Vouch: Separating Structural Validity, Authenticated Evidence, and Decision Authority

Anonymous Author(s)

Abstract

Signed rule-engine receipts can collapse three questions: whether bytes are structurally valid, whether a consumer-authorized key authenticated those exact bytes and they match the consumer's expected context, and whether the result may enter an application-controlled decision sink. We present Vouch, which separates these transitions for a checked rule-engine profile. The released issuer implementation makes the signing operation reachable only after two evaluator paths from one live invocation satisfy the signability predicates; module-private tokens bind both paths to the builder. Verification authenticates the exact payload in a Vouch envelope using DSSE pre-authentication encoding under one consumer-policy entry and checks expected source, input, profile, and engine before minting a non-serializable Native evidence capability. Native denotes output of the released local dual-evaluator pipeline. A separate promotion mints a capability eligible to enter such a sink. Imported external-engine evidence, called Bridge evidence, follows a disjoint capability path. Executable contract fixtures evaluate the authority boundary, while the frozen workload and compile-time mutants characterize the dual-evaluator defense-in-depth behavior. Agreement is defense in depth, not semantic assurance: the paths share substantial substrate, common-mode mutations can survive, and an envelope alone cannot prove use of the released issuer or establish key custody.

CCS Concepts

• **Security and privacy → Software and application security**.

Keywords

evidence artifacts, trust boundaries, DSSE, decision receipts, rule engines, differential validation

1 Introduction

Consider a CI/CD gate that promotes a build after evaluating provenance, dependency, and test evidence. It emits a JSON receipt for later audit. Canonical parsing can establish that the receipt is well formed, while a signature can establish that some key signed it. Neither check shows that the key was authorized for this evaluator and context or that the authenticated run ended in a decision suitable for promotion. The receipt may bind a different source or input, record two disagreeing executions, or carry evidence from an external engine. Reducing these cases to one *verified* flag creates a confused-deputy boundary.

Related failures already appear in other signed formats. JWT algorithm confusion exploits validation rules that depend on attacker-controlled token metadata, while XML signature wrapping verifies one element but induces an application to consume another [23, 27]. Rule-engine receipts pose a similar problem because they outlive the process that produced them. Auditors archive them, CI systems replay them, and applications reduce them to compact outcomes. An external evaluator may also emit a report shaped like a native receipt. A consumer must therefore know which artifact was checked and what authority, if any, follows from that check.

Vouch is the evidence subsystem for a checked Core Semantic Kernel (CSK) profile. Here *authenticated origin* means only that a consumer-authorized key signed the exact payload bytes. It does not establish honest inputs or execution. Let π be the consumer policy and c an immutable copy of the consumer's expected context. The supported state machine is

$$\mathsf{VouchEnvelope}(\mathsf{RawReceipt}) \xrightarrow{\mathsf{Auth}_{\pi,c}} \mathsf{AuthenticatedNativeEvidence},$$

$$\mathsf{AuthenticatedNativeEvidence} \xrightarrow{\mathsf{Promote}} \mathsf{AuthenticatedNativeDecision}.$$

$\mathsf{Auth}_{\pi,c}$ first checks resource limits, the canonical consumer policy, and the canonical envelope; selects one policy entry; verifies its signature over DSSE PAE of the exact payload; then validates the receipt's canonical and intrinsic properties and compares profile, engine, source, and input with c. Its input is $(\mathsf{VouchEnvelope}(\mathsf{RawReceipt}), \pi, c)$, not raw receipt bytes alone. Expected context is not an extra signed field. Promote separately requires agreement, completed terminals, a decision value, empty diagnostics, and release-build status. Attacker-created serialized objects are not states in this machine because only the verifier and promoter can mint its live capabilities. Bridge verification uses another entrypoint and capability.

Issuance has a corresponding authority boundary. Signing a caller-supplied receipt would authenticate bytes without tying them to the current execution, so Vouch constructs the receipt internally. A reference evaluator and a separately implemented graph evaluator, called the Meaning Environment, each mint a module-private trace token. The builder consumes exactly one token from each path, both from the same live invocation, before the signability gate can construct a payload accepted by the key provider.

For this comparison, evidence/decision separation is Y when the cited layer explicitly requires a distinct application decision after evidence verification. Live class capability is Y only when eligibility to enter an application-controlled decision sink is represented by a non-serializable, non-caller-constructible runtime object.

The research delta claimed here is narrow and comprises two mechanisms: invocation-bound dual-evaluator issuance tokens and non-serializable, artifact-class-specific live capabilities that carry authority only inside the verifier process. Table 1 shows that these mechanisms are not specified by the five comparison rows covering six cited specifications or systems; it does not claim historical priority for either mechanism in isolation. Vouch's exact-payload authentication, consumer-supplied expected context, and separation of authenticated evidence from decision authority are

Table 1: Property comparison at the cited specification or released-system layer. Y means explicit, P means partial, optional, or delegated to a profile, and – means not specified at that layer, not that every implementation lacks the property. The columns are exact-payload authentication, consumer-supplied expected context, evidence-versus-decision-authority separation, artifact-class-specific live capability that is non-constructible through the supported package API, invocation-bound issuance token, signability refusal before key access, and non-conversion of external evidence. This scoped comparison does not establish universal absence or historical priority.

System	Exact payload	Expected context	Evidence/ decision	Live class capability	Invocation token	Pre-key refusal	External nonconversion
DSSE/in-toto [6, 22]	Y	P[a]	P[b]	–	–	–	–
RATS RFC 9334 [2]	P[c]	Y	Y	–	–	–	–
SLSA v1.2 [17–19]	P[d]	Y	Y[e]	–	–	–	–
Sigstore policy controller [26]	P[f]	Y	Y	–	–	–	–
SCITT RFC 9943 [1]	Y	P[g]	Y	–	–	–	–
Vouch	Y	Y	Y	Y	Y	Y	Y

[a]The base formats leave expected-context comparison to a predicate or profile. [b]They leave the application decision to a consuming layer. [c]Exact-byte authentication depends on the evidence format and appraisal scheme. [d]Artifact identity is digest-based rather than a requirement to authenticate submitted artifact bytes. [e]A VSA serializes a verifier's policy result that a consumer may accept or further appraise, whereas Vouch keeps decision eligibility non-serializable and local to the verifier process. [f]Signature or attestation requirements are selected by admission policy. [g]Expected context is supplied by Relying Party policy outside registration.

a disciplined implementation of established principles rather than new primitives. The dual-evaluator check is defense in depth for detecting some path-specific implementation bugs, not an assurance argument; the paths share parsing, normalization, numeric and value substrates, canonical writing, and hashing, and the mutation results include common-mode survival. Pre-key refusal and Bridge non-conversion are enforced consequences of the two mechanisms, not additional novelty claims.

