bytes; a master index records every manifest digest together with both tokens and is itself timestamped by both authorities. We audit all 59 committed records: every manifest digest matches its index row, every recorded commit matches the manifest header, and all 120 tokens verify offline against two committed trust anchors, with the token imprint equal to the SHA-256 of the signed file in all 120 cases. Negative controls confirm that a mutated manifest, a manifest belonging to another repository, and a wrong trust anchor are each rejected. The claim is deliberately narrow. The record shows that particular bytes existed before a time attested by two commercial authorities; it does not establish authorship, does not bind file contents unless the corresponding git objects are available, cannot show that either authority was honest at signing time, and does not conceal the names of three private repositories that it indexes. We state these limits, and the quantities our audit did not measure, explicitly.

1 Introduction

A git commit identifier is a content address, but the statement "this commit existed on 13 October 2026" is not carried by the identifier. It is carried by the hosting platform's database, or by nothing at all: git accepts arbitrary author and committer dates, so a commit created today can claim any past date. An author who wants a record of existence that outlives their account, survives a repository rewrite, and can be checked by a stranger has to put the claim somewhere outside the platform.

Trusted timestamping is the standard answer. An RFC 3161 timestamp authority (TSA) signs a structure that contains a hash of the submitted data and the time of signing [Adams et al., 2001]. The signed token can later be verified without contacting the TSA, provided the verifier holds the authority's certificate chain. This paper describes, and then audits, a record built on that primitive: 59 repositories, described by canonical manifests, with every manifest digest timestamped by two commercial authorities, Sectigo and DigiCert, and a master index that is itself timestamped by both.

The contribution is not the cryptography. RFC 3161 [Adams et al., 2001], CMS [Housley, 2009], and hash trees [Merkle, 1989] are prior art, and the record uses each of them unchanged. The contribution is (i) the design of the record: a canonical manifest format, a digest-of-manifest discipline, dual independent timestamping, an index that chains the individual records, and a verification procedure that needs no network and no trust in the producing tool beyond re-implementing the manifest formatting rules; and (ii) a measured audit of the 59 records that were actually produced, including the parts that do not verify, the parts that cannot be verified offline, and the quantities that were not measured.

What the record is for. The intended claim is narrow and, we argue, defensible: *the bytes of manifest M existed no later than time T, as attested by two independent commercial authorities, and anyone holding M can check this offline*. For a repository owner this is a cheap form of archival evidence: no signing key tied to a legal identity, no blockchain, no continuous operation, and a verification path that a third party can execute with `openssl` alone. For a reader of this paper the interesting question is not whether the mechanism works in principle—it does, by construction—but what a real corpus of such records looks like when it is measured, and where the claim stops.

Scope and evidence. Every quantitative statement below is traceable to `scripts/analyze.py`, whose JSON output (`scripts/notarization_stats.json`) accompanies the paper, either directly or by arithmetic on values it records; the figures are generated from the same file by `scripts/make_figures.py`. Where the audit did not measure a quantity we say so rather than estimate it. The most important such omission is verification time, which was not measured at all (Section 6).

2 Background

2.1 Trusted timestamping

Haber and Stornetta introduced the problem of proving that a document existed at a given time without revealing it [Haber and Stornetta, 1991]. RFC 3161 instantiates the idea with a time-stamping protocol over CMS [Adams et al., 2001, Housley, 2009]: the client sends a *timestamp query* containing the hash of the data, the TSA returns a *timestamp token* whose

signed content (`TSTInfo`) repeats that hash as a message imprint, together with the time of signing, a serial number, a policy identifier, and optionally a nonce. Verification checks the signature chain to a trust anchor, the extended key usage of the signer certificate, and the equality between the imprint and the hash of the data at hand. No network access is required once the anchors and any intermediate certificates are available; this property is what makes the record auditable after the fact.

A token asserts existence before a time. It does not assert who submitted the data: the imprint is a hash, and the token carries no submitter identity. It also does not assert that the data is correct or complete—only that the hashed bytes existed. Both properties are inherited by our record and are restated in the threat model and in the limitations.

2.2 Transparency logs and witnesses

Merkle's authentication trees [Merkle, 1989] underpin tamper-evident logs: Crosby and Wallach described efficient append-only history with proofs of inclusion and consistency [Crosby and Wallach, 2009], and Certificate Transparency applies the construction to certificates [Laurie et al., 2013]. Sigstore adds a public transparency log (Rekor) and a signing workflow aimed at open-source software [Newman et al., 2022]. The security property these systems add over a single timestamp is *public witness*: many parties hold the log's head, so equivocation is detectable.

