Proof-Carrying Bilateral Admission for Cross-Organization Agent Tool Calls

Anonymous Author(s)
Anonymous Institution

Abstract

A receiving kernel cannot decide whether to admit an agent action from another organization based on the vendor signature alone. A correctly signed receipt may still describe the wrong request, outcome, treaty, continuation, or receiver policy. We present receiver-owned bilateral admission, a protocol that checks these claims before the receiving kernel invokes a tool. The receiver loads treaty state and authorized keys from its own stores, verifies two distinct keys over the same canonical DSSE predicate, and binds that predicate to the request, outcome, treaty scope, continuation, lineage, and both parties' receipts. The Rust implementation rejects missing, stale, replayed, and mismatched claims before dispatch. We model the admission rules in Lean 4 over a bounded receipt view and prove treaty intersection, ladder-floor stability, and sound no-widening over finite domains. An independent Rust reference evaluator matched the production evaluator on exhaustive atom cases and 1,024 generated cases for each compound property. In a three-vendor SQLite loopback on an eight-core Arm server, full receiver admission took 2.288 s p50 and 2.428 s p99 over 20 runs. Pre-dispatch denial took 13405.539 μs p50 and 22826.188 μs p99 over 30 samples, with zero tool invocations. All 20 named negative cases passed.

1 Introduction

Consider a buyer agent that asks a vendor agent to stage a refund. The vendor returns a correctly signed receipt. The buyer must nevertheless reject it if the receipt belongs to another treaty, binds different arguments, descends from an expired continuation, omits required lineage, or records a policy verdict that the buyer does not recognize. Signature validity answers who possessed a key; it does not answer whether this receiver should admit this action. If the check occurs only in an audit pipeline, rejection arrives after the tool may have changed the vendor ledger.

This paper studies a narrower property: *receiver-owned admission*. For a request classified as cross-organizational, the receiving kernel must resolve treaty and trust material from receiver-controlled stores, verify that all evidence describes one invocation, and decide before tool dispatch. Missing or inconsistent context denies. Request metadata cannot install a trust root. A continuation is consumed once, and replay denies. The result is an authorization boundary rather than an after-the-fact explanation.

Five terms carry the construction. A *receipt* is a kernel-signed record of a mediated tool decision over canonical bytes. A *treaty* fixes participants, keys, action classes, ladder manifests, and a validity interval. A *bilateral predicate* is a canonical DSSE statement that binds two signatures to one request, outcome, treaty, continuation, lineage, and pair of receipt digests [15, 16]. A *receiver-owned trust store* supplies expected peer keys and the referenced artifacts independently of the request. *Pre-dispatch admission* is the receiver's final check before any tool invocation.

The hard case is not creating more signatures. It is making the evidence graph non-substitutable. A sender should not be able to replace the treaty while retaining a valid receipt signature, exchange the requested arguments for another payload, replay a live-looking continuation, or present two signatures over adjacent but unequal statements. Conversely, a verifier must not infer organizational independence from two keys. The strict profile proves possession of two distinct configured keys over the same bytes. Whether those keys are controlled by distinct organizations is a key-provisioning and attestation assumption.

We implement the admission path in Chio, a capability-mediated runtime for agent tool calls. Core kernel authorization and explicit treaty validators enforce the admission decision. A versioned bounded predicate syntax and a verified-artifact constructor supply the receipt view used for formal and differential evaluation. Unknown predicate versions and tags, oversized serialized inputs and atom strings, excess depth, and excess node count deny. We compare this evaluator with an independent reference implementation; the Rust code is not extracted from Lean.

For interpretation, let a receipt-bounded polity be $P =$

(T, K), where T is the closed scope of receipts under consideration and K is its admission rule. "Programmable sovereignty" names the receiver's local authority to decide membership in that receipt history. It does not denote legal statehood, territorial jurisdiction, or authority over another organization's systems.

This paper makes four contributions:

- **A strict bilateral predicate.** It binds request and outcome hashes, treaty scope, ladder intersection, continuation, lineage, receipt hashes, leases, governance references, and exactly two configured signer keys in one canonical DSSE envelope.
- **A production pre-dispatch hook.** The receiving kernel resolves evidence from its own stores, rejects request-supplied trust, consumes continuations once, and denies malformed, stale, replayed, or disagreeing evidence before dispatch.
- **Bounded semantics with executable alignment.** Lean 4 defines a finite predicate language over an explicit *ReceiptView*, proves structural treaty results and finite-domain no-widening, and proves pointwise agreement with identifier-indexed closure semantics. An independent Rust oracle differentially checks the production evaluator.
- **An end-to-end loopback evaluation.** A three-vendor SQLite scenario exercises runtime admission, strict DSSE, lineage, proof regeneration, and buyer review; 20 named negative cases and 50 separate generic replay fixtures provide adversarial and reproducibility evidence.

The central claim is deliberately asymmetric. Chio demonstrates that a receiver can refuse a mismatched cross-organization action at its own boundary and can retain evidence for that decision. It does not demonstrate that both endpoints are honest, separately administered, or legally bound by the same rule.

2 Security Context and Threat Model

Capability mediation. Capability systems replace ambient authority with explicit, attenuable references. KeyKOS and EROS made this discipline an operating-system primitive; Capsicum brought capability mode to UNIX; seL4 connected a microkernel authority model to machine-checked functional correctness [7, 9, 17, 22]. Object-capability work explains authority as reachability and attenuation rather than membership in a global access-control list [12, 13]. Chio applies that discipline to agent tool calls: the untrusted agent supplies a signed capability, while a trusted kernel validates it and mediates the tool server.

Agent isolation and cross-domain authority. IsolateGPT confines agent applications behind information-flow gates; SAGA distributes user-controlled authorization tokens; recent agent-security work identifies confused-deputy and authority-boundary failures as a central systems problem [2, 20, 23]. These systems primarily protect a local principal. Bilateral admission begins when the evidence crosses an administrative boundary and the receiver must decide whether a foreign action belongs to its own authorized history.