Contract fixtures evaluate the authority boundary. The frozen workload and compile-time mutants characterize the dual-evaluator defense-in-depth behavior, and a pinned release run checks reproducibility and absolute costs. Authentication remains relative to consumer policy and establishes neither freshness, key custody, input provenance, nor policy quality. The results apply only to the identified contract, fixtures, and frozen corpus.

2 System and Threat Model

2.1 Checked execution

The host language draws its language shape from R7RS [24] and fixes deterministic execution, but Vouch does not expose the whole language as a signed decision surface. The release uses `csk.checked-profile/v1`, a higher-order functional subset with immutable input, exact canonical values, lexical closures, arithmetic and list operations, the total predicate `exact-integer?`, and four terminal decision constructors `approve`, `deny`, `review`, and `invalid-input`. Mutation, general I/O, continuations, and uncovered forms do not fall back to the full interpreter. They are rejected by lowering or reported as profile escapes.

A canonical `csk.checked-input/v1` object is mapped to an immutable host-language value and bound to the reserved name `input`. Source and input remain external to the receipt. Domain-separated digests bind their exact bytes and the mapped canonical value. The consumer supplies the expected source and input, and the verifier recomputes both bindings from copies made at entry.

The native pipeline evaluates the normalized program twice. One path executes checked Core forms in the reference evaluator. The Meaning Environment path traverses the canonical graph with its own environment, closure application, control flow, and primitive dispatch. The two paths nevertheless share parsing, normalization, numeric and value types, canonical writers, and hashing. We therefore describe them as separately implemented evaluator paths, not independent backends. They are compared by byte-identical canonical transcripts, including terminal state.

Receipt construction also uses an internal Meaning Environment report. That report is not independently authenticated and is neither a Native receipt nor an evidence or decision capability. Consumers receive the differential receipt and, when present, its authorized envelope.

2.2 Evidence and authority

Table 2 separates the objects that a consumer may encounter. A raw receipt can be structurally consistent without having authenticated origin. A valid signature over the Vouch envelope's DSSE PAE bytes authenticates its exact payload under a selected policy key, yet the payload may still record disagreement or another non-promotable result. A Bridge report is checked external evidence, never a Native receipt. Serialized verification reports are audit projections and do not carry the in-process authority of a live capability.

Suppose a reviewer receives a receipt for rule source s and input i. Structural verification can recompute the bindings and check transcript consistency, but receipt bytes alone have no authenticated origin. A signature from a key outside the reviewer's policy fails authentication. An authorized signature over disagreeing transcripts yields authenticated incident evidence, but promotion still refuses it. Likewise, a Bridge report may match s, i, and the expected engine exactly while remaining external evidence. Only an authorized envelope over the exact Native payload, followed by context checks and promotion, reaches the decision renderer. Authenticated disagreements can thus be retained for diagnosis without acquiring decision authority.

Table 2: Artifact classes and the authority they can carry. *Decision* means eligibility to enter an application-controlled decision sink through the supported API. The artifact demonstrates this boundary with the Native renderer, which reports authority status.

Object	What a successful check establishes	Authenticated origin	Native evidence	Decision
Raw Native receipt	Canonical structure, bound digests, graph derivation, and transcript consistency	NO	NO	NO
Policy-authorized signed Native envelope	Exact payload authentication plus structural and expected-context checks	YES	YES	only after promotion
Bridge report	Canonical external report and equality with consumer-supplied expected context	NO	NO	NO
Serialized verification report	Portable account of a prior check; never an authority-bearing object	depends on subject	NO	NO

2.3 Adversary and trust assumptions

The adversary may submit arbitrary artifact bytes, relabel a Bridge report, replace source or input files, mutate a signed envelope, mix objects from different releases, reconstruct a capability-shaped object, or race a filesystem path between check and use. The adversary may also supply a valid signed receipt whose evaluation result is not a decision. Our threat model assumes that SHA-256 is collision resistant and that Ed25519 signatures are unforgeable when authorized keys remain uncompromised. FIPS 180-4 and RFC 8032 identify the concrete hash and signature primitives used here. They do not establish these security assumptions [8, 13].

The out-of-band consumer policy is the trust root. It maps key identifiers to public keys and enumerates allowed profiles, payload types, engines, and minimum versions. The envelope's `keyid` serves only as a candidate lookup key. Authorization succeeds only if that same policy entry permits the payload type, profile, and engine. Permissions cannot be assembled from different entries. Policy quality remains an application responsibility.

The capability boundary also assumes a trusted JavaScript runtime. The module loader, package initialization path, and standard intrinsics used by the closure-owned `WeakSet` and `WeakMap` stores are part of the trusted computing base. The implementation does not capture a separately hardened set of intrinsics, so adversarial code execution before package initialization is outside the threat model.

3 Authenticated-Native Boundary

3.1 Byte-level canonicalization

Vouch treats bytes, rather than a parser's abstract JSON value, as the evidence object. Its shared `csk.artifact-json/v0` writer fixes member order, UTF-8 encoding, line endings, number spelling, and recursively closed schemas. Readers parse a candidate value, write it canonically, and compare the result with the submitted bytes before interpreting semantic fields. Reordered members, CRLF, optional Unicode escaping, trailing whitespace, byte-order marks, and duplicate-member collapse therefore fail at the byte gate.

The one-signature format is a *Vouch envelope using DSSE PAE*, not a conformant DSSE v1.0.2 profile. It applies DSSE's pre-authentication encoding to the payload type and exact payload bytes [22], but accepts only padded Base64 in the standard alphabet. Requiring one alphabet and padding rule removes alternate textual encodings of the same binary payload or signature at the canonical-envelope layer. The restriction gives each accepted payload/signature tuple one textual identity for the `envelope_sha256` recorded by native verification reports and prevents alternate encodings or duplicate-member collapse from producing parser-dependent envelope identities. The interoperability cost is explicit: conformant DSSE envelopes that use the URL-safe alphabet are rejected. The remaining choices are Ed25519, exactly one policy-authorized key, and signature verification before receipt parsing. The verifier neither repairs nor reserializes a payload before accepting its signature.

