The Word and the Ledger

Verifiable Commitment Provenance for SIGIL

A companion note to *SIGIL — The Variational Chain*, whitepaper v1

v0.2 · 2026-07-24 · illustrated + external Bitcoin witnessing (§6)

`bitknight.dipper688@passmail.net`
SIGIL / Flux Foundation · `sigilgraph.quillon.xyz`

"A binary you can verify, wrapped in a promise you cannot, is only half honest."

— after a reader's question, this note's origin

Abstract

SIGIL signs its released binaries with a hybrid post-quantum provenance proof, so that anyone can check, offline and forever, that a given artifact was attested by a named key. This note addresses a gap a reader identified: the *promises* around those binaries — the roadmap commitments, the security disclosures, the "single producer today" admissions, the conditions stated as preconditions for mainnet — carry no such provenance. GitHub can be reorganized, forum posts edited, websites and social posts rewritten or deleted. A project whose entire thesis is *make state divergence explicit and independently checkable* should be able to make its *own history* explicit and independently checkable by exactly the same means. We show that this is not an aspiration requiring new cryptography but a direct composition of primitives SIGIL already ships — content-addressing, require-both SQIsign+Ed25519 signatures, an append-only hash chain, and the block header's event-log commitment — because the project owns the Flux toolchain end to end. We define the *Commitment Record*: a content-addressed, hybrid-signed, hash-chained statement of intent whose digest is anchored in the chain's event-log root, so that a year later a stranger can prove that a specific promise, in specific words, existed at a specific block height, signed by a specific key — and detect, mechanically, any attempt to quietly rewrite it. We are explicit about scope: this makes a broken or edited promise *detectable and dated*; it does not, and cannot, force the promise to be kept. That distinction is the point. The construction is staged, not yet live; the note itself is offered as the first record the mechanism will anchor.

1 The gap a reader found

SIGIL's announcement makes many precise, dated, falsifiable statements: that the network is single-producer today; that the operator can reset or rewrite the chain; that there is no allocation and no sale; that a named short list of hardening items gates mainnet; that the post-quantum claim today covers release provenance and *not* the wallet or consensus signatures. These are exactly the statements that matter later — the ones a future observer, or a future version of the project, has an incentive to soften, reinterpret, or forget.

A reader (`fibonacciopstan`, on BitcoinTalk) put the problem cleanly: the released *binaries* carry cryptographic provenance, but the *promises* surrounding them have none. Repositories can be reorganized or replaced; roadmaps and websites can be updated; forum posts can be edited; announcements can disappear. So: *how does someone verify, a year from now, that these were the original commitments?* And — the sharper form — for a project built around holding the chain to account, shouldn't the community be able to hold the *project's own history* to account?

IN PLAIN WORDS — **the one-sentence version**

We can prove a downloaded program is the one we signed. We could not, until now, prove that a promise we made last year hasn't been quietly reworded since. This note fixes the second half with the same tools that fixed the first.

We think the observation is correct, and that it exposes an asymmetry worth closing rather than a rhetorical gotcha to deflect. The rest of this note is the answer.

Figure 1: The two provenance gaps, closed by the same key. A released binary carries a require-both seal (verifiable, ✓); the promise around it carries none (silently editable, ×). The same signer (`flux-rev` / require-both) that attests the artifact attests the words — one key, two things worth proving.

2 Why a provenance chain owes this to itself

SIGIL's core mechanism is *fail-closed recomputation*: a node commits four domain-separated state roots per block and halts rather than serve a state whose root it cannot reproduce (whitepaper v1, §state commitments). The philosophy companion frames the whole system as an *epistemic instrument* — a machine that makes claims explicit, types their evidence, and refuses authoritative service when local verification fails.

Turn that instrument on the project itself and the demand is symmetric. A chain that says *do not trust my ledger, recompute it* has no principled ground to say *but do trust my roadmap, I remember it correctly.* The credibility of the first statement is undermined by the ungrounded authority of the second. If divergence in *money* must be structurally impossible to hide, then divergence between *what was promised and what is now claimed to have been promised* should be structurally impossible to hide too — or the honesty is selective, applied where it is cheap and withheld where it might one day be inconvenient.

This is not a new principle; it is the transparency-log idea (Certificate Transparency, binary transparency, Sigstore/Rekor) applied to a project's declarative statements rather than its certificates or artifacts. What is specific to SIGIL is that the anchor need not be a third-party log we ask others to trust: it can be *our own chain*, whose whole reason to exist is to be an append-only, independently recomputable record.