Our record has no log. Each record is witnessed by two certificate authorities, not by an append-only public structure, and the project's notes record an attempt to anchor digests in Rekor that failed because the log rejected the submission kind the tooling used [Newman et al., 2022]. We treat this as a real gap rather than a design choice: two witnesses raise the cost of collusion, but they do not make the record publicly auditable.

2.3 Reproducible builds and supply-chain metadata

Reproducible builds let an independent party rebuild an artifact and compare hashes [Lamb and Zacchiroli, 2022]. in-toto binds the steps of a build to signed attestations [Torres-Arias et al., 2019], TUF protects update metadata against key compromise [Samuel et al., 2010], and SLSA defines levels for provenance completeness [Open Source Security Foundation, 2023]. All three bind *what* was built and *how*; they assume a key infrastructure and a policy. Our record binds *that a listing existed* at a time. It is a weaker claim, but it requires no signing identity, no verifier policy, and no network, and it composes with those systems: a manifest digest could be attested by in-toto or published in a transparency log without changing anything in this design.

2.4 Git's object model

A git tree entry names a path, a blob object identifier, and a mode. The blob identifier is the SHA-1 hash of the object header and the file content, so it commits to the content without containing it. A listing of `path`, `blob id`, and `size` therefore commits to the file contents indirectly: anyone with the repository can recompute the identifiers and compare. The record described here stores exactly such a listing; it does not store file bytes.

3 Threat Model

Claim under audit. For each repository r: *the byte string M_r (the manifest) existed no later than the generation time recorded in token T_r, and M_r describes the file set of r at commit C_r.*

Adversaries considered. (A1) The hosting platform, or an attacker who controls the owner's account, rewrites or deletes repository history and the owner's assertions along with it. (A2) The owner later wants to backdate work, or to deny having published something. (A3) A verifier who does not trust the owner's scripts and wants to check the record with independent tools. (A4) One timestamp authority is compromised, coerced, or simply wrong about the time. (A5) An on-path attacker interferes with the timestamp request.

What the mechanisms buy. Against A1 and A2, the tokens are signed by third parties and are stored with the record; changing a manifest byte breaks the imprint equality, and no one other than the TSA can issue a token that carries the recorded generation time with a different imprint. Against A3, verification of an existing manifest requires only `openssl`, the manifest bytes, the two tokens, and the committed anchors; it does not execute project code. Against A4, the record uses two operators with different roots and different timestamp policies, so a single dishonest or compromised authority is not sufficient to sustain a false claim—though two colluding authorities would be. Against A5, the token is signed, so an attacker cannot forge one; the worst case is replay of an older valid token, which the client rejects at mint time through a freshness window (Section 4).

What is explicitly not claimed. The record does not bind the manifest to a person (the tokens carry no submitter identity); does not prove that the files were correct, lawful, or original; does not prove that the hosting API returned an honest tree when the manifest was generated; does not prove that the TSA's clock was accurate; and does not, by itself, let a verifier conclude anything about file *contents* without the git objects. The last point is the one most easily overstated, and Section 6 measures the record's actual coverage: paths, blob identifiers, and sizes.

4 Design

4.1 Canonical manifest

A manifest is a UTF-8, LF-terminated byte string with no BOM. It begins with a format tag and four header lines, followed by a marker and one line per file entry, sorted by the raw bytes of the whole line:

```
QODER-NOTARIZE/1
repo=yxpil/<name>
ref=refs/heads/<branch>
commit=<40-hex commit>
files=<entry count>
--- BEGIN FILE LIST (...) ---
<path>\t<blob-sha>\t<size>
```

The rules that matter for third-party reproduction are: one line per blob (directories and submodules excluded); fields separated by a single tab; the size in bytes; the sort applied to the whole line, which for valid UTF-8 is equivalent to sorting by path and, for ASCII paths, to `LC_ALL=C sort`; a terminating newline on every line, including the last; and no normalisation of any kind on replay. The header repeats the entry count so that a truncated file is detectable without parsing the body.

Byte ordering versus code-point ordering is a real subtlety when paths are not valid UTF-8. The audit measured zero such entries in 6,713 rows (Section 6), so the distinction has no observable effect on this corpus; it remains a specification hazard for other corpora, and the verification tool reads paths as bytes for that reason.

4.2 Digest of the manifest, and an index of digests

The unit of timestamping is the SHA-256 digest of the manifest bytes [National Institute of Standards and Technology, 2015]. The index `index/index.tsv` has one row per repository with nine tab-separated fields: repository name, ref, commit, manifest SHA-256, private flag, and the generation time and serial number of each of the two tokens. The index is itself hashed into `index/index.sha256` and—this is the part that closes the chain—the index file is timestamped by both authorities as well. A verifier therefore has two levels: each manifest digest is bound to two tokens, and the table of digests is bound to two more tokens. Without the second level, the table that maps repository names to digests and token serial numbers would itself be unprotected: an attacker could replace one row with another repository's row, and every check would still pass for the substituted pair.