3.2 Issuance from a live invocation

The `issue-native` command accepts source, input, profile, an opaque key handle, and one output directory. It accepts no receipt, payload, decision, transcript, evaluator token, or caller-supplied engine identity. At entry, it opens the source and input paths once and copies their contents into private buffers. All later parsing, hashing, evaluation, construction, and self-verification use those copies.

Each evaluator returns a module-private token containing its invocation nonce, context digest, input binding, profile, budgets, and complete transcript. Callers can neither serialize nor construct these token types. The receipt builder requires one token of each type, verifies that both belong to the current invocation, and consumes them. Reuse, cross-invocation pairing, transcript substitution, and superficially matching root counts all fail before signing.

After constructing the receipt, the issuer checks internal consistency without rerunning either evaluator. It re-derives graph bytes from the normalized program carried in the receipt, re-maps the retained input, checks transcript completeness and comparison consistency, and checks both tokens against the current invocation. Avoiding a third execution also avoids presenting that execution as provenance.

3.3 The pre-key gate

Internal consistency alone does not make a receipt signable. Vouch next checks, in order, that the transcript comparison agrees, both terminals completed, the final value is a decision, diagnostics are empty, and sealed build metadata identifies a release with no selected mutant. The private `SignablePayload` type exists only

Table 3: Supported-API invariants and their enforcing mechanisms. IDs name v8.6 contract conditions or executed fixtures. U14 is post-freeze supplemental evidence and is excluded from the sealed 165-fixture count.

Invariant	Enforcing mechanism	Contract or fixture evidence
Raw receipts cannot enter the Native renderer	The renderer accepts only `AuthenticatedNativeDecision` and checks its private decision `WeakSet`/`WeakMap` membership.	A-17; N02 rejects a raw receipt; U11/U12 reject unpromoted evidence.
Bridge capability cannot enter Native promotion	Promotion immediately requires membership in the `AuthenticatedNativeEvidence` set; `CheckedBridgeEvidence` has a different brand, constructor secret, set, and snapshot store.	C-CAP-03; supplemental U14 directly rejects live Bridge-to-Native promotion as `wrong-evidence-capability` and returns no decision capability.
A serialized object is never an authority-bearing capability	Constructor secrets are closure-owned, membership is object-identity based, and no deserializer can insert an object into a capability set.	A-16; N19 and U07–U10.
Key resolution follows `SignablePayload` construction	The private Rust type has no earlier constructor; the key provider's signing method requires it, and resolution/load are downstream.	C-ISSUE-04, C-KEY-07/08; I01–I10 and S6-KEY-AUDIT-01 record zero pre-key operations.
Promotion cannot bypass authentication	Only complete signature, intrinsic, and context verification mints a live `AuthenticatedNativeEvidence`; promotion begins with exact membership and snapshot lookup.	A-16; N02, N19, U07–U10.
Policy authorization is atomic within one entry	The envelope `keyid` selects one entry whose payload, profile, and engine permissions cannot be combined with another. Source and input are separately compared with one copied expected-context snapshot.	C-VN-06/C-VN-11; N10–N13 and N20.

after those checks and the running-build identity checks pass. Key-handle resolution and PKCS#8 loading occur downstream. For every enumerated pre-key refusal fixture, counting-provider tests record zero resolve, open, load, query, and sign operations.

Thus, the released issuer implementation makes its signing operation reachable only after a live two-path evaluation satisfies the signability predicates and constructs `SignablePayload`. This is a source-level control-flow property covered by counting-provider fixtures. A consumer holding only the envelope cannot cryptographically verify that this issuer path was used, and key custody is outside the claim.

The release interface accepts only a bounded absolute file URI under the `pkcs8-file` scheme. It rejects inline, environment, standard-input, and relative-file key material. After signing, the issuer derives the public key, constructs a one-key in-memory policy, and invokes the complete Native verifier over the new envelope and retained source and input. On success, it commits the payload, envelope, and issue report together by atomically renaming a staging directory within its parent. A handled failure publishes only its report.

3.4 Verification before promotion

`verify-native` follows the fixed verification order shown in Table 4. Earlier failures mask later ones by design. For example, altering a signed payload so that it is also malformed produces a signature error, not a schema error. The verifier does not rerun either evaluator.

Successful verification mints a live `AuthenticatedNativeEvidence`. An authenticated `disagree`, language fault, or `not-comparable` result can be retained as incident evidence, but cannot be rendered as a decision. A separate promotion operation reads a closure-owned immutable snapshot and repeats the requirements for agreement, completed terminals, a final decision, empty diagnostics, and a release build.

The three public capability classes `AuthenticatedNativeEvidence`, `AuthenticatedNativeDecision`, and `CheckedBridgeEvidence` have separate constructor secrets, membership sets, and snapshot stores. TypeScript brands prevent ordinary cross-use at compile time. Runtime membership checks reject plain objects, proxies, prototypes, structured clones, objects from another package instance, and deserialized reports. The Native renderer accepts only the promoted decision class. The Bridge renderer displays the narrower status *External evidence checked*.

Table 4: Primary order for authenticated-Native verification.

Step	Check
1	Resource limits and canonical consumer policy
2	Canonical submitted bytes and envelope class
3	Envelope schema, encoding, and one-key authorization
4	Ed25519 signature over DSSE PAE bytes
5	Canonical and intrinsically valid receipt payload
6	Profile, engine, source, and input expected context
7	Mint authenticated evidence; promote separately

The released consumer directly supports an in-process capability gate, where verification, promotion, and rendering remain in one trusted process. Across a process boundary, the boundary can be preserved by colocating those operations and the application-controlled decision sink in an application-owned decision service. The artifact does not ship that service or its authenticated transport, and its renderer exposes authority status rather than the underlying decision value, so this is an integration pattern rather than an implemented interface. The live capability must never cross the channel. A consumer that persists or forwards a verification report must later resubmit the original envelope, current policy, and expected profile, source, and input for fresh verification. Capabilities are module-instance-local and do not survive process restart.