Signed statements are necessary but incomplete. DSSE authenticates a payload and predicate type, while in-toto and SLSA bind subjects to supply-chain provenance [16, 19, 21]. A valid provenance statement does not establish receiver authorization for a tool call. The receiver additionally needs freshness, request and outcome binding, participant and treaty binding, continuation state, and its own policy verdict. Chio retains DSSE's canonical signed envelope but defines an action-admission predicate rather than a build-provenance predicate.

Trusted components. We trust the receiving kernel, its cryptographic and canonical-JSON libraries, and its locally provisioned stores. The stores contain expected peer keys, treaty scope, ladder manifests, leases, governance receipts, revocation state, continuations, lineage, and resolved receipts. We assume Ed25519 and SHA-256 behave as specified and that the receiver classifies federated origin correctly. A compromised receiver can authorize anything and lies outside the property proved here.

Adversary. The adversary controls the agent request and may present arbitrary metadata and syntactically valid signed objects. It may replay old material; mix artifacts from different invocations; alter the treaty, request, outcome, receipt, lineage, continuation, or predicate type; omit a signature or governance record; use stale leases; duplicate a signer key identifier; or submit a correctly signed unanimous denial. It may control a remote vendor kernel and any request-carried trust data. The receiver must deny these cases before its tool runs.

The adversary may also control two private keys that the receiver has separately authorized. Cryptography cannot distinguish this condition from two administrative principals. We therefore test rejection of an identical repeated key but record distinct-controller independence as assumption PS-A-01. TEE-backed key custody or external organizational governance could strengthen that assumption, but neither is part of the evaluated construction.

Security objective. Let q be a federated tool request and $A_R(q)$ the receiving kernel's admission decision. The implementation objective is:

$$A_R(q) = \mathsf{allow} \implies \mathsf{coreAllows}(q) \wedge \mathsf{receiverEvidenceAccepts}(q).$$

The second predicate means that receiver-resolved treaty evidence is fresh, mutually bound, strictly verified, and accepted by receiver policy. If any premise fails, the receiver returns a denial before invocation. This is a local safety property. It neither forces the sender to record the same history nor establishes the truth of claims made by a compromised but authorized signer.

3 Receiver-Owned Bilateral Admission

The protocol separates evidence produced by the sender from authority retained by the receiver. A sender may supply receipts and a DSSE envelope. It may not supply the receiver's expected keys, treaty record, trust bundle, or continuation state. Those objects are resolved by identifier from receiver-owned stores.

3.1 Artifacts and bindings

Receipt. A Chio receipt records the capability identifier, tool server and name, canonical action hash, allow or deny decision, policy hash, evidence, metadata, trust level, and kernel public key. The body is signed over canonical JSON. A valid vendor receipt proves that the corresponding key signed that body; it is not a receiver verdict.

Treaty scope and ladder. A *TreatyScope* fixes a treaty identifier, participant kernel identifiers and public keys, hashes of participant ladder manifests, allowed action classes, validity interval, revocation epoch, and trust-bundle digest. Each ladder manifest assigns an enforcement mode and evidence requirement to an action class. The finite order is observation, guarded, receipt-backed, partition-contingency, and quorum-required. Intersection takes the stricter participant mode and rejects a destructive class below receipt-backed. An unknown class denies.

Continuation and lineage. A *CrossKernelContinuation* links child work to a parent receipt and session anchor, capability, action class, target tool, nonce, source and target kernels, and validity interval. A *ReceiptLineageStatement* binds the parent and child receipt digests to that continuation and to the bilateral invocation. The receiver reserves a continuation during admission, consumes it on the admitted path, and releases it when pre-dispatch execution aborts.

Bilateral predicate. The strict predicate type is `chio.bilateral-cosign-invocation.v1`. Its treaty reference binds the treaty scope and ladder-intersection digests; admission report; continuation; lineage bundle; action class and consistency model; request and outcome digests; local and remote receipt digests; lease and governance references; and participant kernel identifiers. DSSE signs the canonical statement through its PAE construction [16]. The envelope contains exactly two signatures whose key identifiers must be distinct and match the configured public keys.

Two qualifications are essential. First, both signatures cover the same predicate bytes; two nearby statements do not compose into bilateral evidence. Second, distinct keys do not establish distinct controllers. The verifier enforces the former. Deployment governance must establish the latter.

3.2 Admission sequence

Figure 2 shows the receiver's control flow.

1. The kernel first performs its ordinary capability, scope, revocation, budget, and guard checks.
2. Trusted runtime state classifies the request. A local request with no Chio context retains the local path. A request marked federated without a complete treaty context denies.
3. The hook parses only identifiers and bindings from request context. Any attempt to carry a trust root, peer directory, signing key, or dynamic trust bundle in request metadata denies.
4. Receiver stores resolve the treaty scope, ladder intersection, continuation, lineage, bilateral invocation, receipts, leases, governance material, peer pins, and revocation state.
5. Validators check schema, canonical hashes, freshness, participant sets, action class, consistency model, required evidence, continuation origin and target, and cross-artifact equality.
6. The strict DSSE verifier checks predicate type, canonical payload bytes, one subject, the embedded receipt, two distinct key identifiers, both signatures, policy agreement, lease and governance requirements, and every treaty binding.
7. Admission either returns a named failure before tool invocation or permits dispatch. On the admitted path, the runtime emits bound receipt metadata and assembles buyer-review evidence.

Receiver-owned policy. The hook enforces treaty policy through explicit Rust validators and the strict bilateral verifier. A versioned syntactic predicate evaluator consumes a verified-artifact receipt view (Section 4) for formal and differential evaluation. Its result does not authorize the hook's allow path. A missing or malformed predicate document therefore cannot bypass the explicit validators.