3 The primitives already exist (because we built the stack)

The reason this is a composition and not a research program is that owning the Flux toolchain from the compiler up means every ingredient is already in hand and already exercised on shipped artifacts:

- ● LIVE **Content-addressing.** `flux-rev` produces a BLAKE3 `full:` identity over a byte set; the download manifest is BLAKE3 content-addressed. A statement's bytes get a canonical, collision-resistant name.
- ● LIVE **Hybrid post-quantum signing.** The release provenance is a *require-both* SQIsign Level 5 *and* Ed25519 attestation over a canonical record; a stripped single-leg bundle is rejected, and `fluxc verify-proof` checks both against pinned builder keys. The same signer signs a promise as readily as it signs a binary.
- ● LIVE **An append-only hash chain.** SIGIL is, definitionally, a chain of per-block commitments folded back to genesis; the light client recomputes the tip fingerprint in microseconds. It is already the anchor we need.
- ● LIVE **A header commitment with a home for arbitrary events.** Every block header carries an *event-log root* — a Merkle root over typed events emitted that block. On `sigil-g0` today it commits the canonical empty root (○ OPEN unexercised), but the mechanism that would carry a commitment event is the same one already producing the header field.
- ● LIVE **A constant-size prefix proof.** The fold compresses the chain prefix to a fixed 2,568 bytes; a commitment anchored below the fold horizon inherits a succinct existence proof *for free*, verifiable on a light client.

Nothing below invents a primitive — and this is the place to be careful: owning the Flux stack from scratch is what makes every ingredient *already in hand*, but it does not, by itself, make the result more *trustworthy*. Building your own tools strengthens nothing on its own. The trust comes from careful composition, canonicalization, *data availability*, and — the point of §6 — *independent witnessing*. The contribution here is naming the record, fixing its canonical form, and specifying the verification a stranger runs a year later.

Figure 2: One vertical spine, all of it ours: Flux compiler → content-addressing → require-both signing → the braid / event-log root → the fold → at the cap, a Commitment Record witnessed *inside* (seal) and *outside* (a Bitcoin receipt). Solid layers are live; dashed layers are staged — the honesty is drawn into the diagram.

4 The Commitment Record

A **Commitment Record** is a canonical statement of intent by the project. Its fields:

Field	Type	Meaning
`kind`	enum	`roadmap` | `disclosure` | `release` | `direction-change` | `retraction`
`body_hash`	`[u8;32]`	BLAKE3 of the exact UTF-8 statement text (the words themselves, stored alongside)
`timestamp_ms`	u64	the author's wall clock at signing (informational; the chain height is the trusted time)
`prev`	`[u8;32]`	content hash of the previous Commitment Record — the records form their own hash chain
`supersedes`	`Option<[u8;32]>`	for a `direction-change`/`retraction`: the record it revises (never deletes)
`signer_keys`	pubkeys	the SQIsign-L5 + Ed25519 public keys (pinned, published)
`sig`	bytes	require-both signature over the canonical serialization of all fields above

Two chained structures give it teeth. First, `prev` makes the records themselves an append-only hash chain: you cannot silently insert, delete, or reorder a past commitment without breaking every content hash that follows it. Second, the record's content hash is emitted as a typed `CommitmentAnchored` event, so it is covered by the block's *event-log root* at the height it lands — and thus by the header hash, the tip fingerprint, and (below the horizon) the fold. The chain's own recomputation now witnesses the promise.

Crucially, a `direction-change` *supersedes* but never erases. Changing your mind is allowed and expected; doing it *silently* is what the mechanism forecloses. The old record stays anchored; the new one points back at it. The history of the project's word becomes a diff, not a palimpsest.

> IN PLAIN WORDS — **what "supersedes, never deletes" buys**
>
> A project is allowed to change direction — honest projects do. What this stops is quietly pretending the old direction was never promised. The change is recorded as an edit on top of the original, both permanent, both signed. You can always see what was said first.

5 Verification, a year later

A stranger in July 2027, handed a claim of the form *"SIGIL committed in July 2026 that the testnet would be reset before mainnet"*, runs:

1. **Recompute the words.** BLAKE3 the statement text; it must equal the record's `body_hash`. (One byte changed anywhere and this fails.)
2. **Check the signature.** `fluxc verify-proof` the require-both signature against the *pinned* builder keys — the same fingerprints already published for binaries. A self-consistent record with swapped keys is a forgery and is caught exactly as a swapped-key binary proof is.
3. **Walk the record chain.** Follow `prev` back; any break means the commitment history was tampered with after the fact.
4. **Confirm the on-chain anchor.** Verify the `CommitmentAnchored` event's inclusion in the event-log root at block height H, and H under the tip / fold. The chain's own append-only time — not a server's clock, not a forum's edit timestamp — dates the promise.

If all four pass, the promise *provably existed, in those exact words, signed by that key, at that height.* If a later public statement contradicts it, the contradiction is now a checkable fact rather than a memory dispute — and if the project revised the commitment properly, step 3 surfaces the `supersedes` edit and its own anchor height.

6 Independent time: borrowing Bitcoin's clock

There is a limit the previous sections must not paper over, and it is exactly the one the original objection targeted. Three different things get proven, and they are not the same thing: the signature proves *this key attested these exact bytes*; the record chain proves *these records form a consistent sequence relative to a known head*; and a SIGIL anchor proves *this digest exists at height H in the presently accepted `sigil-g0` history.* While `sigil-g0` is single-producer and resettable, that third statement is *not* independent

calendar time — the same authority holds the promise key, the record chain, *and* the anchoring chain, and could in principle construct a different signed history later. To claim "provably existed at that height" as though it were a witnessed date would overstate precisely what the reader challenged. Asking SIGIL to certify SIGIL is a circle.