3.5 External Bridge evidence

Bridge reports describe evidence produced outside the Native path. Their schema, expected context, verifier, error set, capability, and renderer remain separate. Before validation, the checker privately copies both the report and the expected source, input, profile, engine, and canonical-input binding. It then checks canonical encoding, schema, and context in a fixed order. A valid report means that

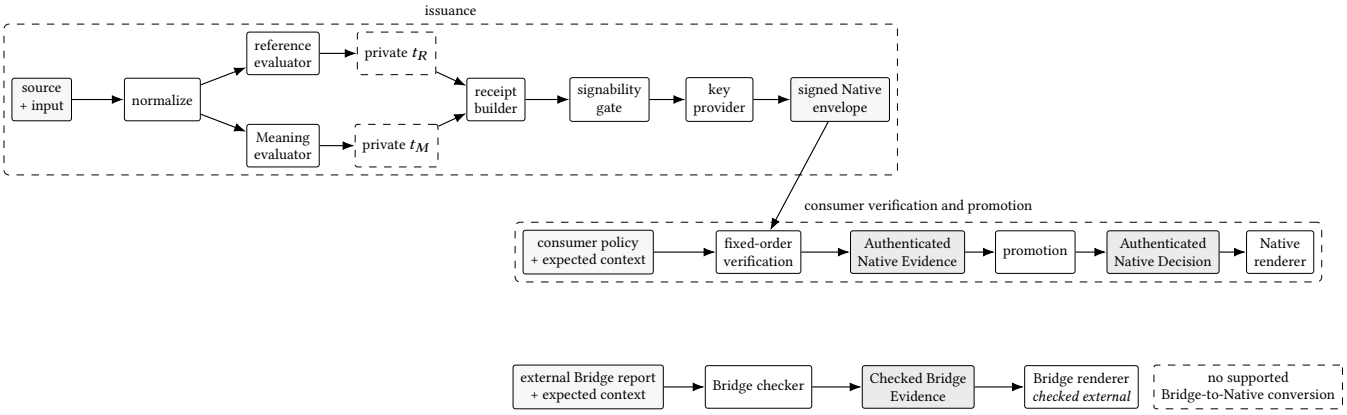

Figure 1: Issuance-to-decision architecture. The only edge into the key provider follows the signability gate. Consumer verification authenticates the exact envelope payload and then checks expected context before minting evidence. The Bridge lane is disjoint and has no edge into Native promotion or rendering.

the external object matches the consumer-supplied context. It does not authenticate the producer or convert the assertion into Native evidence.

No supported operation accepts `CheckedBridgeEvidence` in place of Native evidence or a Native decision. Regression fixtures exercise both cross-uses and include a Boolean-only consumer that labels any successful check *Verified*. The supported consumer instead renders *External evidence checked*.

4 Release Evidence

The release protocol supports inspection and reproduction rather than the core decision-authority claim. Signed descriptor D binds source, archive, executable, toolchains, target, and deterministic targets. After authenticating D, a network-disabled driver produces clean-run report Q. A finalizer derives and signs observation R; unsigned index P records the digests of D and R; phase 3 replays report derivations and emits terminal report S. Only D and R are signed. Phase 1 refuses Q unless every one of the 213 unique v8.6 conditions is implemented and covered by at least one known executable fixture. The immutable-buffer and read-once boundaries also exercise replacement and cross-release-mixing controls. This chain makes the record inspectable. It neither establishes workload representativeness nor constitutes an independent reproduction.

5 Implementation

The authenticated path is an off-by-default Cargo feature, so it does not alter ordinary host-language semantics or the default dependency graph. The Rust implementation contains the checked evaluators, transcript and receipt builders, canonical reader and writer, Ed25519/PAE policy verification, CLI entrypoints, key provider, and release tools. A separate TypeScript package implements the consumer capability boundary and repeats the contract-defined canonical and intrinsic checks without linking or executing the Rust evaluators.

Public object fields carry no security-relevant state. The TypeScript package stores class membership in closure-owned `WeakSets` and verified state in closure-owned `WeakMaps`. Entry buffers and nested values are copied or recursively frozen before authorization. The Rust issuer uses private types to encode the transition from a verified payload to a loaded key. Both implementations use closed error unions, so an authentication failure cannot be reported as `not-comparable`, nor can a promotion refusal be reported as an authentication rejection.

Version 8.6.0 introduces `csk.checked-profile/v1` and `csk.checked-input/v1`. It adds a total `exact-integer?` predicate independently to both evaluators, requires tagged rationals to have denominator greater than one and a nonzero coprime numerator, and requires workload rules to type-check all five positions before numeric operations. The fixture lane exercises every noninteger host-value alternative at every position under both rule versions. These changes resolve the checked-input contradiction recorded for v8.5.1 without reinterpreting the historical contract.

The mutation mechanism is compile-time only. A build selects zero or exactly one of M01–M12, and the interpreter and signing crate must agree on that selection. Unknown, duplicated, unpaired, or environment-injected selections fail compilation. Mutation experiments use an internal non-signing runner. Mutant payloads are never authenticated with the release key.

The release executable targets `x86_64-unknown-linux-gnu` with Rust 1.85.1, Node 22.14.0, npm 10.9.2, TypeScript 5.8.2, and glibc 2.39. The image, base image, lockfiles, vendored dependencies, empty Rust flag environment, and detached source identity are bound in D. The release key signs D and R, while phase-1 reexecution uses an isolated ephemeral key where issuance must be exercised without reproducing release signatures.

6 Evaluation

6.1 Method

We ask four questions. **RQ1, boundary enforcement:** do readers, verifiers, issuers, capability transitions, and release commands produce the results declared by the contract fixtures? **RQ2, workload behavior:** how does the frozen rule change behave over an

algorithmically selected development and held-out corpus? **RQ3, mutation sensitivity:** which path-specific and shared faults does that workload reach and expose? **RQ4, release reproducibility:** do deterministic artifacts reproduce, do authenticated report links close, and what absolute costs appear under the pinned protocol?

The normative v8.6.0 contract has SHA-256 `ecc294798be49f5843bd84e0ebad5d94a930f2b09f51db4852e42d2789addddc`. Its ledger contains exactly 213 unique conditions. The release gate passes only when every condition is implemented and references at least one known executable fixture. Skips, manual waivers, and unmeasured empirical values are not passes. Observed numerical values are recorded results, not thresholds selected as conformance requirements.

