STRIX

Execution Control for AI Systems · strixgov.com

Execution Binding Guarantee v1

Cryptographic binding of approved decision to executed side effect.

Version	1.1.0 (contract frozen)
Owner	Strix Kernel / Evidence
Status	Public technical whitepaper
Related	Signed Evidence v1, K-BIND-1, K-BIND-2, K-BIND-3, K-BIND-4, OB-1, OB-3, CJ-1
Generated	2026-05-10
Source	github.com/Tarshann/strix-platform

Verifiable claim. Every cryptographic claim in this whitepaper is independently verifiable using standard primitives (Ed25519, RFC 7517 JWKS, SHA-256). The reference implementation lives at *github.com/Tarshann/strix-platform*; the public verifier package is *@strixgov/verifier* on npm. The public JWKS endpoint is *https://www.strixgov.com/.well-known/strix-jwks.json*.

contractVersion: 1.1.0 — patched after Hostile Auditor Sim #1 **Status:** Contract frozen · implementation blocked on CP-K-001 §8 + SIM-1 remediations **Owner:** Kernel / Evidence **Related invariants:** CP-I-001 (record signing), Signed Evidence v1 (SE-1, SE-5, SE-7, SE-12, SE-14, SE-18), K-BIND-1, K-BIND-2, CJ-1 **Depends on:** `solo-builder-core/src/record-signing.ts`, `packages/strixgov-verifier`, `docs/architecture/canonical-json.md`

Why this exists

Strix today proves two things with cryptographic strength:

· A decision was evaluated against a specific policy version.

· An execution token was issued, scoped, and redeemed atomically.

It does **not yet** prove the third, load-bearing step:

> The side effect that actually ran was the same side effect that was approved.

Without that binding, "token redeemed" can silently mean "the kernel let it go" rather than "the approved payload executed". That gap violates the kernel invariant that *execution is bounded and revocable* — bounded execution with no record of what executed is an audit hole, not a control.

Execution Binding Guarantee v1 closes that gap with an append-only, signed record type — **`execution_receipt_v1`** — that cryptographically binds:

- the decision outcome,
- the redeemed token,
- the payload hash that was authorized,
- the side effect target,
- and the side effect result (or failure).

Non-goals

- Retrying or compensating failed side effects. Binding records what happened;

reliability is a separate concern.

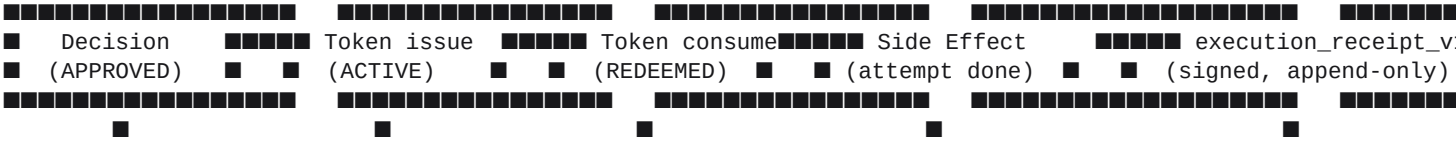
- Replacing **`execution_record`** (CP-H-001). **`execution_receipt_v1`** is a new,

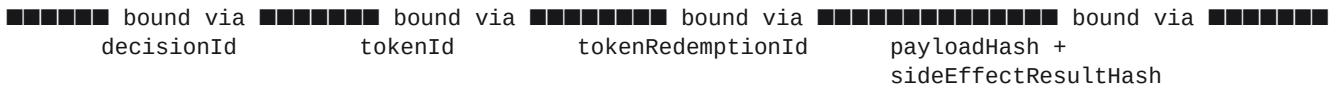
narrower artifact that answers *"did the approved payload execute?"* — the existing record continues to answer *"what did the CLI attempt?"*.

- Changing the Signed Evidence v1 payload schema. The 13-field locked order

is untouched. **`execution_receipt_v1`** is a sibling record, linked by **`evidenceId`**, not a replacement.

Core model





Load-bearing order of operations

This sequence is the kernel invariant, not an implementation choice. Any code path that produces a receipt MUST follow it exactly:

- **`evaluate`** — PolicyEngine runs, decision recorded with **`payloadHash`**.
- **`approve`** — decision transitions to APPROVED in the state machine.
- **`validate`** — execution token validated (signature, scope, expiry, revocation).
- **`consume`** — token atomically redeemed (**`ACTIVE`** → **`REDEEMED`**).
- **`execute`** — side effect attempted. The attempt either resolves

(**`COMPLETED`** / **`FAILED`**) or the caller abandons; there is no third *pending forever* state at the receipt-write call site.