Failure evidence. The negative matrix assigns each adversary capability a stable identifier, enforcement phase, and expected diagnostic. Hook-level denial tests inspect the returned code and, where a dispatch surface exists, the tool invocation counter. Pure envelope-verifier cases have no tool reference

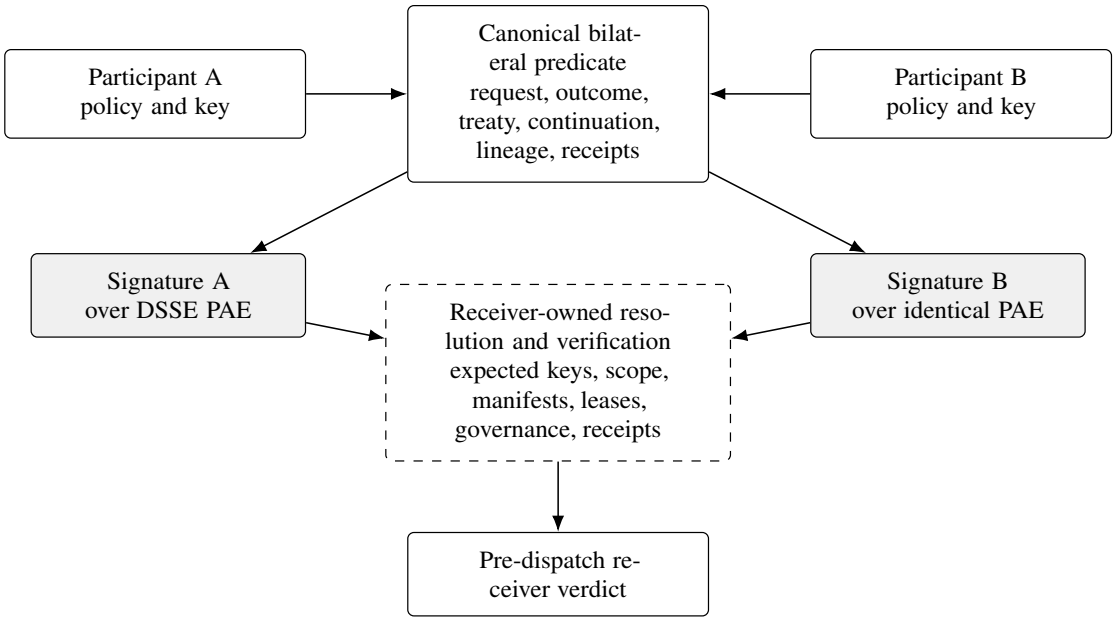

Figure 1: The bilateral artifact. Participant policy produces a common treaty-bound predicate; two configured keys sign the same DSSE PAE bytes. The receiver then resolves and checks every referenced object from its own trust context.

and fail before a dispatch-capable kernel API is entered. We do not claim that every internal exception in the wider Chio runtime emits the same durable signed denial artifact; the claim is confined to the named admission and receipt paths.

Receipt-bounded polity. The protocol admits a restrained institutional interpretation. A polity is $P = (T, K)$: T identifies the receipts whose membership is at issue, and K decides admission. Bilateral relations are conjunctions of treaty and participant predicates. This vocabulary describes local control over a receipt namespace. It adds no claim about population, territory, courts, or legal recognition.

4 Bounded Formal Model

The formal development isolates the admission logic from the mechanisms that construct its inputs. It does not model cryptography, canonical serialization, clock correctness, network transport, durable storage, or organizational key custody. Those mechanisms must establish the fields of a bounded receipt view before any theorem in this section applies.

4.1 Receipts, Predicates, and Polities

A receipt view is the tuple

$$r = (id, hash, action, participants, \\ modeRank, liveContinuations, decision, \\ failureCode, evidenceDigests).$$

The fields are finite data: strings, natural numbers, lists, and an allow-or-deny decision. In particular, denial is represented rather than inferred from the absence of an allow. A predicate can therefore require a specific denial code or an evidence digest just as it can require an action class or participant.

The syntax is

$$p ::= \top \mid \bot \mid \mathsf{atom}(a) \mid p \wedge p \mid p \vee p \mid \neg p.$$

Atoms inspect exactly one of the following relations: receipt identity, receipt hash equality, participant-kernel membership, action-class equality, ladder rank at least a constant, live-continuation membership, decision equality, failure-code equality, or evidence-class and digest equality. The interpreter $\mathsf{denote}(p, r)$ is total and Boolean. This restriction is intentional. It gives every production-named atom a concrete meaning without pretending that implication between arbitrary host-language closures is decidable.

A constitution $K = [p_1, \ldots, p_n]$ admits r when every predicate does:

$$\mathsf{admits}(K, r) = \bigwedge_{p \in K} \mathsf{denote}(p, r).$$

A polity is the pair $P = (T, K)$, where T is a predicate defining the local receipt-admission scope. Thus

$$\mathsf{polityAdmits}(P, r) = \mathsf{denote}(T, r) \wedge \mathsf{admits}(K, r).$$

The model contains no membership roster, territorial jurisdiction, or authority over another log. Its subject is one receiver's classification of receipts at its own boundary.