The workload was frozen before held-out execution. Each five-position vector models one synthetic welfare-policy record used only to exercise threshold-dependent decisions. Two parameter tables define six thresholds for each categorical stratum. From 1,536 candidates spanning threshold boundaries, interval interiors, and specified application-invalid transformations, a hash-ranked procedure selects 240 cases and splits them into 192 development and 48 held-out cases. The freeze commit contains this selection, split, and predeclared impact-scope plan, but no held-out receipts or observations. All inputs are synthetic. The frozen workload exercises deterministic threshold-dependent evaluator behavior only; in a supply-chain deployment, the corresponding inputs would be provenance, test, vulnerability, and dependency-policy observations, while the Native/Bridge and authority transitions are domain-independent.

For example, the five-position input `[2026,0,1,1,250000]` passes the application schema. Both rules use the same thresholds, but the baseline labels the interval $[200156, 300156)$ `approve` while the changed rule labels it `review`. This is a concrete illustration of the frozen rule change, not a claim that the workload represents a deployment population.

The synthetic history records that ordering explicitly. Freeze commit `c90f97ddd6b1d662791a76fe4663b90e79c443ec` fixes the workload inputs, selection, split, and impact scope before any result path exists. The v8.6.0 contract and release implementation enter later, and the clean release uses source commit C0 `3e910c9ff87cc01d3bc241d63297218b44e75ede`. The pre-observation workload state is thus identifiable separately from the release machinery that consumes it. This synthetic history documents the released ordering. It is not an independent timestamp or third-party preregistration record.

Before assembly, a network-disabled container completed `npm ci --offline` from a host-seeded cache. Archive assembly then enabled networking while populating a new in-archive npm cache from the lockfile. *D* subsequently bound the completed cache and archive. Container networking was disabled again for phases 1, 2, and 3. The digest-pinned `linux/amd64` container ran in a Lima VZ virtual machine with Rosetta translation on an Apple host. Thus *pinned Linux run* describes the guest environment, not bare-metal x86 execution or an independent reproduction. The host model and virtual-machine resource allocation were not pinned, so the performance values below are run-specific diagnostics rather than portable benchmarks.

6.2 RQ1: Boundary enforcement

Table 5 reorganizes existing contract results around attacks rather than presenting a new experiment. All 165 fixtures returned their declared black-box result, with no mismatch, skip, or design-target row. Supplemental fixture U14 additionally passes a genuine live `CheckedBridgeEvidence` capability to Native promotion and confirms a wrong-evidence-capability rejection with no minted decision capability; added after the sealed release run, it is excluded from the 165-fixture count.

An authorized signed disagreement can come from a policy-authorized signer other than the released issuer, an older issuer version still permitted by policy, or misuse of an authorized key. The envelope alone cannot distinguish these cases.

The previous v8.5.1 record left two checked-input conditions unresolved. Version 8.6.0 did not relabel those historical rows. It versioned the checked profile and input schema, added the total integer predicate to both evaluator implementations, expanded application-schema fixtures, and made ledger completeness a release gate. At C0, all 213 v8.6.0 conditions are implemented and covered by at least one executable fixture. This establishes conformance to the authors' contract, not completeness of the contract as a security specification.

6.3 RQ2: Frozen workload

All 240 selected cases produced promotable receipts under both rule versions, for 480 receipt payloads in total. There were no profile escapes, not-comparable executions, pipeline failures, or cases excluded from the decision matrix. Table 6 gives the decision transition matrix.

The rule change produced 76 off-diagonal transitions. Of these, 63 fell in development and 13 in the held-out partition. The predeclared plan identified approve to review, review to approve, and deny to review as three potentially changing intervals. Every held-out stratum contained a candidate in one of those intervals, placing all 48 strata within the declared impact scope. The plan did not predict that all 48 held-out cases would flip, and we treat it as a predeclared impact-scope statement rather than a calibrated predictor.

Coverage is the union of version-qualified instrumentation identifiers reached by the 240 selected cases under the baseline and changed rules. Mutant executions are excluded. The workload covered 596 of 620 graph-node identifiers and 102 of 120 branch-arm identifiers. The structured coverage report lists every uncovered identifier.

The 48 `invalid-input` cases remained valid host values and entered both evaluators, which returned the explicit invalid decision under both rules. The fixture lane separately exhausts every non-integer checked-host alternative at each of the five positions. That application-schema result comes from the fixture gate, not from extrapolating these 48 workload cases.

6.4 RQ3: Mutation sensitivity

Mutation testing injects controlled faults to determine whether a test or comparison surface exposes them [7]. From the same source identity, the campaign built all twelve compile-time mutants and one baseline binary. All thirteen executable digests were

Table 5: Attack-driven re-presentation of released evidence. Naive-design outcomes are analytic unless marked U02 or U03, which are executable negative controls. Vouch outcomes name executed fixtures or contract conditions. We did not implement or time a naive baseline and do not present these arguments as measurements.

Scenario	Boolean/plain-signature/self-described-context design	Vouch
Valid signature, wrong context	Accepts a signed self-description when no consumer expectation is compared (analytic).	One copied expectation is checked; engine, source, input, and profile mismatches are rejected (N10–N13, N20; C-VN-11).
Signed disagreement	Collapsing signature success to *verified* can grant entry to a decision sink (analytic).	Authenticates as incident evidence but promotion returns `comparison-not-agree`; released issuance refuses before key access (N14, I01).
Unsigned but structurally valid receipt	A strict schema accepts the native shape without origin authentication (executed negative control U02).	Structure succeeds, but authenticated verification returns `missing-native-attestation` (N02).
Bridge relabeled as Native	A Boolean consumer displays *Verified* after any successful check (executed negative control U03).	Raw relabeling and signed wrong-class payloads fail; a live Bridge capability can neither enter Native promotion nor render natively (B02, N07, U05/U06, U14; repaired display U04).
Cross-invocation trace mixing	A builder over caller-supplied serializable transcripts can combine matching records from different runs (analytic).	Nonce-bound private tokens from the current invocation are required and consumed (C-ISSUE-13; S4-TOKENS-01).
Deserialized capability reconstruction	A public field or structural brand can be reconstructed or cloned (analytic).	Identity membership rejects a serialized report, plain object, Proxy, structured clone, and cross-realm imitation (N19, U07–U10).
Payload mutation that also makes the schema malformed	A parser-first design may report only the later schema failure (analytic; plain DSSE itself is not claimed to accept the mutation).	Signature verification precedes payload parsing, so mutation is `native-signature-invalid` (C-VN-09; N03, J09).
Behavior after pre-key refusal	An issuer that resolves its key before validation exposes the provider to refused inputs (analytic).	The refusal publishes no payload/envelope and records zero resolve, open, load, query, or sign operations (I01–I10; S6-KEY-AUDIT-01).