- **`record`** — execution_receipt_v1 is constructed, signed, and written

only after step 5 resolves.

A receipt MUST NOT be written before step 5 resolves. "Record first, bind later" is explicitly forbidden — it collapses K-1 (evaluation before execution) and K-6 (execution vs outcome distinction).

Four states are made explicit and non-collapsible:

State	Meaning
`decisionStatus`	APPROVED / DENIED / EXPIRED — from the decision state machine.
`executionAuthorization`	TOKEN_ISSUED / TOKEN_REDEEMED / TOKEN_REVOKED.
`executionAttempt`	ATTEMPTED / NOT_ATTEMPTED.
`executionStatus` (write-time)	COMPLETED / FAILED — exactly these two. `UNKNOWN` is a verifier-output value only.

At write time **`executionStatus`** MUST be **`COMPLETED`** or **`FAILED`**. A receipt can never be written with **`UNKNOWN`** — the signer refuses (see §"Signer preconditions" below). **`UNKNOWN`** exists only as the verifier's rendering when sister records (decision, redemption, evidence) are unreachable at verify time.

Canonical signed payload (13 fields, locked order)

The payload follows the same locked-order, canonical-JSON pattern as Signed Evidence v1. Reordering, adding, or removing any field without a new gate report invalidates every receipt.

```
schemaVersion          (1)                      uint
receiptId              "exr_…"                  string (ULID-shaped, globally unique)
evidenceId             "evi_…"                  string — links to Signed Evidence v1 record
decisionId             "dec_…"                  string
tokenRedemptionId      "tr_…"                   string — atomic redemption reference
capabilityId           "cap.deploy.prod"        string — exact match, no wildcards
payloadHash            "sha256:…"               string — SHA-256 of SCJ v1(decision.actionParams)
sideEffectTarget       "https://…" | "urn:…"    string — opaque but stable identifier
```

```
sideEffectResultHash   "sha256:…"                string — always present, SHA-256 of SCJ v1(result descrip
bindingStatus          "BOUND" | "UNBOUND"       enum — BOUND ⇔ all §"Binding rules" pass
executionAttemptedAt   RFC3339 timestamp         string
executionCompletedAt   RFC3339 timestamp         string — always present at write time
executionStatus        "COMPLETED"|"FAILED"      enum — UNKNOWN is verifier-only, never written
```

The signature covers a 15-field document: the 13 payload fields plus **`environment`** and **`tenantId`** (SE-12/SE-14 invariant — captured at signing time, read from the stored record at verify time, never from **`process.env`**).

Hash fields

- **`payloadHash`** — **`"sha256:" + SHA-256(SCJ v1(decision.actionParams))`**.

`decision.actionParams` is the exact **`Record`** the PolicyEngine received at **`evaluate`**. SCJ v1 is pinned in **`docs/architecture/canonical-json.md`**. The hash is written onto the decision row at transition to **`EVALUATED`** and is **immutable thereafter** (invariant K-BIND-1). The receipt echoes the hash; the verifier compares echo to stored. A receipt whose **`payloadHash`** does not match the decision's stored hash is **`UNBOUND`** with reason **`UNBOUND_PAYLOAD_HASH_MISMATCH`** — never a generic error.

- **`sideEffectResultHash`** — **`"sha256:" + SHA-256(SCJ v1(result))`** where

`result` is a canonicalizable result descriptor provided by the caller. For a normal completion that is the HTTP response body, DB write summary, file checksum, etc. For watchdog-abandoned failures (see §"Async side effects") it is the abandonment descriptor: **`{ reason: "watchdog_abandoned", watchdogId, deadlineExceededAt }`**. Always present at write time. There is no null-result receipt and no "UNKNOWN" literal — the signer rejects it.

Signer preconditions (fail-closed)

Before the signer will produce a receipt, it MUST verify all of the following against live kernel state. A single failed check aborts the signing operation and no receipt is written.

· **`decisionId`** resolves to a decision in state APPROVED.

· **`tokenRedemptionId`** resolves to a redemption in state REDEEMED,

referring to **`decisionId`**, matching **`capabilityId`** exactly (no scope widening), and not revoked before **`executionAttemptedAt`**.

· **`payloadHash`** equals the hash stored on the decision

(recomputed from **`decision.actionParams`** under SCJ v1 and byte-compared to **`decision.payloadHash`**).