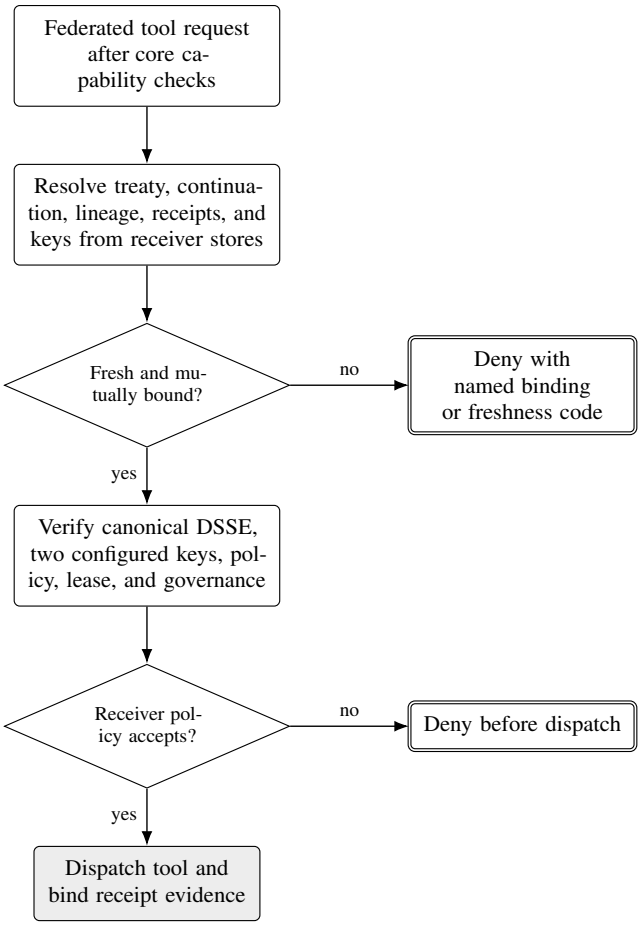

Figure 2: Receiver-side pre-dispatch admission. Trust material flows from receiver-owned stores, not from the request. Every denial terminal precedes the tool boundary.

4.2 Bilateral Intersection

A bilateral treaty $\tau = (T_\tau, K_\tau, P_L, P_R, f_\tau)$ contains treaty scope and constitutional predicates, the two participant polities, and a minimum ladder mode f_τ. Admission is defined by the grouped expression

$$\begin{aligned}\mathsf{treatyAdmits}(\tau, r) = {} & \mathsf{denote}(T_\tau, r) \\ & \wedge \mathsf{admits}(K_\tau, r) \\ & \wedge \mathsf{polityAdmits}(P_L, r) \\ & \wedge \mathsf{polityAdmits}(P_R, r).\end{aligned}$$

`treaty_admission_iff_predicate_intersection` proves that this expression is equivalent to its fully expanded six-conjunct form. The theorem is structural, but that is precisely its role: neither participant's scope or constitution disappears when the predicates are regrouped for evaluation. It says nothing about whether Rust constructed the intended receipt view or verified the signatures that bind it.

Mode admission prepends a finite ladder check:

$$\mathsf{underMode}(\tau, m, r) = \mathsf{atLeast}(m, f_\tau) \wedge \mathsf{treatyAdmits}(\tau, r).$$

`treaty_admission_stable_under_ladder_floor` proves that, under the hypothesis $\mathsf{atLeast}(m, f_\tau) = \mathsf{true}$, mode admission reduces to treaty admission. A mode below the floor denies. The theorem is bounded to the ladder representation imported by the model.

4.3 Finite-Domain No-Widening

For constitutions K', K, and an explicitly declared finite domain D, define

$$\mathsf{Refines}_D(K', K) \equiv \forall r \in D.\ \mathsf{admits}(K', r) \Rightarrow \mathsf{admits}(K, r).$$

The executable checker evaluates this implication for every member of D. `refinesOnConstitution_iff` proves that the Boolean result is equivalent to $\mathsf{Refines}_D$, and `bridge_decidable_soundness` exposes its soundness direction. A `ConstitutionalDelta` carries K, K', D, and the successful check as a construction-time witness. The development contains both a nontrivial narrowing that constructs such a delta and a widening candidate that evaluates to false.

This property is no-widening on D. It does not establish refinement outside D, require K' to preserve every receipt admitted by K, or justify a claim about all possible future schemas. A narrower K' can reject receipts admitted by K while satisfying the theorem. The Rust runtime has no amendment-enactment path consuming `ConstitutionalDelta`; amendment is therefore a model-level extension, not a production contribution.

4.4 Bridges and Assurance Boundary

A second Lean representation uses opaque predicates over receipt identifiers. Given an explicit resolver from an identifier to a `ReceiptView`, `toClosure` lifts a syntactic constitution into that representation. `bridge_pointwise`, `treaty_admission_agrees`, and `treaty_admission_under_mode_agrees` prove pointwise agreement after the lift. They do not turn a finite domain into a universal one.

The Rust predicate evaluator was written independently from the Lean interpreter. Differential tests apply a mechanical conversion from generated Rust values into a second Rust specification that mirrors the Lean definitions. They cover every atom form and 1,024 generated cases each for compound predicates, constitutional admission, and finite refinement. This is useful evidence against representation drift. It is neither extraction from Lean nor a proof of equivalence between Lean and production Rust.

All listed theorems are imported by the proof root, inventoried, and checked without `sorry` or added axioms. Their assurance boundary ends at the explicit data structures above.

Table 1: Formal results and their implementation status.

Result	Model claim	Implementation relation
`treaty_admission_iff_predicate_intersection`	Six bounded predicates remain conjunctive	Production counterpart plus differential cases
`treaty_admission_stable_under_ladder_floor`	A satisfied finite floor reduces to treaty admission	Production ladder validator counterpart
`refinesOnConstitution_iff`	Boolean no-widening is exact on declared D	Model only
`treaty_admission_agrees`	Syntactic and identifier-indexed forms agree pointwise	Model only

The implementation evidence for constructing and enforcing those structures is independent and is described next.

5 Implementation

The implementation follows the receiver's control flow. A core kernel first validates ordinary capability authority. Federated context then enters a runtime admission hook, which parses request metadata, resolves referenced artifacts from receiver-owned stores, validates treaty and continuation state, verifies the bilateral statement, and either returns a named denial or permits tool dispatch. The runtime harness assembles the resulting receipts into a proof package that an offline buyer verifier can replay.

5.1 Receiver-Owned Resolution

`ChioRuntimeAdmissionHook::evaluate` is the production boundary. Requests with no Chio context retain the local compatibility path. Once Chio context is present, malformed or incomplete context denies. In particular, `treaty_ref_from_request` rejects request-carried trust roots, peer directories, signing keys, revocations, and dynamic trust bundles. The hook accepts identifiers for such material, then resolves the material through its own `RuntimeAdmissionStores`.