Table 6: Decision transitions over the 240 selected cases. Rows are baseline and columns are changed.

	Approve	Deny	Review	Invalid	Row total
Approve	37	0	28	0	65
Deny	0	53	16	0	69
Review	32	0	26	0	58
Invalid	0	0	0	48	48
Column total	69	53	70	48	240

distinct. The frozen workload was selected to exercise the rule change, not to maximize mutant coverage. A mutant activates only when a selected case reaches the altered operation and changes at least one path relative to baseline; code outside that surface can remain inactive. M07, M08, M11, and M12 deliberately alter shared numeric/value or reader/normalizer substrate, so their dedicated-witness outcome is intentionally common-mode rather than disagreement.

The campaign targets lowering, evaluator, shared numeric, path-serialization, and reader/normalizer faults. It does not mutate invocation-bound issuance tokens or the live-capability boundary; authority-boundary mutation remains future work.

A separate witness suite activated all twelve mutants and confirmed their fault classes: eight witnesses produced disagreement and the four shared-substrate witnesses produced common-mode change. On the frozen workload, M01, M03, M05, M09, and M12 activated. The first four produced disagreement; M12 changed both paths for seven cases and survived the comparison. Table 7 reports this class-level result. The registry-level differential detection yield

Table 7: Mutation results over the frozen workload. *Activated* means activated by this workload, not merely by the dedicated witness suite.

Fault class	Built	Activated	Detected
Lowering	2	1	1
Graph evaluator	2	1	1
Source evaluator	2	1	1
Shared numeric	2	0	0
Path serialization	2	1	1
Shared reader/normalizer	2	1	0
Total	12	5	4

is therefore $4/12 = 33.3\%$. The conditional $4/5$ among activated mutants is only diagnostic because reachability is part of workload adequacy.

The case-level unit is one (mutant, selected case) pair, evaluated under both rule versions and collapsed to one outcome. Across twelve mutants and 240 selected cases, 647 of 2,880 pairs activated. Of these, 640 produced differential disagreement and seven produced a common-mode change. The remaining 2,233 pairs did not activate the selected mutation. Under the registry definition, there were no pipeline or infrastructure failures and no activated path-specific survivor.

6.5 RQ4: Release and cost

The outer driver compared 482 deterministic targets and recorded byte identity for each one. The targets are 480 receipt payloads, the replay manifest, and the workload-runner binary. During phase 1,

stored authenticated envelopes are verified rather than regenerated.

The terminal report records status pass, chain verification pass, paper-claim comparison true, and claim-language scan pass for the machine-generated release record.

The timed phase-1 window lasted 5122 seconds. That window covers archive authentication and extraction, offline installation, release build, fixtures, workload, mutation campaign, performance measurement, release scans, and the final worktree-clean check. It excludes phase-2 finalization and phase-3 record rendering.

Performance measurements use five warmups, thirty measured runs, a monotonic clock, and nearest-rank percentiles. Native verification has 14,400 observations (480 envelopes times 30 runs). Envelope size has a population of 480, and replay latency and peak RSS each have 30 observations. Table 8 reports absolute measurements under the pinned protocol. The container image, guest toolchains, and target were pinned. The Apple host model and the virtual machine's CPU and memory allocation were not. The host was also not isolated from concurrent activity. These values are therefore run-specific diagnostics, not portable performance estimates, isolation results, or comparative cost measurements. Because the experiment has no matched unauthenticated baseline, we do not report an overhead ratio.

7 Security Discussion and Limitations

7.1 Enforced properties

The supported APIs enforce five scoped properties. *Exact-payload authentication* covers the canonical bytes later interpreted as the receipt. *Expected-context binding* checks profile, engine, source, and input instead of trusting the payload's self-description. *Capability separation* leaves raw receipts and Bridge capabilities without a supported conversion to a Native decision. *Pre-key refusal* keeps incomplete, disagreeing, diagnostic, non-decision, and mutant executions from accessing the key provider in the released issuer implementation; it is not envelope-verifiable provenance. Finally, for *release linkage*, D authenticates the deterministic release identity, R authenticates Q and the recorded observations, P indexes D and R, and S records phase-3 verification. Phase 3 co-publishes the machine-record PDF and S, but S does not carry the PDF digest.

These properties are not interchangeable. Structural validity does not authorize a signer, authentication does not imply promotion, and a passing release chain does not make the workload representative. Keeping the properties separate prevents one kind of evidence from being laundered into another.

7.2 Validity threats

An accepted Native envelope authenticates exact payload bytes under a key that the consumer authorized for the payload type, profile, and engine. It does not cryptographically prove that the released issuer, its live two-path evaluation, or its pre-key ordering produced those bytes. It also does not establish key custody, revocation, timestamping, honest input provenance, or deployment behavior. Promotion establishes eligibility to enter an application-controlled decision sink only for an agreed final decision with completed terminals and empty diagnostics, but says nothing about fairness, legality, or rule quality. Applications can also ignore the capability API, so capability non-constructibility is scoped to the supported package and runtime boundary.

The evaluator paths differ in traversal, control flow, environment handling, and primitive dispatch but share readers, normalization, numeric and value substrates, writers, and hashing. Disagreement is a direct counterexample. Agreement is a bounded observation. The seven common-mode mutant–case observations show how a shared fault can survive the differential comparison. The deterministic workload is synthetic and threshold-focused rather than sampled from a deployment population, so its complete 240-case result carries no population confidence interval.

The release ran in a pinned Linux guest translated on an Apple host. Its host model and virtual-machine resource allocation were not pinned. The authors implemented the system and ran the release and audit workflow themselves. The signed D and R objects, verified digest links, and public source make the run inspectable, but they do not constitute an independent reproduction. A clean rerun by another party remains necessary.

The anonymous artifact's `.git`-free source projection contains all 2,367 tracked files from sanitized C0 byte for byte. These include the Rust and TypeScript implementation, release tools, tests, contracts, workload and mutation definitions, documentation, lockfiles, and vendored Rust dependencies. The source manifest also records every projection-authored review-tooling and offline-dependency file and its reason. The projection contains no release private key and omits the original non-anonymous upstream history. It is a convenience view, not an object authenticated by D or S. For exact checkout, the snapshot also carries a synthetic-history Git bundle resolving the freeze and C0 commits. That synthetic history is not an independent timestamp. For a phase-1 rerun, it provides the D-bound release archive as ordered 7 MiB chunks with per-chunk and whole-archive digest checks.

