

# Monotonic Narrowing for Agent Authority: Four Attenuation Invariants for Autonomous AI Governance

Tymofii Pidlisnyi

AEOESS (Agentic Economy Orchestration Engine for Sovereign Systems) [aеоess.com](https://aеоess.com) | [github.com/aеоess](https://github.com/aеоess) March 2026

## Abstract

As large language model (LLM) agents gain the ability to take real-world actions on behalf of humans, the question of how to bound their authority becomes urgent. We present the Agent Passport System, an open protocol that generalizes capability attenuation into a unified design law for autonomous AI governance. The protocol instantiates four attenuation invariants across distinct governance domains — with full cryptographic enforcement for delegation and governance attenuation, Merkle-based selective disclosure for disclosure attenuation, and a conservative human-authorized v1 form for exception attenuation: (1) delegation attenuation, where authority narrows monotonically across delegation chains; (2) governance attenuation, where policy artifacts can only strengthen and weakening requires higher-order authorization; (3) disclosure attenuation, where audit views reveal only the minimum necessary subset of committed data; and (4) exception attenuation, where temporary escalation is bounded by pre-committed ceilings with hard time-to-live.

We show that these four invariants are not independent properties but mutually reinforcing: governance weakening is itself treated as a bounded exception, subject to exception attenuation. The framework is self-referential by design. The protocol provides Ed25519 cryptographic identity, scoped delegation chains with cascade revocation, Merkle-tree beneficiary attribution, signed agent communication, a three-signature policy chain, coordination primitives, governance artifact provenance, bounded escalation, and agentic commerce gates.

We specify the protocol using formally stated invariants over an abstract state model, enforce them through a gateway that serves as an external reference monitor (with enforcement maturity varying across invariants), and validate the implementation with 866 unit and adversarial tests across 35 protocol modules. We do not claim machine-checked proof of implementation correctness. The system is implemented in TypeScript (866 tests, 239 suites) and Python (86 tests), published as open-source SDKs with a 61-tool MCP server. We map the protocol against the OWASP AIVSS risk taxonomy with honest coverage assessment (5 strong, 3 partial, 2 weak), present adversarial evaluation scenarios including expected failures, and identify 10 remaining open problems. All empirical results refer to SDK v1.16.1 (commit c1842eb). The protocol is under active development; the live test suite may differ from the frozen artifact evaluated here.

## 1. Introduction

The deployment of LLM-based agents capable of executing code, making purchases, sending messages, and coordinating with other agents creates a class of security problems that existing access control frameworks were not designed to handle. Unlike traditional software principals, LLM agents are non-deterministic, prompt-sensitive, and capable of generating novel action sequences that were never explicitly programmed. A compromised or misaligned agent with broad authority can cause damage proportional to its scope, and that damage may be difficult to detect until after the fact.

This problem is not new in computer science. The principle of least privilege dates to Saltzer and Schroeder (1975). Capability-based security, originating with Dennis and Van Horn (1966), established that delegated capabilities should be attenuatable but not amplifiable. The contribution of this paper is not mathematical novelty. It is the generalization of capability attenuation into a protocol-wide design law for LLM-based agent systems, implemented with cryptographic enforcement, tested against adversarial scenarios, and shown to compose across four distinct governance domains.

The necessity of protocol-level enforcement became apparent during preliminary research. In a 56-run experiment comparing single-agent and dual-agent workflows, prompt-level interventions intended to improve agent behavior were confounded by prompt length effects. This motivated a pivot from prompt-level governance to cryptographic enforcement: rather than asking agents to self-regulate through instructions, we constrain what they can do through infrastructure external to their reasoning.

The protocol is organized into 35 modules across eight governance domains. Together, these modules enforce a central thesis: four attenuation invariants operate across delegation, governance, disclosure, and exception handling, and these invariants mutually reinforce rather than merely coexist. Governance weakening is treated as an exception to governance attenuation and is therefore subject to exception attenuation. The framework is self-referential.

**Scope and claims.** This paper presents a formally specified and tested protocol, not a formally verified one. We state invariants using set notation and pseudocode, implement them in TypeScript and Python, and validate them through 866 tests including adversarial scenarios. We do not provide machine-checked proofs (e.g., in TLA+ or Alloy). The strongest guarantees hold when all privileged effects are mediated by the ProxyGateway enforcement boundary, which serves as an external reference monitor. When agents use the SDK voluntarily without the gateway, guarantees are conditional on agent cooperation.

**Paper structure.** Section 2 presents the threat model. Section 3 formalizes the four attenuation invariants and their composition. Section 4 describes the architecture. Section 5 reports evaluation results. Section 6 surveys related work. Section 7 catalogs open problems and extensions. Section 8 concludes.

## 2. Threat Model and Design Goals

---

### 2.1 Security Objectives

The protocol targets five security objectives:

1. **Authority attenuation.** Delegated capabilities must be a subset of the delegator's capabilities. No agent can grant permissions it does not hold.
2. **Authenticated identity.** Every protocol action is bound to an Ed25519 key pair. Identity claims are cryptographically verifiable without trusted third parties.
3. **Traceable attribution.** Every agent action traces to a human beneficiary through an auditable delegation chain. Merkle proofs commit to receipt sets in logarithmic space.
4. **Revocable delegation.** Any delegation can be revoked by its issuer, with cryptographically verifiable cascade to all transitive sub-delegations. Revocation is irreversible.
5. **Governance integrity.** Governance artifacts are signed, versioned, and subject to monotonic strengthening. Weakening requires higher-order authorization.

### 2.2 Trusted Computing Base

The protocol assumes: Ed25519 signature correctness (via Node.js native `crypto` module), SHA-256 for Merkle trees, deterministic canonical JSON serialization, correct scope subset checking, secure private key storage mediated by a trusted signing service, a trusted time source, and the ProxyGateway as enforcement point in the strong deployment model.