· **`executionStatus`** ∈ **`{COMPLETED, FAILED}`**. **`UNKNOWN`** refused.

· **`executionCompletedAt`** is non-null and ≥ **`executionAttemptedAt`**.

· **`sideEffectResultHash`** is non-null.

· **`(decisionId, tokenRedemptionId)`** tuple has no existing receipt.

· **`receiptId`** is globally unique in the receipt store.

These preconditions enforce invariant K-BIND-2 (*no signed-but-unbound receipts*). A signed receipt that would bind-fail at verify time is not allowed to be written in the first place.

Append-only semantics

- One receipt per **`(decisionId, tokenRedemptionId)`** tuple.
- **`receiptId`** is globally unique (ULID + DB unique constraint).
- Writes are refused if a receipt already exists for that tuple.
- Updates are refused categorically.
- Deletion is not a supported operation.

Async side effects

If a side effect is legitimately async (e.g. a job that the caller does not await), the caller has two — and only two — options:

· **Block until the attempt resolves**, then write the receipt with

`executionStatus=COMPLETED` or **`FAILED`**. This is the default path.

· **Watchdog-abandoned failure.** After a caller-owned watchdog

deadline passes, the caller writes a receipt with **`executionStatus=FAILED`** and a **`sideEffectResultHash`** covering an abandonment descriptor (see §"Hash fields"). **This path carries a hard integration obligation:** before writing the FAILED receipt, the caller MUST have either (a) genuinely killed the side-effect path such that late completion is not possible, or (b) ensured the side effect is idempotent / compensable so a late completion is a no-op against the compensation already applied. Writing a watchdog-abandoned FAILED receipt for a side effect that later silently succeeds is a trust-claim violation and a launch-blocker integration bug.

`executionStatus=UNKNOWN` is reserved for *verifier output* — a receipt that could not be resolved to a decision or side-effect result at verify time. It is never a write-time state. Writers MUST NOT produce **`UNKNOWN`** receipts. This preserves the rule that a receipt reflects *actual outcome, not intent*.

Reconciliation of redemptions (K-BIND-2)

Every token redemption MUST be matched by exactly one receipt within a bounded **`RECONCILIATION_WINDOW`** (default 60 seconds, configurable per capability but ≤ 5 minutes). The kernel runs a reconciliation pass:

- For each **`REDEEMED`** token with no receipt after the window, emit an

`UNRECONCILED_REDEMPTION` event to the proof surface and to telemetry/on-call.

- **`UNRECONCILED_REDEMPTION`** is visible on the console and public proof

page as a distinct state — never collapsed into "executed" and never hidden. An observer cannot be left unable to distinguish "token redeemed, execution happened, receipt suppressed" from "token redeemed, execution failed to attempt, no receipt".

This is how Strix defends against single-point receipt suppression.

Binding rules (verifier)

A receipt verifies as **`BOUND`** iff all of the following hold:

· **`signatureValid`** — Ed25519 verification succeeds against the JWKS entry

identified by **`signingKeyId`**.

· **`payloadHash`** equals **`decision.payloadHash`** as stored on the

referenced decision's append-only audit trail (not just the current row — invariant K-BIND-1 requires payloadHash is immutable from EVALUATED onward; verifier reads the audit entry at EVALUATED time).

· **`tokenRedemptionId`** references a redemption that:

• was **`ACTIVE`** → **`REDEEMED`** via atomic **`updateMany`**,

• matches **`capabilityId`** exactly (no scope widening),

• was not revoked before **`executionAttemptedAt`**.

· **`capabilityId`** equals the decision's capability.

· **`evidenceId`** resolves to a VERIFIED Signed Evidence v1 record whose

`capabilityId`, **`environment`**, and **`tenantId`** match the receipt.

· **`sideEffectResultHash`** is non-null (always, per signer precondition 6).

· **`receiptId`** is globally unique — no collision with any other receipt.

Any other combination is explicitly **`UNBOUND`** with a machine-readable reason code. The verifier never returns "bound" for a receipt whose signature is valid but whose payload hash mismatches — that is precisely the attack this feature exists to defeat.