Resolution precedes policy evaluation. The receiver loads treaty scope, participant manifests, pinned keys, revocation state, continuation state, governance receipts, and referenced execution receipts. It checks schema and semantic hashes, validity intervals, participant uniqueness, action-class coverage, and the receiver's minimum ladder mode. Reserved continuation state is consumed for an admitted dispatch and released after denial or abort. Consequently, a sender cannot upgrade its authority by attaching a more permissive policy bundle to the request.

5.2 Bilateral Statement

`verify_chio_bilateral_invocation` verifies a strict DSSE profile. Both signatures cover the same pre-authentication encoding of one canonical statement. The statement binds the invocation identifier, treaty identifier, ladder-intersection hash, continuation hash, action and consistency classes, capability identifier, request and outcome hashes, lineage, local and remote receipt hashes, and signer identifiers. The envelope verifier rejects an unknown payload type, an incorrect signature count, duplicate key identifiers, noncanonical JSON, an unrecognized statement or predicate type, missing subjects, and reuse of one public key.

The operational verifier then checks those authenticated names against receiver-owned state: configured peer pins, live manifests, revocation epoch, receipt signatures and hashes, capability lease, policy verdicts, and the governance evidence required by receipt-backed action classes. The `BilateralCoSigningError::code` mapping assigns stable diagnostic codes to failures that cross the artifact boundary. Distinct configured keys establish two-key authorization. Organizational independence is an operational assumption because one actor can control both keys.

5.3 Pre-Dispatch Enforcement

The admission hook runs after core capability checks and before the `ToolServer` invocation. It treats parser, store, freshness, cryptographic, lineage, policy, and continuation errors as denials. The dispatch-path test server increments a counter only on invocation, and denial samples assert that it stays unchanged. Direct verifier tests make no dispatch observation.

The hook authorizes through its explicit checks. The versioned bounded predicate module `chio.federation.bounded-treaty-predicate.v1` denies unknown versions, tags, and fields and enforces a 64 KiB serialized-input limit, a 1 KiB atom-string limit, and depth and node limits of 32 and 1,024. `bounded_treaty_receipt_view_from_verified_artifacts` requires an independently authenticated canonical report digest, checks that the action is inside the treaty scope, and cross-checks report, invocation, and continuation bindings before constructing the evaluator's input. The module supports differential testing and a restricted library interface. Its result is not an alternative production allow path.

5.4 Runtime and Offline Review

`run_runtime_loopback_scenario` drives a deterministic three-party loopback through the real runtime path and persistent SQLite receipt stores. It produces the treaty scope, ladder intersection, continuation, lineage, bilateral DSSE statement, local and remote receipts, and buyer audit package. The producer-only mode stops after generating and schema-validating those artifacts. The full mode invokes the buyer CLI, validates the package schema, and performs semantic review. The offline `verify_package` path re-resolves the linked artifacts and checks their hashes and signatures without access to vendor internals.

The negative matrix uses the same public verification surfaces. Each selected test emits its observed diagnostic for comparison with the corpus. Envelope cases call the strict verifier; store and policy cases call the admission hook directly. None instantiates a tool server, so the matrix demonstrates named denials while a separate dispatch-path test supplies the counter observation.

The proof root, theorem inventory, differential corpus, runtime tests, matrix, benchmark outputs, and source snapshot are tied together by the supplementary artifact manifest. The manifest names symbols and files rather than source line numbers, which makes movement within a file harmless while still failing if a load-bearing surface disappears.

6 Evaluation

We ask four questions. RQ1 tests whether the implementation rejects the attacks implied by the threat model before dispatch. RQ2 tests whether the bounded Lean and Rust semantics remain aligned. RQ3 measures the complete receiver path and its principal components. RQ4 examines replayability and artifact size.

6.1 Method

Experiments ran on Linux/aarch64 with eight Arm Neoverse-N1 cores and 49.2 GB RAM, using rustc 1.93.0 and Cargo 1.93.0. All reported bilateral measurements use the release profile. Each result document records the tested commit and whether the source tree was clean when the experiment began; the artifact manifest pins the result and source hashes. The end-to-end harness uses persistent SQLite stores on one host. It includes process startup, artifact production, CLI execution, schema validation, and semantic review, rather than approximating those costs with an in-process function.

`run-bilateral-admission.sh` performs two warmups, 20 end-to-end samples per path, and 30 Criterion samples per component. It retains raw CSV, machine and toolchain metadata, a JSON summary, and generated TEX input. `run-replay-corpus.sh` reports the generic replay suite and bilateral matrix separately. No result below is transcribed from a console log.

6.2 RQ1: Do Attacks Deny Before Dispatch?

The corpus contains 20 stable cases, `PS-TH-01` through `PS-TH-20`. Each entry names the exact test target, expected failure code, and expected phase. The matrix runner rejects duplicate identifiers, empty selections, unexpected codes, and any entry whose expected dispatch value is not false.

All 20 cases returned their named denial in both matrix runs. The runner compares each test's machine-readable code with the corpus, so a wrong diagnostic fails. These targets call validators or the admission hook without creating a tool server; the matrix claim is therefore the named rejection at the declared phase. A separate dispatch-capable benchmark observed an unchanged invocation counter. The release matrix took 159.5 s, a reproducibility wall time, not denial latency.

The corpus also declares `PS-A-01`: one actor may control two distinct keys that the receiver has authorized as the treaty participants. Cryptography cannot distinguish that deployment from two organizations. We record it as a non-testable operational assumption rather than a passing attack case.