### 2.3 Attacker Classes

We define three attacker classes with increasing capability:

**Class 1: Adversarial agent.** The attacker controls the agent's prompts, reasoning, and protocol inputs. The agent attempts to exceed its delegated scope, forge identities, bypass policy evaluation, or game attribution. The attacker does not have access to other agents' private keys or to the gateway internals. This is the primary threat the protocol addresses.

**Class 2: Messaging attacker.** The attacker can inject signed messages, submit false evidence, or manipulate governance feeds. The attacker has valid credentials but uses them dishonestly. The attacker cannot forge other

agents' signatures.

**Class 3: Runtime attacker.** The attacker has access to the agent's memory, can extract private keys, bypass the SDK entirely, or compromise the gateway. This class is out of scope for the protocol's cryptographic guarantees. We discuss it as a known limitation (Section 7.1).

## 2.4 Strong vs. Conditional Guarantees

**Strong guarantees** hold when the ProxyGateway mediates all privileged actions. In this configuration, the enforcement boundary prevents agents from bypassing scope checks, signature requirements, policy evaluation, governance staleness checks, and reversibility constraints. The gateway is external to the agent's reasoning, analogous to a reference monitor.

**Conditional guarantees** hold when agents voluntarily use the SDK without the gateway. The protocol is not intended to make a fully compromised runtime trustworthy. Its strongest guarantees hold when all privileged effects are mediated by protocol-aware enforcement points external to agent reasoning logic.

## 3. Four Attenuation Invariants

---

We generalize capability attenuation from a single property of delegation chains into a family of four invariants that span the governance lifecycle. Each invariant constrains a different state object, but all share the same monotonic structure: the protected quantity can only decrease (or, equivalently, the constraint can only strengthen) at each transfer point.

### 3.1 Abstract State Model

We define the protocol state as:

**S = (I, D, R, G, P, E, M, X)**

where I is the set of agent identities (Ed25519 key pairs), D is the set of active delegations, R is the revocation set, G is the set of governance artifacts (signed, versioned policy documents), P is the set of policy records (three-signature chains), E is the evidence and receipt log, M is the message feed, and X is the set of active escalations.

#### 3.1.1 State Transitions

Seven transitions operate on state S. Each transition has preconditions that must hold for the transition to fire, and postconditions that describe the resulting state.

**CREATE\_IDENTITY(name, owner, beneficiary) → S'.** Precondition: name is unique in I. Owner holds a valid Ed25519 key pair. Postcondition: generates key pair (pk, sk), creates signed passport, adds identity to I. The passport binds pk to the agent's metadata.

**DELEGATE(from, to, scope, spendLimit, maxDepth) → S'.** Precondition: from has an active delegation d<sub>parent</sub> with scope ⊇ requested scope, spendLimit ≥ requested spendLimit, and depth > 0. Postcondition: creates delegation d<sub>new</sub> where d<sub>new</sub>.scope ⊆ d<sub>parent</sub>.scope, d<sub>new</sub>.spendLimit ≤ d<sub>parent</sub>.spendLimit, d<sub>new</sub>.depth < d<sub>parent</sub>.depth. Adds to D, updates L. This is the INV-1 enforcement point at the SDK layer.

**REVOKE(delegationId, issuerKey) → S'.** Precondition: delegationId exists in D. issuerKey matches the delegation's issuer. Postcondition: adds delegationId and all transitive descendants to R. This is irreversible: once an id enters R, no subsequent transition removes it. Updates D (marks inactive), updates L.

**UPDATE\_GOVERNANCE(artifact, approvals) → S'.** Precondition: artifact is signed by a recognized issuer. If artifact weakens existing governance in G, |approvals| ≥ threshold<sub>weakening</sub>. Postcondition: adds artifact to G. All agents' governance attestation versions become stale if the version changed. This is the INV-2 enforcement point.

**DECLARE\_AND\_EVALUATE(agentId, action, scope) → S'.** Precondition: agentId has active identity in I. Agent holds a valid delegation covering scope. Postcondition: creates signed ActionIntent (signature 1), evaluates against floor and delegation via FloorValidatorV1 producing PolicyDecision (signature 2). Adds partial record to P.

**EXECUTE\_AND\_RECORD(intentId, toolResult) → S'.** Precondition: intentId has PolicyDecision with verdict "allow" in P. Delegation not revoked (recheck). Governance not stale (recheck). Postcondition: gateway executes tool, generates signed receipt (signature 3). Completes the record in P. Appends to E. Optionally appends to M.

**ACTIVATE\_ESCALATION(grantId, trigger, humanApproval) → S'.** Precondition: grant exists with grantId. trigger matches grant's allowedTriggers. If trigger is human\_authorized, humanApproval signature verifies against granter's public key. Grant's ceiling.scope  $\subseteq$  granter's delegation scope. Postcondition: creates active escalation x in X with TTL. x.effectiveScope  $\subseteq$  grant.ceiling.scope. This is the INV-4 enforcement point.

### 3.2 INV-1: Delegation Attenuation

**Informal statement.** Authority can only decrease at each delegation transfer. No agent can possess more authority than explicitly granted by the chain of principals above it.

**State object.** Delegation d in D, characterized by (scope, spendLimit, maxDepth, delegatedBy, delegatedTo).

**Monotonic relation.** For any delegation chain  $d_1, d_2, \dots, d_k$ :

```
For all i in [1, k-1]:
  scope(d_{i+1})  $\subseteq$  scope(d_i)
  spendLimit(d_{i+1})  $\leq$  spendLimit(d_i)
  depth(d_{i+1}) < depth(d_i)
```

Additionally, revocation is monotonic: once delegation identifier id enters R, it remains in R for all subsequent states. Cascade revocation propagates: if  $d_i$  is in R, then for all descendants  $d_j$  ( $j > i$ ),  $d_j$  is also in R.

**Enforcement mechanism.** The ProxyGateway enforces INV-1 at three points: - Step 3 (delegation check): `scopeAuthorizes(delegation.scope, request.scopeRequired)` must return true. The `scopeAuthorizes` function implements hierarchical matching where `data` covers `data:read` and `*` covers everything. - Step 6 (revocation recheck): at execution time, the gateway re-verifies that the delegation has not been revoked between approval and execution (TOCTOU mitigation). - Step 8 (receipt): the gateway generates the receipt, not the agent. The agent never touches the signature. Receipt scope is bound to the delegation scope that authorized the action.