The ref column is informative only. The manifest commits to a commit identifier, not to a branch: a branch can move after the fact, and the record remains valid for the commit it names.

4.3 Dual timestamping

Each manifest is submitted twice, once to Sectigo over HTTPS and once to DigiCert over plain HTTP, with the signer chain requested in both cases (`openssl ts -query -cert`). The two authorities differ in operator, root, and policy: the audit records policy `1.3.6.1.4.1.6449.2.1.1` for Sectigo and `2.16.840.1.114412.7.1` for DigiCert, with distinct serial number spaces (60 distinct serials from each authority).

The plain-HTTP endpoint deserves a note. An on-path attacker can read and replay a request but cannot forge a response, because the token is signed; the residual attack is substitution of an older valid token for the same data, which is strictly weaker evidence. The client therefore parses the generation time out of the reply and rejects tokens older than 900 seconds (or more than 90 seconds in the future) at mint time. This is a minting-time control only; verification does not enforce freshness, for reasons discussed in Section 6.

Minting is idempotent. Before requesting a new token the client verifies the existing one against the current manifest; if it verifies, the token is kept and its generation time and serial are copied into the index. A corrected manifest therefore receives a new token while untouched manifests keep their original evidence. This matters for the integrity of the record: re-stamping everything, as a monthly job might, would replace old evidence with new evidence and weaken the historical claim.

4.4 Verification procedure

Algorithm 1 is the procedure the repository's verifier implements. Two points distinguish it from a naive "check the signature" script. First, the digest comparison is over exact bytes: the verifier never parses and re-serialises the manifest, because re-serialisation would hide whitespace and ordering defects. Second, every step fails closed: a missing artifact, an unparsable header, a digest mismatch, or an `openssl` failure all yield a non-zero exit status, so the record can be used as a gate in a workflow.

The final, optional step is what connects the record to the repository. The digest and token checks prove that the manifest bytes are the bytes that were timestamped; they do not prove that the manifest describes the repository, because the manifest was produced by our own code from a hosting API. Re-deriving the body from a clone closes that gap, and the formatting rules above are simple enough to re-implement independently. We did not write a second implementation, which is a limitation we return to in Section 7.

4.5 Worked example

Take `SM4Kit`, the repository used by the audit's negative controls. Its manifest is 1,318 bytes and lists 19 entries; its header names commit `008b8712806f92bc08606e82145cb3c080332685` on `refs/heads/main`; its SHA-256 is `f6a1dffc...246a91be` (abridged; the full value is in the record and in the audit JSON), and the audit's index-digest control records the same value for this repository. Sectigo generated a token at `13:28:50` UTC on 13 October 2026 and DigiCert at `13:47:19`; both are recorded in the index with their serial numbers, and both tokens verify against the manifest bytes with the anchors in `authorities/`. The audit's negative controls use this pair: flipping one byte in

Algorithm 1 Offline verification of one repository record. The digest, commit, and token checks need only `openssl` and standard tools; the last step is optional and needs a git clone.

Require: repository r; manifest M; tokens T_S, T_D; anchors A_S, A_D; chains C_S, C_D; index row I_r

1: $d \leftarrow \mathrm{SHA256}(M)$ over the exact bytes, no normalisation
2: **assert** $d = I_r.sha256$
3: **assert** the `commit=` header of M equals $I_r.commit$
4: **for** each $(T, A, C) \in \{(T_S, A_S, C_S), (T_D, A_D, C_D)\}$ **do**
5: **assert** `openssl ts -verify -data` M `-in` T `-CAfile` A `-untrusted` C prints `Verification: OK`
6: **end for**
7: **assert** both index tokens verify with `-data index.tsv`, and that SHA256(`index.tsv`) equals the recorded `index.sha256`
8: *optional:* re-derive the body from `git ls-tree -r -z` at the recorded commit and compare paths, blob identifiers, and sizes
9: **return** accept

Table 1: Components of the record. Line counts are the audit's totals over the five Python files (517 lines) and the single shell file (47 lines).