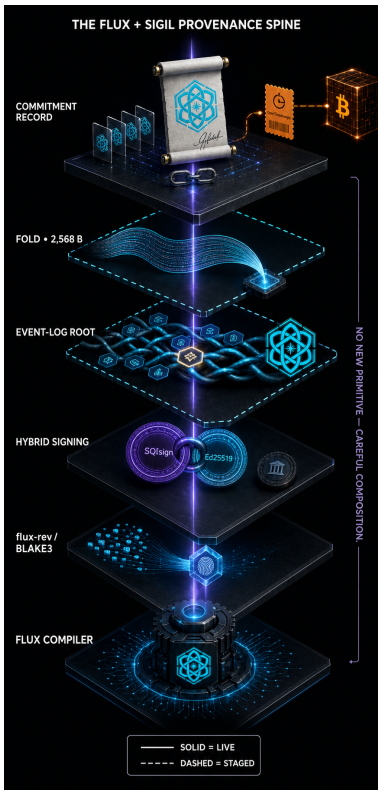

Figure 3: You cannot be your own witness. The promise key, the record chain, and the anchoring chain are one hand (the closed violet loop); a Commitment Record breaks the circle by also stamping onto Bitcoin — an authority we do not control. SIGIL proves *inclusion*; Bitcoin proves the *time*.

The break in the circle is an external, non-SIGIL witness. Each signed Commitment Record head is timestamped onto Bitcoin via OpenTimestamps: a proof that the data existed *no later than* an independently witnessed Bitcoin block, checkable by anyone with no trust in us. Until SIGIL has an independently operated, non-resettable producer set, that external timestamp — not the SIGIL height — carries the "no later than" guarantee; the on-chain anchor *strengthens* as the permissionless mesh lands. We did not defer this to a promise: • LIVE **this note is itself already Bitcoin-timestamped**, its receipt published beside it (`SIGIL_COMMITMENT_PROVENANCE_v0.pdf.ots`) and verifiable with `ots verify`. The best answer to "how will we know you kept your word" is a receipt that already exists, not one we intend to produce.

> IN PLAIN WORDS — **why Bitcoin, of all things**
>
> We can reset our own chain, so a timestamp only on our chain isn't independent proof of *when* — we could redo it. Bitcoin we can't redo. Pinning each promise to a Bitcoin block means the date is witnessed by something outside our control. That's the difference between a notary who works for you and one who doesn't.

7 What this does not do (scope, stated plainly)

The honesty section is load-bearing here as everywhere in the SIGIL corpus.

- ○ OPEN **It does not force a promise to be kept.** It makes a broken or edited promise *detectable, dated, and attributable.* Accountability is detection plus a permanent record, not enforcement. A project can still break its word — it just can no longer do so *quietly*, and that is the entire aim.

- ○ OPEN **It binds a key, not a person.** The signature proves the pinned project key attested the statement. Key custody, rotation, and compromise are the usual residual trust, identical to the binary-provenance case; the honest posture is the same.
- ◇ STAGED **The anchor's strength tracks the chain's.** Today `sigil-g0` is single-producer and resettable, so an anchor's censorship-resistance is only as strong as the network — which is to say, staged. The record chain (`prev`) and the signatures stand *independently* of that, giving tamper-evidence even before the chain is adversarial; the on-chain anchor *strengthens* as the permissionless multi-producer mesh lands. We label which guarantee is which.
- ○ OPEN **It is not yet live.** The event kind, the record schema, and the verifier subcommand are specified here and staged in the backlog, not shipped. This note claims a design and a first artifact, not a running system — consistent with the project's rule of showing receipts as pieces land, not before.

8 The recursion, and this note as its seed

The pleasing part is the symmetry. SIGIL asks you not to trust its ledger but to recompute it; this mechanism asks you not to trust its roadmap but to recompute *that*. The same four-step verification — recompute the bytes, check the signature, walk the chain, confirm the anchor — serves a wallet balance and a promise alike. The instrument built to hold money to account is turned, without new machinery, to hold the project's own word to account.

Accordingly, the first Commitment Record the mechanism anchors will be the SIGIL announcement's own preconditions for mainnet — the single-producer admission, the no-allocation statement, the named hardening list, the reset-before-mainnet pledge — signed and content-addressed, so that the promises this project is judged by are, from the start, the promises this project cannot later revise in silence. This note is offered as the seed of that record. The reader who asked the question will, by design, be able to check that we kept it.

Origin: this note answers a question raised by `fibonacciopstan` on BitcoinTalk. That a community reader defined the requirement, and that the project records the requirement's origin here, is itself the accountability the note is about.