6.3 RQ2: Are the Bounded Semantics Aligned?

The Lean gate rebuilds the root import, rejects `sorry` and unregistered axioms, and checks the theorem inventory. Independent Rust tests exercise every atomic predicate form. They then run 1,024 generated cases for each of three properties: compound predicate denotation, constitutional admission, and finite-domain refinement. The Rust specification, the production bounded evaluator, and the mechanical view conversion agreed on all tested cases.

This result detects many representation and Boolean-composition errors. It does not prove the Rust code equivalent to Lean, establish completeness of the generator, or verify the runtime admission hook. The production tests and RQ1 cover the latter boundary independently.

6.4 RQ3: What Does Admission Cost?

Full receiver admission had a 2.288 s median and 2.428 s p99. The path is dominated by orchestration, process, validation, and storage work rather than the bilateral cryptographic check. The two end-to-end distributions differ by 304.321 ms at the median, but that difference is not a pure buyer-verification ablation: the full path also invokes the CLI and performs additional schema and semantic checks. The isolated offline verifier median was 49.875 ms.

The real receiver-owned pre-dispatch denial path was 13405.539 μs at p50 and 22826.188 μs at p99. Its timed body traverses the kernel capability, federation, and persistence gates; resolves the stored treaty scope, ladder intersection, continuation, lineage, invocation, and bilateral DSSE; verifies the signed policy denial; and persists the signed SQLite denial receipt. The instrumented tool invocation counter remains unchanged. Receipt signing measured 242.422 μs at the median, strict bilateral DSSE verification measured 340.225 μs, and persistent SQLite append measured 7011.802 μs. These component measurements use production-shaped data and production functions, but they do not sum to the end-to-end result because the full loop includes work absent from the microbenchmarks.

Table 2: Load-bearing artifacts and the boundary of each claim.

Symbol or artifact	Role	Evidence boundary
`ChioRuntimeAdmissionHook::evaluate`	Receiver-owned resolution and pre-dispatch verdict	Production Rust, focused tests, live loopback
`verify_chio_bilateral_invocation`	Strict DSSE and operational binding verification	Two configured keys; organizational control is assumed
`CrossKernelContinuation`	Parent receipt, capability, nonce, action, target, and lifetime binding	Replay safety depends on receiver store correctness
`bounded_treaty_receipt_view_from_verified_artifacts`	Constructs the restricted syntactic evaluator view	Library and differential target, not a replacement hook
`run_runtime_loopback_scenario`	Producer execution and proof-package assembly	Deterministic single-host three-party case study
`verify_package`	Offline receiver-side replay	Verifies the emitted package, not remote process integrity
`PS-TH-01`–`PS-TH-20`	Named bilateral negative corpus	Executable cases plus one separate non-testable assumption

Table 3: Bilateral negative matrix. All 20 cases passed.

Attack family	Cases	Rejected evidence
Binding substitution	01, 03, 05–07	Treaty, intersection, receipt, request, outcome
Signature and syntax	08–12, 18	Missing or duplicate signature, reused key, predicate, canonical form, schema
Freshness and continuity	02, 13, 15	Treaty, lease, continuation replay
Receiver trust and evidence	04, 14, 16, 17	Lineage, governance, smuggled trust
Policy outcome	19, 20	Disagreement and unanimous deny

6.5 RQ4: Replay and Evidence Volume

The emitted buyer proof package was 51,843 bytes (50.6 KiB). The replay script validated 50 generic kernel-capability fixtures in 3 s and, in a separate run, the 20 bilateral negative cases in 25 s. The first corpus covers generic capability verdict stability; it is not evidence for bilateral admission. The second covers the named failures in Table 3. Reporting them separately prevents the larger generic count from inflating the treaty evaluation.

Public-anchor inclusion is withheld. The protocol can carry anchor evidence, but this evaluation does not measure a live transparency service or claim that one is required for local admission.

6.6 Threats to Measurement Validity

The deterministic, single-host loopback removes network variance and makes artifact comparison reproducible, but it does not estimate wide-area latency or behavior under partial failure. 20 path samples and 30 component samples characterize this machine, not a deployment population. Criterion percentiles describe benchmark samples rather than service-level objectives. The retained result metadata records source cleanliness before execution. A clean reconstruction gate in the artifact package verifies the precise source set used here.

7 Discussion

The meaning of programmable sovereignty. The system grants no sovereignty in public or international law. It gives a receiver final technical authority over one narrow object: whether a foreign agent call crosses its tool-dispatch boundary and enters its receipt history. Within that boundary, the receiver selects the trust stores, resolves the evidence, applies the predicates, and records the outcome. We call this *receipt-bounded sovereignty*. The modifier is essential because another organization remains free to reject the same evidence under its own policy.

Why the receiver owns resolution. A signature authenticates bytes, not the authority used to interpret them. If the sender can supply the peer keys, revocation state, treaty manifest, or policy bundle, a perfectly valid signature can authenticate a self-appointed trust domain. Receiver-owned resolution makes those references capabilities to local state rather than serialized policy arguments. The resulting asymmetry is desirable: bilateral signing establishes agreement on one statement, while each side retains unilateral control of admission.

Two keys and two organizations. The protocol proves possession of two distinct authorized private keys over identical bytes. It cannot prove that independent organizations controlled the keys or evaluated policy independently. Deployments that require that stronger claim need identity governance, separated key custody, or attested execution. The paper keeps this distinction visible because collapsing it would turn an operational assumption into a cryptographic theorem.

Amendment as a bounded extension. Finite-domain no-widening illustrates how a polity could require a checkable predicate relation before accepting a candidate constitution. The theorem is useful for configuration sets whose admissible receipt domain is enumerated. It is not a production amendment protocol and does not guarantee that every receipt admitted by the current constitution stays admitted. A real amendment mechanism would need domain versioning, witness transport, enactment and rollback state, and an explicit procedure for discontinuous changes. Those requirements

Table 4: Release-profile results. End-to-end paths use 20 samples; components use 30.