File	Role
`notary.py`	Builds manifests from the hosting Git Data API; refuses truncated trees; writes the staging table
`stamp.py`	Dual-CA client; verify-before-keep, verify-after-mint, 900 s freshness window, index anchoring
`build_index.py`	Master index, its SHA-256, and the human-readable `INDEX.md`; refuses incomplete coverage
`verify.sh`	Offline verifier: digest, commit, both tokens
`verify_git.py`	Re-derives rows from a clone with `git ls-tree -r -z`
`extract_chain.py`	One-off extraction of signer chains from a token (needs the `cryptography` package)
`re-stamp.yml`	Monthly workflow: rebuild, stamp, index, verify, commit

the middle of the manifest, or presenting the token with the manifest of another repository, causes `openssl` to fail with a message-imprint mismatch.

5 Implementation

Table 1 lists the components. The manifest builder (`notary.py`) walks the account's repositories, reads each default branch's commit, requests the recursive tree, and refuses to proceed if the tree is truncated—a failure mode the project hit in practice with a package mirror whose tree exceeds the API's entry limit, which is why that repository is excluded from the corpus. The builder sorts the entries before formatting, so the manifest is a pure function of the tree listing and the header values.

The timestamp client (`stamp.py`) is the only component that touches the network. It contains the two authority profiles (endpoint, anchor, chain), verifies every reply before writing it, parses the generation time, enforces the freshness window, and, when it mints, writes a text rendering of the token next to the DER file for human inspection. The rendering is not evidence—it is derived from the DER token—but it makes the record readable without `openssl`.

The index builder refuses to run when any staged repository lacks one of the two tokens, so a partial run cannot produce an index that looks complete. The verifier (`verify.sh`, 47 lines of shell) loops over the index, recomputes each digest with `sha256sum`, compares the commit header with the index, and runs `openssl ts -verify` twice per repository. It records any failure and continues the loop so that all failures are reported, then exits non-zero if any occurred.

The clone-based checker (`verify_git.py`) re-derives the body from `git ls-tree -r -z` at the recorded commit, compares paths and blob identifiers, and compares sizes when the blobs are present locally; it detects a blobless clone from the remote's partial-clone filter so that size checks are skipped rather than silently triggering a network fetch. This is the only component that can establish the manifest-to-repository link, and it requires the repository itself.

Not implemented. The manifest builder contains a function that submits a digest to the Sigstore Rekor log [Newman et al., 2022], but nothing in the pipeline calls it; it is a placeholder, not a feature, and the corpus contains no transparency-log entries. There is no certificate revocation check (no CRL, no OCSP), no independent implementation of the manifest rules, and no mirror of the record outside the repository that holds it. The workflow that re-stamps monthly is configured, but the record audited here is a single generation epoch, not a series of observations.

6 Evaluation

The audit reads the committed artifacts, recomputes every digest, re-verifies every token with `openssl` 3.5.7 [The OpenSSL Project, 2026] against the committed anchors, runs

Table 2: Corpus as measured. Entry counts and byte counts are over manifest files; the digest and commit matches compare each manifest with its index row.

Quantity	Measured
Repositories (index rows)	59
Manifest files parsed	59
File entries, total	6,713
Manifest bytes, total	540,085
Entries per repository: min / median / mean / max	3 / 36 / 113.78 / 1,646
Manifest bytes per repository: min / max	351 / 149,178
Manifests whose declared count equals parsed rows	59 of 59
Manifests byte-sorted	59 of 59
Entries with non-UTF-8 path bytes	0
Rows not of the form path/blob/size	0
Manifest digest equals index row	59 of 59
Manifest commit equals index row	59 of 59
Index SHA-256 equals recorded value	yes
Refs: `refs/heads/main` / `refs/heads/master`	55 / 4
Repositories flagged private	3

the repository's own verifier verbatim, and executes five controls (four negative, one positive). It was run on 20 October 2026; the record's stamps are from 13 October 2026.

6.1 The corpus

Table 2 gives the measured shape of the corpus. The distribution is extremely skewed: the largest repository contributes 1,646 entries and 149,178 bytes (24.5% of entries, 27.6% of bytes), while the two smallest contribute 3 entries and 351 bytes each. Across the corpus the mean is 80.45 bytes of manifest per file entry, which is consistent with a fixed-width blob identifier dominating each line.

Three properties of the format held without exception. Every manifest's `files=` header equalled the number of body rows, so no manifest is truncated. Every manifest was byte-sorted, so the canonical order is not merely specified but realised. And every path was valid UTF-8, which means the byte-order/code-point-order ambiguity in the specification is untested by this corpus—a limitation of the evidence, not a property that the format guarantees.

6.2 Index and tokens

The audit verified all 120 tokens (60 per authority: 59 repositories plus the index) against the two committed anchors, with zero rejections, and found the imprint equal to the SHA-256 of the signed file in all 120 cases (Table 3). The two authorities returned distinct serial numbers with no reuse, and both used SHA-256. The index's own digest matched its recorded value, and every manifest digest and commit matched its row.