8 Related Work

Confused artifact types. JWT guidance recommends explicit typing and mutually exclusive validation rules because a token accepted in the wrong context can cross a trust boundary [23]. XML signature wrapping similarly showed that verifying one element while consuming another can defeat an otherwise valid signature [27]. Capability systems address confused-deputy failures by binding designation to authority and supporting least-privilege delegation [5, 12]. Vouch applies the same principle at a narrower boundary. A verified result does not obtain decision authority merely because its syntax resembles an accepted artifact. The verifier authorizes a specific artifact class, payload, profile, engine, and execution context.

Provenance and attestation. RATS is the closest prior architecture: an Attester supplies Evidence, a Verifier appraises it into an Attestation Result, and a Relying Party applies its own policy [2]. Vouch does not claim that evidence/appraisal separation as new. It enforces a narrower rule-receipt boundary with live artifact-specific capabilities.

In SLSA's Verification Summary Attestation (VSA), a trusted verifier evaluates an artifact and its attestation bundle against policy, then issues a serialized attestation of that result so downstream consumers can delegate verification rather than recheck the full

Table 8: Absolute performance observations from the pinned release-key-backed run.

Metric	Unit	Median	p95	Maximum
Native verification latency	ms	59.459	170.124	3 618.773
Selected-corpus replay latency	ms	14 864.689	17 709.783	24 130.904
Peak resident memory	byte	81014784	81174528	81195008
Signed envelope size	byte	59974	59994	59994

bundle [19]. Vouch takes the opposite authority-carrying choice: serialized verification reports remain non-authoritative and eligibility to enter such a sink exists only as a live verifier-process capability. This trades portability for resistance to report-shaped-object confusion.

SLSA provenance relates an artifact to its production and gives consumers verification expectations [17, 18]. An in-toto Statement binds a typed predicate to digest-identified subjects [6]; DSSE authenticates a payload type and exact bytes [22]. Sigstore's policy controller combines signature or attestation authorities with admission policy, including deployment-specific paths that do not require signatures [26]. SCITT authenticates signed statements and transparency receipts while leaving Relying Party trust decisions outside registration [1]. Rekor and Certificate Transparency likewise add append-only publication and inclusion evidence [10, 25]. Vouch uses a non-conformant Vouch envelope with DSSE PAE and provides no transparency log, trusted timestamp, or deployment attestation. Table 1 states the narrower mechanism-level delta.

Policy and rule systems. Cedar provides a purpose-built authorization language, formal semantics, an open-source Rust implementation, and mechanized analysis infrastructure [4]. Its symbolic compiler can check relations such as equivalence and subsumption and produce counterexamples for supported policies [3]. OPA records policy-query events in decision logs [15], and Styra documents replaying prior decisions against modified policy and data [28]. OpenFisca and PolicyEngine support tax and benefit analysis [16, 21]. JsonLogic serializes rules as JSON [9]. These systems focus on policy authoring, checking, logging, or replay. Vouch instead addresses the point at which an offline evaluation record is authenticated and made eligible to enter an application-controlled decision sink.

Differential testing and translation validation. Differential testing compares implementations to expose disagreement [11]. Translation validation checks a particular transformation rather than proving a compiler correct in general [14, 20]. Vouch performs no source-to-target translation. The connection is the narrower practice of validating each execution. Its evaluator paths share substantial substrate and have the same origin. A receipt reports their observed agreement, and an authorized envelope authenticates those receipt bytes. Neither establishes semantic equivalence.

9 Broader Impact

Decision receipts can improve procedural transparency but can also become an accountability shield. A deployer may present an authenticated receipt as if it showed that a rule was fair, that the input was true, or that an affected person had recourse. None follows from Vouch's checks. Real deployments should pair technical evidence with policy review, data-governance controls, explanations, and appeal mechanisms. Low-entropy input digests should not be published because they invite guessing attacks. This study uses synthetic inputs rather than personal, administrative, or production records.

10 Conclusion

Vouch separates three facts that rule-engine consumers often collapse. These are whether receipt bytes are structurally consistent, whether an authorized key authenticated those exact bytes, and whether the authenticated result may enter an application-controlled decision sink. Live evaluator tokens and a pre-key signability gate bind issuance to the current execution. Fixed-order verification binds origin to consumer policy and expected context, while opaque capabilities keep Native decisions apart from incident evidence and external Bridge reports.

The v8.6.0 release implements 213 contract conditions, each covered by at least one executable fixture. It matches all 165 black-box fixtures, exercises a frozen 240-case workload and twelve compile-time mutants, and carries the results through a release-key-backed lifecycle. These results remain bounded by shared evaluator substrate, a synthetic workload, a translated Linux guest, and the absence of an independent reproduction. The public artifact exposes every sanitized C0-tracked file for inspection and reexecution without distributing the release private key or original non-anonymous history.

Artifact Availability

The anonymized artifact is available at https://anonymous.4open.science/r/vouchartifactscored26/. The reviewed Git commit is `2261344682fdaa963a2225970132a567aa84c657`. It contains the normative contract, structured results, validators, sanitized source projection, synthetic-history bundle, and the *D*-bound release archive. Files explicitly listed in `LICENSE-SCOPE.md` are Apache-2.0. Projected first-party source is `UNLICENSED` and distributed only under the limited peer-review and artifact-evaluation permission in `source/RIGHTS.md`; it grants no general open-source, redistribution, commercial, or production-use license. Vendored third-party files retain their own terms.

From the repository root, `npm run check` verifies signatures, digest links, report derivations, published counts, negative controls, source-manifest and archive integrity, and anonymity and secret scans. The runbook documents the pinned-Rust source check and clean-room rerun. The release private key is not distributed.

References

[1] Henk Birkholz, Antoine Delignat-Lavaud, Cédric Fournet, Yogesh Deshpande, and Steve Lasker. 2026. *An Architecture for Trustworthy and Transparent Digital Supply Chains.* Request for Comments RFC 9943. Internet Engineering Task Force (IETF). doi:10.17487/RFC9943

[2] Henk Birkholz, Dave Thaler, Michael Richardson, Ned Smith, and Wei Pan. 2023. *Remote ATtestation procedureS (RATS) Architecture.* Request for Comments RFC 9334. Internet Engineering Task Force (IETF). doi:10.17487/RFC9334

[3] Cedar Policy Project. 2026. cedar-policy-symcc: Symbolic Cedar Compiler. Open-source Rust crate, https://github.com/cedar-policy/cedar/tree/cedar-policy-symcc-v0.5.3/cedar-policy-symcc. Compiles Cedar policies to SMT-LIB; compiler and verifiers are formally modeled and verified in Lean. Accessed 2026-07-15.