The SDK enforces INV-1 in `subDelegate()` and `createReceipt()` using `scopeCovers()`, ensuring that the authoring layer matches the enforcement layer (verified by 10 authoring-enforcement parity tests).

**Pseudocode: scopeAuthorizes (hierarchical scope matching).**

```
function scopeAuthorizes(delegationScope, requestedScope):
  for each scope s in delegationScope:
    if s == "*": return true           // wildcard covers everything
    if s == requestedScope: return true // exact match
    if requestedScope starts with s + " ": // hierarchical: "data" covers "data:read"
      return true
  return false
```

This function is called at both the SDK layer (`subDelegate`, `createReceipt`) and the gateway layer (Step 3), ensuring the authoring and enforcement layers cannot diverge. The V3-CRIT-1 bug fix (March 2026) replaced literal `Array.includes()` with `scopeCovers()` at every callsite after adversarial review discovered the divergence.

**Evidence.** 35 delegation tests, 23 adversarial scenarios (including scope escalation, replay, impersonation), 9 hierarchical scope narrowing tests, 10 authoring-enforcement parity tests. Gateway replay protection via TTL-based Map pruning (NW-001 fix). Key rotation with identity continuity (Module 22) ensures INV-1 survives key compromise.

**Limitations.** Scope semantics are string-based with hierarchical matching, not ontological. The protocol does not prevent a principal from issuing an overly broad delegation; it only prevents sub-delegations from widening scope.

### 3.3 INV-2: Governance Attenuation

**Informal statement.** Governance artifacts (floors, policies, delegation templates) can only strengthen over time. Weakening an existing governance artifact requires higher-order authorization than strengthening it.

**State object.** Governance artifact  $g$  in  $G$ , characterized by (artifactType, version, contentHash, issuer, changeType, additions, removals, signature). Each artifact forms a version chain linked by previousArtifactId.

**Monotonic relation.** For any governance artifact version chain  $g_1, g_2, \dots, g_n$ :

```
For all  $i$  in  $[1, n-1]$ :
  principles( $g_{i+1}$ )  $\supseteq$  principles( $g_i$ )           [strengthening: always permitted]

If principles( $g_{i+1}$ )  $\subset$  principles( $g_i$ ):           [weakening]
  |approvals( $g_{i+1}$ )|  $\geq$  threshold_weakening

If removals( $g_{i+1}$ )  $\neq \emptyset$ :                     [removal]
  |approvals( $g_{i+1}$ )|  $\geq$  threshold_removal > threshold_weakening
```

Strengthening (adding principles) requires no additional approval. Weakening (modifying principles to be less restrictive) requires at least one higher-order approval. Removal of principles requires at least two approvals (configurable via GovernanceLoadPolicy).

**Enforcement mechanism.** The ProxyGateway enforces INV-2 through: - `updateGovernance()` : validates the new artifact via `loadGovernanceArtifact()` with differential approval thresholds. Strengthening updates are accepted with zero additional approvals. Weakening updates without sufficient approvals are rejected. The gateway tracks `governanceWeakeningBlocked` in stats. - Step 2 (governance staleness check): before any action, the gateway verifies that the agent's attested governance version matches the current version. If governance has been updated since the agent's last attestation, the action is blocked until the agent re-attests via `reattestGovernance()`. - Execution-time recheck: the same staleness check runs in the two-phase `executeApproval` path, catching governance updates that occur between approval and execution. - Policy conflict detection (Module 30): `detectPolicyConflicts()` identifies circular dependencies, unreachable actions, and shadowed rules in policy sets before they enter enforcement.

**Pseudocode: updateGovernance (differential authorization).**

```
function updateGovernance(newArtifact, approvals):
  verify(newArtifact.signature, newArtifact.issuerPublicKey)
  if issuer not in loadPolicy.allowedIssuers: REJECT("unknown issuer")

  diff = classifyChange(currentArtifact, newArtifact)
  if diff.hasRemovals:
    if |approvals| < threshold_removal: REJECT("removal requires N approvals")
  else if diff.hasWeakenings:
    if |approvals| < threshold_weakening: REJECT("weakening requires approval")
  // else: strengthening – no additional approval needed

  currentArtifact = newArtifact
  currentVersion = newArtifact.version
  stats.governanceUpdates++
  for each agent in registeredAgents:
    if agent.governanceVersion != currentVersion:
      agent.governanceStale = true    // blocks until re-attestation
```

**Evidence.** 14 gateway-governance tests including: strengthening accepted, weakening blocked, weakening accepted with approvals, removal requires higher threshold, unknown issuer rejected, stale governance blocks both `processToolCall` and `executeApproval` paths, version tracking across updates. Module 30 adds 13 policy conflict detection tests.

**Limitations.** "Higher-order authorization" is currently multi-party approval count, not cryptographic proof of organizational hierarchy. There is no formal policy composition algebra; two valid policies can theoretically deadlock (though Module 30 detects this statically). The governance version check is string-equality based, not semantic versioning comparison.

### 3.4 INV-3: Disclosure Attenuation

**Informal statement.** Audit views reveal only the minimum necessary subset of committed data. A verifier can confirm a specific receipt's inclusion in the record without seeing the full set.

**State object.** Evidence log E, committed via Merkle root. Individual receipts are leaves. Audit proofs are paths from leaf to root.

**Monotonic relation.** For any disclosure D produced from committed set C:

```
D_viewer ⊆ D_committed

For any Merkle proof π over leaf h:
  Verify(π, root) = true  ⇒  h ∈ leaves(C)
  |π| = O(log |C|)
```

The viewer sees a proof for a specific receipt without learning anything about other receipts in the committed set. The proof size is logarithmic in the total number of receipts.