The generation times reveal how the record was produced. The first Sectigo token is from 13:28:43 and the first DigiCert token from 13:47:16, so the two witnesses are not simultaneous: the median gap between the two tokens for the same object is 1,096.5 seconds, the maximum 1,851.0 seconds, and the minimum 1.0 second, the latter attained by the index pair—stamped one second apart at 14:03:17 and 14:03:18—and by one repository (Fig. 3). A verifier should read the pair as "the manifest existed before each of two times roughly 18 minutes apart", not as two independent observations of the same instant.

One authority detail is worth recording: the Sectigo tokens name the signer in the token's TSA field, while the DigiCert tokens report the TSA field as `unspecified`. For DigiCert the signer is identified only by the certificate that signed the token, which the verifier obtains from the token's embedded chain. A verifier that relies on the TSA name field to decide whom to trust would silently skip every DigiCert token.

6.3 Negative controls

Table 4 summarises the controls. The first two show where the binding between token and data lives—in the imprint: a single flipped bit in the middle of the manifest, and a manifest from a different repository, are both rejected by `openssl` with a message-imprint mismatch. NC3 shows that the anchors are not interchangeable: verifying a Sectigo token with the DigiCert anchor and chain fails with "unable to get local issuer certificate". NC4 is a positive control confirming that the index row examined by the negative controls is the digest of the file on disk.

NC5 is the negative result that matters operationally. A token generated at 13:28:50 on 13 October still verified on 20 October, at an age of seven days, and it will keep verifying. `openssl` applies no expiry policy, and neither does the record's verifier. This is correct behaviour for an archival record—a timestamp is supposed to remain valid—but it means that "the token verifies" must never be read as "the token is recent". The freshness control that prevents replay lives in the minting client, not in the verifier.

6.4 The repository's own verifier, run verbatim

Running `verify.sh -all` produced exit status 0, 59 lines reporting "OK (digest matches index, both CA tokens verify)", 118 "token valid" lines, zero "token INVALID", zero "DIGEST MISMATCH", and zero "MISSING artifact" lines. The 118 is 59 repositories times two tokens; the two index tokens are outside the verifier's loop, which iterates over repositories only. This is a real gap in coverage rather than a defect in the tokens: the audit verifies the index tokens through a separate step of its own procedure.

6.5 Canonicalisation, re-derived

The audit also runs the canonicalisation end to end on a real tree: it rebuilds a manifest for the notarization repository itself at commit `de2255ddacba8cbe425ea315ad589824a053f1c1`, re-derives the same rows from the local git history with the repository's own re-derivation code, and compares them. The result is 316 entries, 316 re-derived

Table 3: Index integrity and token verification. "Imprint" means the message imprint inside the token equals the SHA-256 of the file it was verified against. The last generation times are those of the two index tokens.

	Sectigo	DigiCert
Timestamp tokens (`.tsr`)	60	60
Text renderings (`.txt`)	60	60
Accepted by `openssl ts -verify` offline	60	60
Rejected	0	0
Imprint equals SHA-256 of signed file	60	60
Distinct serial numbers	60	60
Hash algorithm	SHA-256	SHA-256
Policy OID	1.3.6.1.4.1.6449.2.1.1	2.16.840.1.114412.7.1
First / last generation time (UTC, 13 Oct 2026)	13:28:43 / 14:03:17	13:47:16 / 14:03:18

Table 4: Negative controls (NC1–NC3, NC5) and one positive control (NC4), executed by the audit against the `SM4Kit` record.

Control	Accepted	Observed
NC1 manifest with one bit flipped	no	imprint mismatch
NC2 token presented with another repository's manifest	no	imprint mismatch
NC3 token verified against the wrong anchor	no	no local issuer
NC4 index row equals manifest digest	yes	match
NC5 seven-day-old token re-verified	yes	no freshness policy

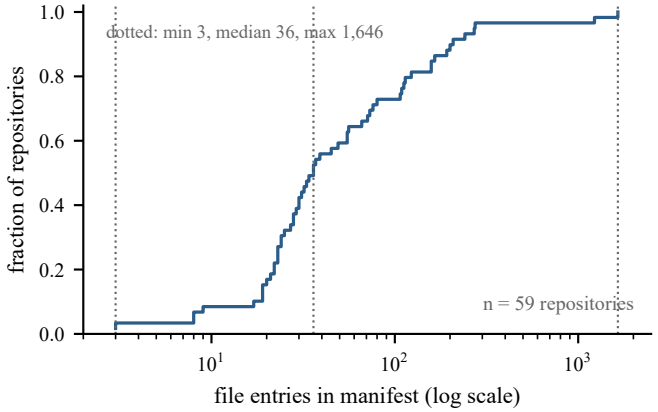


Figure 1: Empirical distribution of file entries per repository (59 records, logarithmic axis). Marked: minimum 3, median 36, maximum 1,646.

rows, 23,706 manifest bytes, an identical row sequence, and a manifest digest of `3bc3d9fe...193271` (abridged; the full value is in the audit JSON). This is the only place where the specification is checked against real git data rather than against itself, and it is a single repository.