Measured path	p50	p99	Included work
Full receiver admission	2288.145 ms	2428.387 ms	Producer, SQLite, buyer CLI, schema and semantic review
Producer without buyer review	1983.824 ms	2054.601 ms	Runtime loopback, SQLite, package production and schema validation
Pre-dispatch treaty denial	13405.539 μs	22826.188 μs	Full kernel path; real hook and SQLite denial receipt
Receipt sign	242.422 μs	309.318 μs	Buyer-shaped Ed25519 receipt
Receipt verify	370.163 μs	476.491 μs	Buyer-shaped Ed25519 receipt
Strict bilateral DSSE verify	340.225 μs	366.795 μs	Two signatures and embedded treaty bindings
Cross-boundary admission allow	22.070 μs	28.077 μs	Receiver treaty scope and verified evidence
SQLite receipt append	7011.802 μs	10477.089 μs	Production receipt store; signing outside timed body
Offline buyer-package verify	49875.059 μs	52293.652 μs	Three-party proof-package verification

are deliberately outside the demonstrated bilateral admission path.

Evidence after enforcement. The same bindings that make a decision enforceable also make it reviewable. The buyer package is therefore a consequence of the admission design, not a substitute for it. Offline verification can show that the package is internally consistent and authorized by its keys; it cannot recover a compromised receiver kernel, prove remote process integrity, or compel a third party to recognize the result.

8 Related Work

Capabilities and agent isolation. KeyKOS, EROS, object-capability systems, Capsicum, and seL4 established explicit, attenuable authority and small enforcement boundaries [7, 9, 12, 13, 17, 22]. Chio applies that discipline to agent tool calls and retains a signed artifact of the admission decision. IsolateGPT separates agent applications behind information-flow gates, while SAGA distributes user-controlled authorization tokens [20, 23]. Recent analyses identify confused-deputy and authority-boundary failures as central risks in agentic systems [2]. Receiver-owned bilateral admission is complementary: it governs the call after authority crosses an organizational boundary.

Authenticated evidence. DSSE authenticates a typed payload without making its internal policy semantics universal [16]. in-toto and SLSA use signed statements to describe software supply-chain provenance [19, 21]; transparency logs make authenticated statements auditable over time [10, 18]. Our bilateral statement instead binds an invocation and the receiver's admission evidence. It can cite supply-chain provenance, but provenance alone does not establish a live treaty, continuation, policy verdict, or receipt binding.

Verified authorization. Cedar combines a formal policy semantics with a production Rust evaluator and differential testing [3, 5]. Large verified systems such as seL4, IronFleet, and Project Everest connect stronger implementation claims to explicit refinement arguments [8, 9, 14]. Our Lean 4 model [4] is intentionally weaker: it proves properties of a bounded syntax and checks an independent Rust implementation by differential tests. We do not claim refinement of the runtime.

Cross-domain agreement. PBFT and federated Byzantine agreement establish replicated decisions under quorum assumptions [1, 11]. IBC verifies commitments across consensus zones [6]. Bilateral admission solves a different problem. It does not converge histories or provide consensus; it lets one receiver decide whether a statement co-signed under two policy contexts is sufficient for one local dispatch. A consensus or transparency layer can strengthen later audit without becoming part of the local safety condition.

9 Limitations

The formal results are bounded to `ReceiptView`, the stated atom language, and explicit finite domains. They abstract cryptography, canonicalization, clocks, and storage. Generated differential cases provide alignment evidence but no completeness guarantee, proof extraction, or whole-runtime verification.

The syntactic Rust evaluator is strict and resource-bounded, but the production hook still uses explicit admission checks as its authoritative policy. There is no production amendment path consuming a finite refinement witness. Schema evolution, cross-implementation interoperability, and negotiation between different predicate versions remain open protocol work.

Two valid signatures prove control of two configured keys. They do not prove two organizations, independent policy evaluation, uncompromised key custody, or execution inside an attested kernel. HSM-, KMS-, or TEE-backed keys could strengthen this assumption, but those deployments are not evaluated.

The evaluation is deterministic and single-host. It does not exercise wide-area transport, mTLS setup, partitions, concurrent load, large receipt stores, hardware security modules, trusted execution environments, or crash recovery during continuation consumption. Its 20 end-to-end and 30 component

Table 5: Assumptions outside the bounded Lean model.

Assumption	Required for	Failure consequence
Ed25519, SHA-256, and DSSE behave as specified	Signature, digest, and envelope integrity	Forgery or ambiguous identity invalidates evidence binding
Canonical JSON matches RFC 8785	Stable statement and receipt bytes [15]	Equivalent data can acquire inconsistent identities
Receiver kernel and stores are trustworthy	Correct resolution, clocks, continuation state, and non-dispatch	A compromised receiver can admit, suppress, or misrecord calls
Authorized keys have the intended organizational custody	Interpretation of bilateral control	One actor with two keys satisfies the cryptographic profile

samples are too small for operational tail-latency claims. The measured path uses SQLite and includes process and schema-validation costs specific to the harness.

Not every internal runtime decision is encoded as the same signed bilateral artifact. Hook-level denials have stable failure codes and observable non-dispatch; verifier-only failures terminate before a runtime dispatch surface exists. The paper therefore claims auditable named failures where the tested path produces them, not that every error in the system necessarily becomes a universally signed denial receipt.

The generic 50-fixture replay corpus and the 20-case bilateral corpus establish different properties. Neither proves coverage of an unbounded adversary. Public-anchor inclusion is withheld, and no live transparency service is needed for the local admission result. The buyer proof package establishes internal artifact consistency, not remote process integrity or public non-equivocation.

Finally, receipt-bounded sovereignty is a technical metaphor with an explicit limit. The mechanism cannot confer legal jurisdiction, corporate authority, regulatory accreditation, territorial rights, or recognition by an external institution. It can produce evidence for such institutions, which remain free to reject or reinterpret it.