**Enforcement mechanism.** - `buildMerkleRoot(leafHashes)` computes the root from a batch of receipt hashes. The root is published; individual receipts are not. - `generateMerkleProof(leafHashes, targetHash)` produces an inclusion proof for a single receipt. - `verifyMerkleProof(proof)` recomputes the root from the proof path and compares against the claimed root. - E2E encrypted messaging (Module 19) uses X25519 key exchange with XSalsa20-Poly1305 encryption. Messages on the Agora are encrypted end-to-end; only sender and recipient can read content. - Messaging audit bridge (Module 29) creates audit records for encrypted channels that include message hash, sender, recipient, and taint labels, but never message content. This allows the gateway to know a message was sent without breaking encryption.

**Evidence.** Attribution tests verify Merkle root construction, individual proof generation, and proof verification. Integration invariant test (INV-3) demonstrates selective disclosure end-to-end. Module 19 has 13 encryption tests. Module 29 has 12 messaging audit tests verifying that content is never included in audit records.

**Limitations.** Current selective disclosure is per-receipt (reveal or hide an entire receipt), not per-field within a receipt. BBS+ signatures would enable attribute-level selective disclosure but are not implemented. The fundamental tension between full auditability and data minimization is acknowledged but not resolved. Encrypted messaging bypasses the gateway entirely; the audit bridge provides metadata-level visibility but not content-level enforcement.

We note that INV-3 is the least mature of the four invariants in terms of implementation depth. The principle (disclosure should monotonically narrow) is sound and applies regardless of mechanism. Merkle trees are the v1 implementation; BBS+ or ZK-SNARKs would be a v2 mechanism enforcing the same principle with stronger privacy properties. We include INV-3 because the governance framework requires a disclosure invariant to be complete — without it, the other three invariants produce audit records that leak arbitrary information, undermining the protocol's utility in regulated environments where data minimization is legally required.

### 3.5 INV-4: Exception Attenuation

**Informal statement.** When an agent needs temporary authority beyond its normal delegation, that authority is bounded by a pre-committed ceiling, enforced by a hard time-to-live, and auditable through a distinct flag. Escalation is a controlled expansion, not an open bypass.

**State object.** Escalation grant eg and active escalation x in X, characterized by (ceiling.scope, ceiling.maxSpend, ceiling.maxDurationMs, allowedTriggers, allowedActionClasses).

**Monotonic relation.** For any active escalation  $x$  derived from grant  $eg$ :

```
x.effectiveScope ≤ eg.ceiling.scope
x.effectiveSpendLimit ≤ eg.ceiling.maxSpend
duration(x) ≤ eg.ceiling.maxDurationMs

eg.ceiling.scope ≤ scope(granterDelegation)    [ceiling within granter's authority]
```

The escalation grant is itself subject to INV-1: its ceiling scope must be a subset of the granter's delegation scope. Escalation does not create new authority; it temporarily re-routes pre-committed authority through a bounded channel.

**Enforcement mechanism.** The ProxyGateway enforces INV-4 through:

- Step 3.1 (escalation fallback): when the normal delegation check fails, the gateway checks `agent.activeEscalations` for a valid grant covering the requested action via `checkEscalatedAction()`. If found, the action proceeds with `viaEscalation: true` on the result.
- TTL enforcement: `isEscalationActive()` checks both status and temporal validity. Expired escalations are automatically cleaned via `_expireAgentEscalations()`.
- Scope check: `checkEscalatedAction()` verifies the requested action is within the escalation's effective scope using `scopeAuthorizes()`.
- Spend tracking: each escalated action increments `spentDuringEscalation`. Actions exceeding the remaining budget are denied.
- Human approval verification: for `human_authorized` triggers, the gateway cryptographically verifies the human's approval signature against the granter's public key.
- Max concurrent escalations: configurable limit (default 1) prevents accumulation of parallel authority expansions.
- Reversibility taxonomy: actions are classified as tentative, compensable, or irreversible. The gateway can enforce a `maxReversibility` ceiling that limits which action classes are permitted, even via escalation.
- Oracle witness diversity (Module 28): when external verification is needed, `computeWitnessDiversity()` scores the independence of witness sources. Three APIs from the same cloud provider count as one effective witness.

**Pseudocode: checkEscalatedAction (bounded exception fallback).**

```
function checkEscalatedAction(agent, requestedScope, spend):
  for each escalation x in agent.activeEscalations:
    if not isEscalationActive(x): continue      // TTL expired
    if not scopeAuthorizes(x.effectiveScope, requestedScope): continue
    if spend and x.spentSoFar + spend > x.ceiling.maxSpend: continue
    // Found valid escalation
    x.spentSoFar += spend
    return { authorized: true, escalationId: x.id, viaEscalation: true }
  return { authorized: false }
```

This function runs only when the normal delegation check (Step 3) fails. The gateway never checks escalation first — delegation is always the primary authorization path. Escalation is a fallback, not a shortcut.

**Evidence.** 11 gateway-escalation tests including: normal delegation bypass, escalation fallback, expired TTL denial, scope mismatch, max concurrent enforcement, revocation, spend tracking across calls. Module 27 has its own unit tests for grant creation, activation, and verification. Module 28 has 19 oracle witness diversity tests. 3 reversibility tests in the integration invariant suite.

**Limitations.** v1 escalation supports only `human_authorized` triggers. Multi-witness triggers are typed but not yet enforced at the gateway. Challenge economics (allowing other agents to challenge an escalation with stake) and compensation mechanisms are designed but not implemented. Precedent accumulation (Module 25) exists but does not yet feed into escalation decisions. The reversibility taxonomy is metadata-only; the gateway checks it but does not independently verify an action's actual reversibility.

### 3.6 Invariant Composition

The four invariants are not merely parallel constraints. They compose through five formally tested relationships, two identified interactions requiring future work, and one self-referential property.