6.6 Figures

Figures 1–3 show the corpus shape, the approximately linear relation between entries and bytes, and the two timestamp passes. The byte ratio is descriptive, not a verification rule; the audit treats it only as a way to describe how much text a record costs.

6.7 What was not measured

Verification time was not measured. The audit's JSON contains no timing field, so this paper reports no latency or throughput for token verification, manifest hashing, or the full 59-repository pass. Producing such numbers would require instrumenting `verify.sh` or wrapping each of the 120 `openssl` invocations in the audit with a monotonic clock and repeating the run; that was not done, and we prefer to say so rather than report a one-off wall-clock reading without a protocol.

The 59 manifests were not re-derived from their repositories. Re-deriving requires a clone of each repository; the audit used no network access, so only the notarization repository itself was re-derived (Section 6, 316 entries). The committed manifests were compared with the index, not with git. The clone-based checker exists and implements the comparison, but it was not run for any of the 59 records in the corpus.

Token accuracy and clock quality were not audited. The audit records the generation time but not the accuracy field of the tokens, and nothing in the record establishes a bound on either authority's clock error. The two timestamps are 18 minutes apart, which reflects how the record was scheduled; the gap is not evidence that the authorities disagree about the time.

7 Limitations

1. **Offline verification cannot prove that the TSA was honest at signing time.** A token proves that a key signed a structure containing a digest and a time. If the authority backdates, or if its key was compromised before signing, the record cannot detect it offline. Two authorities raise the cost of a false claim; they do not eliminate it, and they do not protect against collusion or against a compelled signing.

2. **There is no independent witness.** The record is not in a transparency log, and no third party holds a copy. An attacker who can delete the repository deletes the evidence with it; the tokens are unforgeable, but they are not repli-

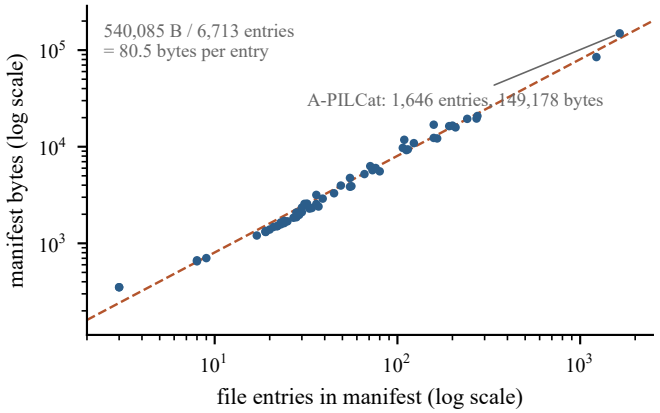


Figure 2: Manifest bytes against file entries, one point per repository (log–log). The reference line is the corpus mean of 80.45 bytes per entry (540,085 bytes over 6,713 entries); the largest record is annotated.

cated. The project's notes record a failed attempt to anchor digests in Rekor, so this is a known gap, not an oversight.

3. **The trust anchors are local artifacts.** The two roots were copied from a local operating-system trust store, and the committed chain files contain signer and intermediate certificates, including a root certificate that is not the configured anchor. Trust comes only from the `-CAfile` anchor; certificates in the untrusted bundle are candidates, not authorities. A verifier must decide independently whether those two anchors are the roots they claim to be; the record carries their SHA-256 fingerprints, but nothing in the offline procedure forces the comparison.

4. **Revocation is not checked.** With no network there is no CRL and no OCSP. A signer certificate that was revoked after signing, or that has expired, is not detected by the verification described here.

5. **The manifest binds names, identifiers, and sizes—not contents.** A row commits to a git blob identifier, which in turn commits to file content, but the content is not in the record. Without the repository, a verifier can establish that a listing with these blob identifiers existed; not that the files did. The blob identifiers are SHA-1, which is no longer collision-resistant against a determined attacker [Stevens et al., 2017]; the manifest's own digest is SHA-256, so tampering with the record is detected, but a deliberately crafted blob collision is outside what the record can rule out.