[4] Joseph W. Cutler, Craig Disselkoen, Aaron Eline, Shaobo He, Kyle Headley, Michael Hicks, Kesha Hietala, Eleftherios Ioannidis, John Kastner, Anwar Mamat, Darin McAdams, Matt McCutchen, Neha Rungta, Emina Torlak, and Andrew M. Wells. 2024. Cedar: A New Language for Expressive, Fast, Safe, and Analyzable Authorization. *Proceedings of the ACM on Programming Languages* 8, OOPSLA1, Article 118 (2024), 28 pages. doi:10.1145/3649835

[5] Norm Hardy. 1988. The Confused Deputy (or why capabilities might have been invented). *ACM SIGOPS Operating Systems Review* 22, 4 (1988), 36–38. doi:10.1145/54289.871709

[6] in-toto Project. 2026. in-toto Attestation Statement, Version 1.2.0. Specification, https://github.com/in-toto/attestation/blob/v1.2.0/spec/v1/statement.md. Accessed 2026-07-15.

[7] Yue Jia and Mark Harman. 2011. An Analysis and Survey of the Development of Mutation Testing. *IEEE Transactions on Software Engineering* 37, 5 (2011), 649–678. doi:10.1109/TSE.2010.62

[8] Simon Josefsson and Ilari Liusvaara. 2017. *Edwards-Curve Digital Signature Algorithm (EdDSA).* Request for Comments RFC 8032. Internet Engineering Task Force (IETF). doi:10.17487/RFC8032

[9] JsonLogic. n.d.. JSON Logic Documentation. Online documentation, https://jsonlogic.com. Accessed 2026-07-15.

[10] Ben Laurie, Eran Messeri, and Rob Stradling. 2021. *Certificate Transparency Version 2.0.* Request for Comments RFC 9162. Internet Engineering Task Force (IETF). doi:10.17487/RFC9162

[11] William M. McKeeman. 1998. Differential Testing for Software. *Digital Technical Journal* 10, 1 (1998), 100–107.

[12] Mark S. Miller, Ka-Ping Yee, and Jonathan S. Shapiro. 2003. *Capability Myths Demolished.* Technical Report SRL2003-02. Systems Research Laboratory, Johns Hopkins University. https://srl.cs.jhu.edu/pubs/SRL2003-02.pdf

[13] National Institute of Standards and Technology. 2015. *Secure Hash Standard (SHS).* Federal Information Processing Standards Publication FIPS PUB 180-4. National Institute of Standards and Technology. doi:10.6028/NIST.FIPS.180-4

[14] George C. Necula. 2000. Translation Validation for an Optimizing Compiler. In *Proceedings of the ACM SIGPLAN 2000 Conference on Programming Language Design and Implementation (PLDI '00).* ACM, Vancouver, British Columbia, Canada, 83–94. doi:10.1145/349299.349314

[15] Open Policy Agent Project. 2025. Open Policy Agent (OPA): Decision Logs. Online documentation, https://www.openpolicyagent.org/docs/management-decision-logs. Accessed 2026-07-15.

[16] OpenFisca. n.d.. OpenFisca: A Rules-as-Code Engine for Tax and Benefit Systems. Online documentation, https://openfisca.org/doc/index.html. Accessed 2026-07-15.

[17] OpenSSF. 2025. SLSA Build: Verifying Artifacts, Version 1.2. Specification, https://slsa.dev/spec/v1.2/verifying-artifacts. Accessed 2026-07-20.

[18] OpenSSF. 2025. SLSA Provenance, Version 1.2. Specification, https://slsa.dev/spec/v1.2/provenance. Accessed 2026-07-15.

[19] OpenSSF. 2025. SLSA Verification Summary Attestation, Version 1.2. Specification, https://slsa.dev/spec/v1.2/verification_summary. Accessed 2026-07-20.

[20] Amir Pnueli, Michael Siegel, and Eli Singerman. 1998. Translation Validation. In *Tools and Algorithms for the Construction and Analysis of Systems (TACAS 1998) (Lecture Notes in Computer Science, Vol. 1384)*, Bernhard Steffen (Ed.). Springer, Berlin, Heidelberg, 151–166. doi:10.1007/BFb0054170

[21] PolicyEngine. n.d.. PolicyEngine: Open-source Tax and Benefit Analysis. Project website, https://www.policyengine.org/us. Accessed 2026-07-15.

[22] Secure Systems Lab. 2024. Dead Simple Signing Envelope (DSSE) Protocol, Version 1.0.2. Protocol specification, https://github.com/secure-systems-lab/dsse/blob/v1.0.2/protocol.md.

[23] Yaron Sheffer, Dick Hardt, and Michael B. Jones. 2020. *JSON Web Token Best Current Practices.* Request for Comments RFC 8725. Internet Engineering Task Force (IETF). doi:10.17487/RFC8725

[24] Alex Shinn, John Cowan, and Arthur A. Gleckler. 2013. Revised^7 Report on the Algorithmic Language Scheme. Language report, https://standards.scheme.org/official/r7rs.pdf.

[25] Sigstore. n.d.. Rekor Transparency Log. Online documentation, https://docs.sigstore.dev/logging/overview/. Accessed 2026-07-15.

[26] Sigstore Project. 2026. Kubernetes Policy Controller. Online documentation, https://docs.sigstore.dev/policy-controller/overview/. Accessed 2026-07-20.

[27] Juraj Somorovsky, Andreas Mayer, Jörg Schwenk, Marco Kampmann, and Meiko Jensen. 2012. On Breaking SAML: Be Whoever You Want to Be. In *21st USENIX Security Symposium (USENIX Security 12).* USENIX Association, Bellevue, WA, 397–412.

[28] Styra, Inc. 2026. Styra DAS: Decision Log Replay and Impact Analysis. Product documentation, https://docs.styra.com/das/observability-and-audit/decision-logs/log-replay. Accessed 2026-07-15.