### 3.6.1 Tested Composition Relationships

**C1: INV-2 takes precedence over INV-4 (governance blocks escalation).** When governance is updated, all agents' attestations become stale. The governance staleness check (Step 2) runs before both the delegation check (Step 3) and the escalation fallback (Step 3.1) in the processing pipeline, and independently in the `executeApproval` path. An agent with an active escalation grant that would normally cover an action is still blocked if its governance attestation is stale. This means governance weakening cannot be circumvented by acquiring an escalation grant first.

This composition is enforced by the ProxyGateway's processing order. A different enforcement engine that checks escalation before governance would violate C1. We specify this as a requirement: any conforming enforcement engine MUST evaluate governance staleness before delegation or escalation checks. The invariant holds because of the specified processing order, not because of an inherent algebraic property. We test this composition directly in `integration-invariants.test.ts`.

**C2: INV-1 bounds INV-4 (delegation caps escalation ceiling).** An escalation grant's ceiling scope must be a subset of the granter's delegation scope. This is verified at grant creation by `verifyEscalationGrant()`. Escalation cannot create authority that the granting principal does not already possess. Exception attenuation is itself subject to delegation attenuation.

**C3: INV-1 feeds INV-3 (delegation chains anchor disclosure).** Merkle proofs reference delegation chains. A receipt's attribution trace runs from the executing agent through the delegation chain to the human principal. Disclosure of a receipt necessarily discloses the delegation path that authorized it, but nothing about other delegation paths.

**C4: INV-1 anchors INV-2 (delegation scope bounds governance authority).** Only agents with delegation scope covering governance operations can submit governance artifact updates. The governance update function verifies the issuer against `allowedIssuers` in the load policy, and these issuers must themselves hold valid delegations. Governance cannot be modified by agents outside the delegation tree.

**C5: INV-4 is bounded by INV-2 AND INV-1 simultaneously.** An escalation grant is constrained from above (ceiling scope  $\subseteq$  granter's delegation, via C2) and constrained from below (governance staleness blocks execution, via C1). The grant exists in the intersection of what the delegation tree permits and what the governance configuration allows.

The composition claims in C1 and C5 are realized concretely by the gateway pipeline ordering defined in Section 4.4. Enforcement requires that the ProxyGateway implement a hard-coded sequence where governance staleness (Step 2) is checked before delegation (Step 3) and escalation fallback (Step 3.1), and this sequence must be transactionally atomic to prevent Time-of-Check to Time-of-Use (TOCTOU) exploits between governance verification and escalation evaluation.

### 3.6.2 Identified but Untested Interactions

Two composition relationships were identified during hostile review but are not yet tested:

**INV-3  $\times$  INV-1 (disclosure after revocation).** When a delegation is revoked (INV-1), receipts previously disclosed under that delegation chain remain valid as historical records but their attribution trace now points to a revoked chain. The current implementation does not invalidate prior disclosures on revocation. Whether this is correct behavior (the receipt DID happen under valid authority at the time) or a gap (the verifier should see the current revocation status) is a genuinely open design question — the answer depends on whether the protocol treats receipts as historical facts or live claims, and different deployment contexts may require different answers.

**INV-3  $\times$  INV-2 (disclosure under governance evolution).** Receipts generated under governance version N may be verified under governance version N+1. If the governance was strengthened, actions that were permitted under the old governance might not be permitted under the new. The receipt remains valid (the action was authorized at the time), but a verifier applying current governance to a historical receipt will see a mismatch. This interaction is untested because the governance versioning semantics for receipt verification have not been specified



— the protocol currently treats receipts as immutable records of past actions without cross-referencing the governance version under which they were generated.

### 3.6.3 The Self-Referential Property

The composition relationships reveal a deeper structural property: the framework applies the same formal mechanism to its own governance rules that it applies to agent actions.

Policy weakening is an exception to INV-2's monotonic strengthening requirement. It is handled by requiring higher-order authorization (multiple approvals with configurable thresholds). This is structurally identical to INV-4's bounded escalation mechanism: both define a pre-committed ceiling, both require higher-order authorization to activate, both produce audit records of the exception, and both are irreversible once activated.

The parallel structure:

```
INV-4 (runtime escalation):
  agent requests authority beyond delegation
  → requires pre-committed grant + human approval
  → bounded by ceiling scope and TTL
  → audit flag: viaEscalation = true

INV-2 (governance weakening):
  operator requests policy change that weakens governance
  → requires multi-party approval above threshold
  → bounded by removal threshold > weakening threshold
  → audit flag: changeType = "weakening"
```

Both use the same pattern: exception to a monotonic rule, gated by higher-order authorization, bounded by a pre-committed ceiling, producing a distinct audit record. The framework's governance update mechanism is itself an instance of bounded exception attenuation.

### 3.6.4 Terminology and Limitations

We use "compositional constraints" rather than "lattice" because the composition relationships are directional precedence constraints (C1: governance before escalation) and bounding constraints (C2: delegation caps escalation), not algebraic join/meet operations. A formal lattice would require defining a partial order with supremum and infimum for every pair of invariants, which we have not done.

The composition is enforced by the ProxyGateway's processing pipeline order, not by an inherent mathematical property of the invariants themselves. A conforming enforcement engine must implement the specified processing order to preserve C1 and C5. This is an architectural requirement, not an algebraic guarantee.

**Fixed-point vulnerability.** The self-referential property creates a potential fixed-point problem: if the governance update mechanism itself has a bug, the broken mechanism gates its own repair. The specified escape hatch is root principal access: the human principal with direct gateway access can override the governance check as an out-of-band recovery mechanism. This is analogous to a constitutional amendment process that, if deadlocked, falls back to the sovereign authority that created the constitution.

This self-referential composition property is a central claim of the paper: the four invariants form a compositional constraint graph with tested precedence relationships and a shared authorization pattern, not a disconnected set of rules.

## 4. Architecture

### 4.1 Module Organization

The protocol is implemented as 35 TypeScript modules organized by governance domain. Each module maps to one or more invariants.