6. **Commit hashes were re-derived only once.** For the notarization repository itself the audit re-derived 316 rows from git and they matched. For the 59 records in the corpus the audit checked that the manifest header and the index agree; it did not re-derive them from their repositories.

7. **The private repositories are visible in the index.** Three repositories are marked private, and the record does not hide them: their names appear in `index/index.tsv`, and their manifests contain complete file listings, which is why the repository holding the record is itself private. The design provides no confidential mode; publishing the index would publish the directory structure of those repositories. The audit artifact that accompanies this paper inherits the same names.

8. **Single-author custody.** One person holds the access token that reads the repositories and runs the workflow. There is no separation of duties, no hardware-backed key, and no institutional continuity plan. The record outlives a deleted repository, but not necessarily a lost account: the tokens would survive only where copies of them exist.

9. **Freshness is a minting-time property.** The 900-second window prevents replay when a token is requested; verification accepts arbitrarily old tokens, as it must (NC5). A verifier who needs recency has to check the generation time against a policy of their own.

10. **Coverage is a snapshot of one account.** The corpus is 59 repositories: four forks were excluded, three because they contain other people's work and one because its tree exceeds the hosting API's entry limit and cannot be listed completely. Completeness is relative to the account listing at generation time, not to any external notion of the author's work.

11. **The verifier is small but not independently implemented.** Checking an existing manifest against its tokens needs only `openssl`; checking that the manifest describes a repository needs the formatting rules, and we did not write a second implementation of them. A verifier who runs `verify_git.py` is trusting a 47-line shell verifier and the Python that supports it, neither of which they have audited—less than trusting a platform, but not zero.

8 Related Work

Timestamping has a long line from Haber and Stornetta's computational time-stamping [Haber and Stornetta, 1991] to RFC 3161's TSP over CMS [Adams et al., 2001, Housley, 2009]. Commercial TSAs are routinely used as countersignatures in code signing and long-term validation of signatures; our use is the same primitive applied to a manifest of a repository rather than to a single artifact. The dual-authority structure is a simple redundancy: two operators, two roots, two policies, one digest.

Transparency logs make timestamping publicly auditable. Merkle trees [Merkle, 1989], history trees with efficient proofs [Crosby and Wallach, 2009], Certificate Transparency [Laurie et al., 2013], and Sigstore's Rekor [Newman et al., 2022] all provide append-only public evidence that our record lacks. A natural extension is to publish each manifest digest in such a

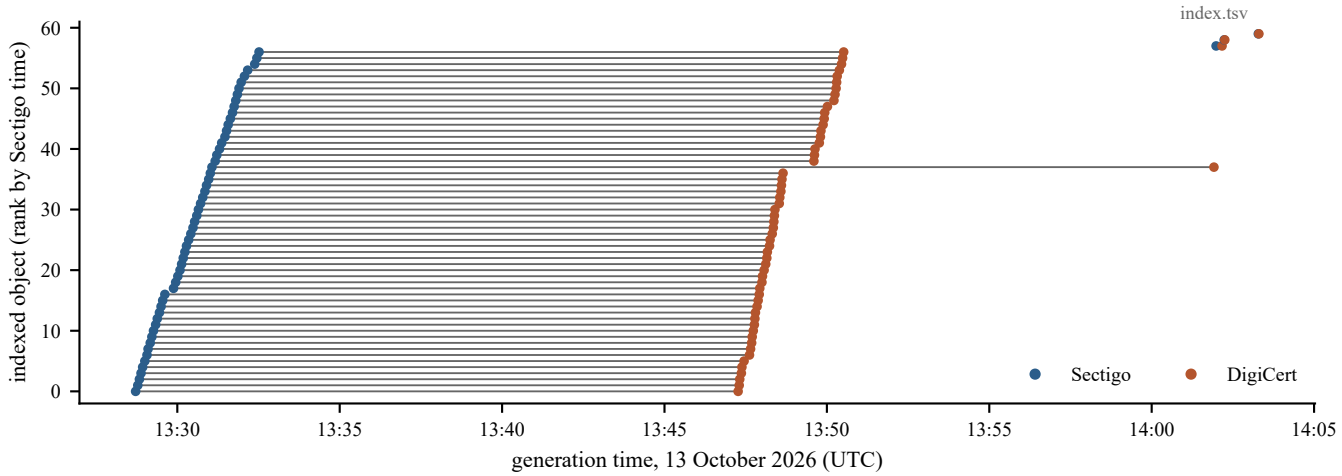


Figure 3: Both generation times for each of the 60 indexed objects (59 repositories and `index.tsv`), one horizontal segment per object, ordered by Sectigo time. The DigiCert pass starts 18 minutes 33 seconds after the first Sectigo stamp; median skew 1,096.5 s, maximum 1,851.0 s. The last segment is the index pair, one second apart.

log and to keep the RFC 3161 tokens as the offline fallback; the project attempted this and failed for tooling reasons recorded in its notes, which we report as a limitation rather than a result.