10 Conclusion

A cross-organization agent call should not run merely because a vendor signed a receipt. The receiving organization must decide which keys, treaty, continuation, policy, and prior receipts give that signature meaning. Chio implements this decision as a receiver-owned, fail-closed hook before tool dispatch and binds the result into a replayable bilateral package.

The evidence supports a deliberately bounded result. 20 named attacks deny, the pre-dispatch denial path leaves the tool uninvoked, a deterministic three-party receiver path completes with persistent stores, and an independent Rust evaluator agrees with a Lean model over the tested bounded domain. The Lean theorems establish intersection, ladder reduction, and finite-domain no-widening for that model; they do not verify the Rust runtime. Within these limits, programmable sovereignty denotes a concrete systems property: local, auditable authority over whether a foreign agent action crosses a receiver's own execution boundary.

References

[1] Miguel Castro and Barbara Liskov. Practical byzantine fault tolerance. In *Proceedings of the Third Symposium on Operating Systems Design and Implementation*, pages 173–186, Berkeley, CA, USA, 1999. USENIX Association.

[2] Mihai Christodorescu, Earlence Fernandes, Ashish Hooda, Somesh Jha, Johann Rehberger, Kamalika Chaudhuri, Xiaohan Fu, Khawaja Shams, Guy Amir, Jihye Choi, Sarthak Choudhary, Nils Palumbo, Andrey Labunets, and Nishit V. Pandya. Systems security foundations for agentic computing. arXiv:2512.01295, 2025.

[3] 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 Wells. Cedar: A new language for expressive, fast, safe, and analyzable authorization. *Proceedings of the ACM on Programming Languages*, 8(OOPSLA1):123:1–123:30, 2024.

[4] Leonardo de Moura and Sebastian Ullrich. The lean 4 theorem prover and programming language. In *Automated Deduction – CADE 28*, pages 625–635, Cham, Switzerland, 2021. Springer.

[5] Craig Disselkoen, Aaron Eline, Shaobo He, Kyle Headley, Michael Hicks, Kesha Hietala, John Kastner, Anwar Mamat, Matt McCutchen, Neha Rungta, Bhakti Shah, Emina Torlak, and Andrew Wells. How we built cedar: A verification-guided approach. arXiv:2407.01688, 2024.

[6] Christopher Goes. The Interblockchain Communication Protocol: An overview. arXiv:2006.15918, 2020.

[7] Norman Hardy. Keykos architecture. *ACM SIGOPS Operating Systems Review*, 19(4):8–25, 1985.

[8] Chris Hawblitzel, Jon Howell, Manos Kapritsos, Jacob R. Lorch, Bryan Parno, Michael L. Roberts, Srinath Setty, and Brian Zill. Ironfleet: Proving practical distributed systems correct. In *Proceedings of the 25th Symposium on Operating Systems Principles*, pages 1–17, New York, NY, USA, 2015. ACM.

[9] Gerwin Klein, Kevin Elphinstone, Gernot Heiser, June Andronick, David Cock, Philip Derrin, Dhammika Elkaduwe, Kai Engelhardt, Rafal Kolanski, Michael Norrish, Thomas Sewell, Harvey Tuch, and Simon Winwood. sel4: Formal verification of an os kernel. In *Proceedings of the 22nd ACM Symposium on Operating Systems Principles*, pages 207–220, New York, NY, USA, 2009. ACM.

[10] Ben Laurie, Emilia Messeri, and Rob Stradling. Certificate transparency version 2.0. RFC 9162, 2021.

[11] David Mazieres. The stellar consensus protocol: A federated model for internet-level consensus. Technical report, Stellar Development Foundation, San Francisco, CA, USA, 2015.

[12] Mark S. Miller. *Robust Composition: Towards a Unified Approach to Access Control and Concurrency Control*. PhD thesis, Johns Hopkins University, 2006.

[13] Toby Murray. *Analysing the Security Properties of Object-Capability Patterns*. PhD thesis, University of Oxford, 2010.

[14] Project Everest. Project everest: Verified secure implementations for the HTTPS ecosystem, 2017.

[15] Anders Rundgren, Bret Jordan, and Samuel Erdtman. Json canonicalization scheme (jcs). RFC 8785, 2020.

[16] Secure Systems Lab. DSSE: Dead simple signing envelope (v1.0.2), 2024.

[17] Jonathan S. Shapiro, Jonathan M. Smith, and David J. Farber. Eros: A fast capability system. In *Proceedings of the Seventeenth ACM Symposium on Operating Systems Principles*, pages 170–185, New York, NY, USA, 1999. ACM.

[18] Sigstore Project. Sigstore security model, 2021.

[19] SLSA Framework. Supply-chain levels for software artifacts specification (v1.0), 2023.

[20] Georgios Syros, Anshuman Suri, Jacob Ginesin, Cristina Nita-Rotaru, and Alina Oprea. SAGA: A security architecture for governing AI agentic systems. In *33rd Network and Distributed System Security Symposium (NDSS)*, pages 1–19, Reston, VA, USA, 2026. Internet Society.

[21] Santiago Torres-Arias, Hammad Afzali, Trishank Karthik Kuppusamy, Reza Curtmola, and Justin Cappos. in-toto: Providing farm-to-table guarantees for bits and bytes. In *28th USENIX Security Symposium*, pages 1393–1410, Berkeley, CA, USA, 2019. USENIX Association.

[22] Robert N. M. Watson, Jonathan Anderson, Ben Laurie, and Kris Kennaway. Capsicum: Practical capabilities for unix. In *19th USENIX Security Symposium*, pages 29–46, Berkeley, CA, USA, 2010. USENIX Association.

[23] Yuhao Wu, Franziska Roesner, Tadayoshi Kohno, Ning Zhang, and Umar Iqbal. IsolateGPT: An execution isolation architecture for LLM-based agentic systems. In *32nd Network and Distributed System Security Symposium (NDSS)*, pages 1–18, Reston, VA, USA, 2025. Internet Society.