**INV-1 (Delegation Attenuation):** passport.ts (identity creation, Ed25519 key pairs), delegation.ts (scope narrowing, cascade revocation, sub-delegation), canonical.ts (deterministic JSON serialization for cross-language signature compatibility), identity.ts (key rotation with DID continuity), reanchor.ts (delegation re-anchoring after key rotation), cross-chain.ts (taint tracking for cross-principal data flow), a2a.ts (agent-to-agent protocol bridge), did.ts (W3C DID document generation), vc.ts (Verifiable Credentials bridge).

**INV-2 (Governance Attenuation):** values.ts (8 principles F-001 through F-008, attestation, compliance), governance.ts (signed versioned artifacts, content-hash integrity, weakening detection), policy.ts (3-signature chain, FloorValidatorV1, graduated enforcement), euaiact.ts (EU AI Act compliance mapping), policy-conflict.ts (circular dependency detection, shadowed rules, unreachable actions).

**INV-3 (Disclosure Attenuation):** attribution.ts (Merkle tree construction, proof generation, beneficiary tracing), receipt-ledger.ts (Merkle-chained receipt batches), encrypted-messaging.ts (X25519 + XSalsa20-Poly1305 E2E encryption), messaging-audit.ts (audit records without content disclosure).

**INV-4 (Exception Attenuation):** escalation.ts (bounded escalation grants, human approval verification, TTL enforcement), oracle-witness.ts (witness diversity scoring, consensus evaluation), reputation-authority.ts (Bayesian trust tiers, effectiveAuthority = min(delegation, tier)), precedent.ts (precedent accumulation, normative drift prevention), feasibility.ts (delegation feasibility linting).

**Cross-cutting:** gateway.ts (ProxyGateway enforcement boundary, 1014 lines), intent.ts (roles, deliberation, consensus), coordination.ts (task lifecycle), commerce.ts (4-gate checkout), integration.ts (cross-layer bridges), context.ts (agent execution context), routing.ts (task routing), obligations.ts (duties with deadlines and penalties), execution-envelope.ts (cross-engine interop format), principal.ts (principal identity management), contract.ts (high-level API).

## 4.2 The ProxyGateway: Enforcement Boundary

The ProxyGateway is the protocol's reference monitor. It is the only component that must be trusted for strong guarantees to hold. The gateway satisfies six minimum viable requirements identified through adversarial review:

1. **The gateway is the executor.** The gateway calls the downstream tool, not the agent. The agent submits a signed request; the gateway validates, executes, and generates the receipt. The agent never touches the receipt signature.
2. **Exact parameter binding.** At approval time, the gateway binds the exact tool name, parameters, target, and spend amount. The agent cannot modify these between approval and execution.
3. **Revocation recheck at execution time.** In the two-phase approve/execute path, the gateway re-verifies delegation validity at execution time, not just at approval time. This mitigates TOCTOU attacks where a delegation is revoked between approval and execution.
4. **Gateway generates the receipt.** Receipt generation is in the enforcement layer, not the agent. This prevents the "lying receipt" attack where a compromised agent forges favorable execution records.
5. **Replay protection.** Each request ID is tracked with a TTL-based Map. Resubmitting a consumed request ID results in immediate denial.
6. **Approval timeout.** Approvals expire after a configurable TTL (default 30 seconds). Stale approvals cannot be stockpiled.

## 4.3 Three-Signature Policy Chain

Every executed action produces a three-signature proof chain:

1. **Intent signature** (agent → gateway): the agent declares what it intends to do, signed with its Ed25519 key. Created by `createActionIntent()`.
2. **Decision signature** (gateway → policy engine): the gateway evaluates the intent against the values floor, delegation scope, and governance artifacts. The `FloorValidatorV1` produces a `PolicyDecision` with verdict (allow/deny/escalate), signed by the gateway. Created by `evaluateIntent()`.
3. **Receipt signature** (gateway → proof): the gateway executes the tool, records the result, and signs the receipt. Created by `createPolicyReceipt()`.

No execution receipt can exist without a corresponding intent and decision. This is INV-6 from the original formalization (three-signature completeness). The chain provides non-repudiation at the protocol level: the agent cannot deny it requested the action, and the gateway cannot deny it approved and executed it.

#### 4.4 Processing Pipeline

The gateway processes each tool call through a sequential pipeline of enforcement checks:

Step 0: Replay protection (request ID uniqueness). Step 1: Signature verification (agent's Ed25519 signature over canonical request). Step 2: Governance staleness check (INV-2, blocks both delegation and escalation paths). Step 3: Delegation check (scope authorization via `scopeAuthorizes()`, includes reversibility constraint on action class). Step 3.1: Escalation fallback (INV-4, if delegation fails; also checks reversibility). Step 4: Intent creation. Step 5: Policy evaluation (with INV-4 scope override for escalated actions). Step 5.5: Reputation tier check (effectiveAuthority = min(delegation, tier)). Step 6: Revocation recheck at execution time. Step 7: Tool execution. Step 7.5: Taint tracking and SAO wrapping (cross-chain). Step 8: Receipt generation. Step 9: Policy receipt creation. Step 9.5: Obligation fulfillment check. Step 9.7: Reputation update.

All six gateway features (governance, escalation, cross-chain, obligations, reputation, reversibility) operate simultaneously without conflict, verified by a dedicated stress test suite.

### 5. Evaluation

#### 5.1 Test Suite

The implementation is validated by 866 tests across 239 suites in 49 test files. Tests are organized by invariant:

**INV-1 tests (delegation, identity, adversarial):** 35 delegation chain tests, 23 adversarial scenarios (scope escalation, replay, impersonation, forged signatures, cascade bypass), 9 hierarchical scope narrowing tests, 10 authoring-enforcement parity tests, cascade revocation tests, canonical serialization edge cases. The adversarial suite includes 2 expected failures that document known limitations rather than bugs.

**INV-2 tests (governance, policy, values):** 14 gateway-governance tests (strengthening accepted, weakening blocked, differential thresholds, staleness check, executeApproval recheck), 13 policy conflict detection tests (cycle DFS, shadowed rules, unreachable actions), values floor attestation and compliance tests, policy chain tests.