Supply-chain frameworks—TUF [Samuel et al., 2010], in-toto [Torres-Arias et al., 2019], SLSA [Open Source Security Foundation, 2023]—solve a different problem: they protect the integrity of updates and builds against compromised keys and builders, with signed metadata and verifier policy. Reproducible builds [Lamb and Zacchiroli, 2022] let a third party confirm that an artifact corresponds to source. Our record is complementary and weaker: it says nothing about how the code was built, only that a listing of it existed before two attested times. Its advantage is operational: no keys to manage, no policy language, and a verification path that is a hash and two signature checks.

9 Conclusion

We described a record that binds 59 repositories, 6,713 file entries, to two independently operated RFC 3161 authorities, and measured it. The audit found the record internally consistent and cryptographically intact: all 59 manifest digests and commits match the index, all 120 tokens verify offline against two committed anchors, and in all 120 cases the imprint inside the token equals the SHA-256 of the bytes it was checked against. Negative controls reject a mutated manifest, a mismatched manifest, and a wrong anchor, and one control shows that verification imposes no freshness—a property to be understood, not a defect.

The record's claim is narrow. It shows that particular bytes existed before a time attested by two commercial authorities, and it can be checked without network access and without executing the producer's code, provided the verifier reimplements the formatting rules to connect the manifest to a repository. It does not show authorship, does not carry file contents, cannot detect a dishonest authority, has no public witness, and does not hide the three private repositories it indexes. Those limits are the price of a record that one person can produce, store, and verify with a shell script and a TLS library's command-line tool.

Artifacts. The audit script, its JSON output, and the figure script accompany this paper. The record itself is maintained in a private repository because its manifests include private repositories; the verification procedure and the manifest format are fully specified above.

References

C. Adams, P. Cain, D. Pinkas, and R. Zorn. Internet X.509 public key infrastructure time-stamp protocol (TSP). RFC 3161, Internet Engineering Task Force, 2001. https://www.rfc-editor.org/rfc/rfc3161.

S. A. Crosby and D. S. Wallach. Efficient data structures for tamper-evident logging. In *Proceedings of the 18th USENIX Security Symposium*, pages 317–334, 2009.

S. Haber and W. S. Stornetta. How to time-stamp a digital document. *Journal of Cryptology*, 3(2):99–111, 1991.

R. Housley. Cryptographic message syntax (CMS). RFC 5652, Internet Engineering Task Force, 2009. https://www.rfc-editor.org/rfc/rfc5652.

C. Lamb and S. Zacchiroli. Reproducible builds: Increasing the integrity of software supply chains. *IEEE Software*, 39 (2):62–70, 2022.

B. Laurie, A. Langley, and E. Kasper. Certificate transparency. RFC 6962, Internet Engineering Task Force, 2013. https://www.rfc-editor.org/rfc/rfc6962.

R. C. Merkle. A certified digital signature. In *Advances in Cryptology — CRYPTO '89*, volume 435 of *Lecture Notes in Computer Science*, pages 218–238. Springer, 1989.

National Institute of Standards and Technology. Secure hash standard (SHS). FIPS PUB 180-4, 2015. https://doi.org/10.6028/NIST.FIPS.180-4.

Z. Newman, J. S. Meyers, and S. Torres-Arias. Sigstore: Software signing for everybody. In *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security (CCS)*, 2022.

Open Source Security Foundation. SLSA: Supply-chain levels for software artifacts. https://slsa.dev, 2023. Specification, accessed October 2026.

J. Samuel, N. Mathewson, J. Cappos, and R. Dingledine. Survivable key compromise in software update systems. In *Proceedings of the 17th ACM Conference on Computer and Communications Security (CCS)*, 2010.

M. Stevens, E. Bursztein, P. Karpman, A. Albertini, and Y. Markov. The first collision for full SHA-1. In *Advances in Cryptology — CRYPTO 2017*, volume 10401 of *Lecture Notes in Computer Science*. Springer, 2017.

The OpenSSL Project. OpenSSL 3.5.7. https://www.openssl.org, 2026.

S. Torres-Arias, H. Afzali, T. K. Kuppusamy, R. Curtmola, and J. Cappos. in-toto: Providing farm-to-table guarantees for bits and bytes. In *Proceedings of the 28th USENIX Security Symposium*, 2019.