Reason codes (stable strings)

```
BOUND                                  — all rules passed
UNBOUND_SIGNATURE_INVALID
UNBOUND_PAYLOAD_HASH_MISMATCH
UNBOUND_CAPABILITY_MISMATCH
UNBOUND_REDEMPTION_MISSING
UNBOUND_REDEMPTION_REVOKED
UNBOUND_DECISION_NOT_APPROVED
UNBOUND_DECISION_PAYLOAD_MUTATED     — K-BIND-1 violation (decision row mutated post-EVALUATED)
UNBOUND_EVIDENCE_UNVERIFIED
UNBOUND_RESULT_HASH_INCONSISTENT
UNBOUND_ENVIRONMENT_MISMATCH
UNBOUND_TENANT_MISMATCH
UNBOUND_RECEIPTID_COLLISION
UNRECONCILED_REDEMPTION              — K-BIND-2 signal (not a receipt state; surfaces on token, not receipt
```

These codes are part of the public verifier contract. They are not human-readable strings to display — UIs MUST map them to localized copy. Changing a code string is a breaking change.

Proof API shape

`GET /api/public/verify?receiptId=…` returns:

```
{
  "receiptId": "exr_01HX…",
  "evidenceId": "evi_01HX…",
  "decisionStatus": "APPROVED",
  "executionAuthorization": "TOKEN_REDEEMED",
  "executionAttempt": "ATTEMPTED",
```

```
  "executionStatus": "COMPLETED",
  "bindingStatus": "BOUND",
  "bindingReason": "BOUND",
  "signatureValid": true,
  "environmentMatch": true,
  "tenantMatch": true,
  "signingKeyId": "strix-prod-2026-04"
}
```

Four states are always reported independently. A consumer that only inspects **`bindingStatus`** cannot accidentally collapse a token redemption into an execution success.

What breaks if this is wrong

- **`payloadHash`** reordered → every live receipt becomes invalid.

(Treat this file's field order as load-bearing, same as **`tokens.ts:175`**.)

- Receipt written before side effect attempt resolves → write is refused

at the signer (**`EXR_WRITE_BEFORE_RESOLUTION`**). There is no write-time **`UNKNOWN`** state. Consumers never have to decide how to interpret a "pending" receipt, because one cannot exist.

- Receipt written on token revocation → **`bindingStatus=UNBOUND`** with

reason **`UNBOUND_REDEMPTION_REVOKED`**. The receipt still exists because an attempt happened; it just does not bind.

Test obligations

See **`solo-builder-core/tests/execution-receipt.test.ts`**:

- approved payload executes and binds correctly (**`BOUND`**)
- payload hash mismatch → **`UNBOUND_PAYLOAD_HASH_MISMATCH`**
- capability mismatch → **`UNBOUND_CAPABILITY_MISMATCH`**
- tamper on **`sideEffectResultHash`** → signature invalid, binding fails
- tamper on **`executionStatus`** → signature invalid, binding fails
- signer refuses **`executionStatus=UNKNOWN`** at write time (K-BIND-3)
- signer refuses a receipt whose preconditions (decision APPROVED,

redemption REDEEMED, payloadHash recomputed-matches) do not all hold — even if **`executionStatus`** $\in$ **`{COMPLETED, FAILED}`** (K-BIND-3)

- duplicate **`(decisionId, tokenRedemptionId)`** → refused
- duplicate **`receiptId`** → refused (K-BIND-4)
- orphan redemption (no receipt within reconciliation window)

surfaces as **`UNRECONCILED_REDEMPTION`** (K-BIND-2)

- decision.payloadHash mutated post-EVALUATED → verifier detects

via audit trail → **`UNBOUND_DECISION_PAYLOAD_MUTATED`** (K-BIND-1)

• environment mismatch (signed in prod, verified in dev) → **`environmentMatch=false`**, **`signatureValid`** independent (SE-14)

• canonical field ordering: golden-vector canary committed before signer code under **`tests/golden-vectors/execution_receipt_v1/`** (F-16)

These are regression tests, not aspirational — the feature is not complete until all of them run green on CI.

Rollout

· Land schema + signer + verifier behind **`STRIX_EXECUTION_BINDING_V1=false`** (no behavior change by default).

· Write receipts in shadow mode (receipt created, not yet surfaced on proof pages).

· Enable verifier surfacing on **`/proof`** once shadow mode covers ≥ 99% of redemptions for 7 days.

· Flip **`STRIX_EXECUTION_BINDING_V1=true`** in production; from that point a missing receipt for a redeemed token is an incident, not a warning.

Open questions (deferred, not blocking)

• Full proof-chain walk across receipts (genesis → tip) is deferred to a later gate; **`evidenceId`** linkage is sufficient for v1.

• Replay/counterfactual surfaces ("would this payload have been approved?") stay out of v1 — they belong to a later, separate gate.