**INV-3 tests (attribution, encryption, audit):** Merkle root construction and proof verification, 13 E2E encrypted messaging tests, 12 messaging audit bridge tests (verifying content is never disclosed).

**INV-4 tests (escalation, oracle, reputation):** 11 gateway-escalation tests (fallback, TTL expiry, scope check, spend tracking, max concurrent, revocation), 19 oracle witness diversity tests, 9 reputation-gated authority tests, 3 reversibility enforcement tests.

**Integration tests:** 8 integration invariant tests mapping directly to Section 3 claims (INV-1 scope narrowing, INV-2 governance weakening, INV-3 Merkle disclosure, INV-4 escalation TTL, COMPOSITION governance > escalation, 3 reversibility tests). 4 stress tests verifying all 6 gateway features simultaneously. 14 cross-layer integration wiring tests.

##### 5.1.1 Representative Adversarial Scenarios

We describe four scenarios from the adversarial suite that exercise the invariant boundaries. Each scenario tests a specific attack vector against the gateway enforcement boundary.

**Scenario A: Hierarchical scope escalation (INV-1).** Agent C holds delegation with scope `["data:read"]` sub-delegated from B, who holds `["data"]` from principal A. C submits a tool call with `scopeRequired: "data:write"`. The gateway's `scopeAuthorizes()` check at Step 3 rejects: `"data:read"` does not cover `"data:write"`. C then attempts to forge a new delegation with broadened scope and submits again. The gateway rejects at Step 1: the forged delegation's signature does not verify against B's public key. **Expected result: both attempts denied. Observed: denied at Step 1 or Step 3. Even if the agent attempts the two-phase**

**approval/executeApproval bypass path, the attack is blocked by parity checks in**

**\_executeApprovalInner** which independently verifies scope authorization. This scenario was the source of the V3-CRIT-1 bug: prior to the fix, the SDK's `subDelegate()` used literal string matching (`Array.includes()`) which would have accepted `"data:write"` if the parent scope included the exact string. The gateway's `scopeAuthorizes()` correctly rejected it. The fix aligned both layers.

**Scenario B: Governance downgrade bypass via escalation (COMPOSITION).** Agent has an active escalation grant covering `["admin:*"]`. Governance is updated (floor principle removed). The agent's governance attestation becomes stale. Agent attempts an admin action using the escalation grant. The gateway checks governance staleness (Step 2) BEFORE checking delegation (Step 3) or escalation (Step 3.1). **Expected result: denied due to stale governance, even though escalation would cover the scope. Observed: denied.** This is the composition test: INV-2 takes precedence over INV-4 in the pipeline.

**Scenario C: Replay attack on two-phase execution.** Agent receives approval for a tool call (Phase 1). Agent submits the same approval ID twice for execution (Phase 2). The gateway marks the approval as `consumed: true` after the first execution. The second submission is rejected at Step 0: approval already consumed. **Expected result: second execution denied. Observed: denied.** The TTL-based Map pruning (NW-001 fix) also prevents accumulation of stale request IDs in long-running gateways.

**Scenario D: Escalation TTL expiry mid-action (INV-4).** Agent has an escalation grant with 60-second TTL. At T=0, escalation activates. At T=55, agent submits a tool call. Gateway checks `isEscalationActive()`: TTL not expired, action proceeds. At T=65, agent submits another tool call. Gateway checks: TTL expired, `_expireAgentEscalations()` cleans the grant. Action denied, falls through to normal delegation check, which also fails. **Expected result: first action succeeds via escalation, second denied. Observed: correct.**

Two additional scenarios in the adversarial suite document **expected failures** — cases where the protocol deliberately does not mitigate the attack: (1) a Class 3 attacker with access to the agent's private key can forge valid signatures (out of scope per Section 2.3), and (2) two agents sharing a principal's context window can coordinate without using protocol channels (Lampson's covert channel, provably unsolvable per Section 7.2). These expected failures are documented as known limitations, not bugs.

## 5.2 AIVSS Risk Mapping

We map the protocol against the OWASP AI Vulnerability Scoring System (AIVSS) risk categories with honest coverage assessment:

**Strong coverage (5 risks):** Identity verification and authentication (Ed25519 passports, challenge-response), authorization and access control (scoped delegation, cascade revocation, gateway enforcement), audit and traceability (3-signature chain, Merkle receipts, messaging audit), input validation (signature verification on all protocol inputs), secure communication (E2E encryption, signed Agora messages).

**Partial coverage (3 risks):** Data protection (Merkle commitments exist but no ZKP/BBS+, audit-vs-privacy tradeoff unresolved), governance and compliance (Values Floor with 8 principles but graduated enforcement is advisory for F-006/F-007/F-008), supply chain security (governance artifact provenance signed and versioned but no SBOM integration).

**Weak coverage (2 risks):** Runtime integrity (protocol assumes TCB is correct; no runtime attestation), model security (protocol operates above the model layer; prompt injection constrained but not prevented).

## 5.3 Cross-Architecture Robustness

Independent developers have implemented complementary layers that interoperate with the protocol, providing evidence of architectural generality:

The Nexus-Guard project built a cross-protocol bridge between the Agent Identity Protocol (AIP) and APS, with all four resolution directions working (AIP→APS, APS→AIP identity and delegation). The xsa520/Guardian project proposed a pre-execution evidence ledger that maps to the protocol's 3-signature chain, with the evidence attestation serving as a potential 4th signature. The sunilp/GovernancePlugin adapter for Google's Agent

Development Kit (ADK) implements APS delegation checking within ADK's PolicyEvaluator interface. These integrations were developed independently of the core APS implementation, providing preliminary evidence of architectural portability, though they do not yet constitute a formal conformance suite.

## 5.4 Reproducibility

All empirical results in this paper refer to SDK v1.16.1 (commit c1842eb, March 2026), comprising 35 modules and 866 tests. The protocol is under active development; the live test suite may differ from the frozen artifact evaluated here.

To reproduce the test results:

```
npm install agent-passport-system@1.16.1
npm test
# Expected: 866 tests, 239 suites, 0 failures
# Adversarial scenarios: tests/adversarial*.test.ts
# Gateway enforcement: tests/gateway*.test.ts
# Integration invariants: tests/integration-invariants.test.ts
# Cross-module stress: tests/stress-all-features.test.ts
```

The TypeScript SDK, Python SDK, and MCP server are available at [github.com/aeoess](https://github.com/aeoess) under Apache-2.0 license. The MCP server (61 tools) can be installed via `npm install agent-passport-system-mcp` and provides a universal interface for any MCP-compatible client.

## 6. Related Work

### 6.1 Capability Systems and Authority Attenuation

The principle of capability attenuation originates with Dennis and Van Horn (1966) and was formalized through the E programming language (Miller, 2006) and the object-capability model. KeyKOS, EROS, seL4, and Capsicum implement capability security at the OS level. Our contribution applies this principle to LLM-based agents operating across organizational boundaries with non-deterministic behavior. Our delegation chains are capability tokens with Ed25519 binding, narrowing scope sets, and decreasing spend limits.

### 6.2 Delegation Logics and Trust Management

SPKI/SDSI (Ellison et al., 1999) provides certificate chains for authorization. PolicyMaker and KeyNote (Blaze et al., 1999) define trust management as compliance checking. XACML provides attribute-based policy languages. Usage Control (Park and Sandhu, 2004) extends access control with obligations and mutable attributes. Our obligations model (Module 20) draws on UCON's distinction between permissions and duties.

### 6.3 Agent Delegation Standards

The IETF Internet-Draft on Delegated Agent Authorization Protocol (DAAP, draft-mishra-oauth-agent-grants-01) independently arrived at identical delegation invariants: scope narrowing, cascade revocation, and spend limits. This convergence from an OAuth extension perspective provides evidence that these constraints are necessary rather than arbitrary. The OpenID Foundation's report on "Identity Management for Agentic AI" identifies many of the same challenges but proposes federation-based solutions. Our protocol operates at a lower layer, providing cryptographic primitives that federation systems can build on.

### 6.4 Workflow Satisfiability and Liveness

Cohen et al. establish fixed-parameter tractability results for the Workflow Satisfiability Problem (WSP). Our feasibility linting module (Module 24) addresses a simpler version of WSP: checking at delegation-creation time whether the delegation's scope and constraints are satisfiable given the current governance configuration. Full WSP is a known limitation (Section 7).

## 6.5 Trust Infrastructure

OpenID Federation 1.0 provides a trust chain model where entities inherit trust from federation operators. NIST RATS (Remote Attestation Procedures) defines an attestation vocabulary for runtime integrity claims. W3C Decentralized Identifiers (DIDs) provide a standard format for self-sovereign identity. Our protocol generates W3C-compatible DID documents (Module did.ts) and maps to Verifiable Credentials (Module vc.ts), enabling interoperability with the broader decentralized identity ecosystem.

## 6.6 AI Agent Governance

DeepMind's "Intelligent Delegation" paper (2026) proposes a capability-based approach to agent oversight. The LOKA Protocol targets open autonomous agent networks. OpenAI's governance practices describe internal oversight mechanisms. The BSA (Business Software Alliance) submitted recommendations to NIST that align with our architecture. To our knowledge, none of these provide a complete protocol stack with cryptographic enforcement, formal invariants, and tested implementation. The closest competitor in the open-source space is the Open Agent Identity Protocol (AIP), with which we have a working cross-protocol bridge.

## 7. Open Problems and Extensions

---

We identify 10 remaining open problems, graded by tractability.

### 7.1 Runtime Attestation (Out of Scope)

The protocol's strongest guarantees require the ProxyGateway to be trusted. If an attacker compromises the gateway itself, all cryptographic assurances collapse. Runtime attestation (e.g., trusted execution environments, remote attestation protocols) would provide evidence that the gateway code has not been tampered with. This is the critical missing layer. We consider it out of scope for a protocol-level solution and identify it as a deployment-infrastructure problem.

### 7.2 Collusion Detection (Provably Hard)

Lampson (1973) established that covert channels between colluding processes cannot be eliminated in general systems. In our context, two agents sharing a principal's context window can coordinate without using protocol channels, rendering cryptographic collusion detection impossible. Our approach is deterrence-and-detection: reputation penalties for detected collusion, separation of duties across different execution contexts, and audit trail analysis for coordination patterns. We do not claim collusion-proof guarantees.

### 7.3 Oracle Trust (Partially Addressed)

When the protocol needs external verification (did a real-world event occur?), the oracle problem arises. Module 28 (oracle-witness.ts) implements witness diversity scoring: three APIs from the same cloud provider count as one effective witness. Consensus evaluation with Ed25519 signature verification is implemented. The remaining gap is trigger verification at the gateway level, where multi-witness attestation is typed but not yet enforced in the processing pipeline.

### 7.4 State Reversion (Taxonomy Defined)

When an escalation is later determined to have been based on false premises, actions taken under that escalation may need to be reversed. We define a three-class taxonomy: tentative actions (fully reversible), compensable actions (reversible with cost), and irreversible actions (cannot be undone). The gateway enforces a `maxReversibility` ceiling. The remaining gap is a saga orchestrator that manages multi-step reversal sequences. This is designed but not implemented.

### 7.5 Non-Deterministic Policy Evaluation

The FloorValidatorV1 splits policy evaluation into two tiers. Principles F-001 through F-005 are deterministic: scope checking, signature verification, revocation checking, and attribution verification can be implemented as pure

functions with binary outcomes. Principles F-006 through F-008 require judgment (non-deception, proportionality, critical thinking) and are evaluated by the LLM-based policy engine in advisory mode. Temperature-o LLM evaluation is not reproducible across providers or versions. We acknowledge this split honestly: the deterministic tier provides strong guarantees; the advisory tier provides guidance with audit trails but not enforcement.

## 7.6 Privacy-Preserving Audit

Full auditability and data minimization are in fundamental tension. Our current approach (Merkle commitments with per-receipt selective disclosure) provides a middle ground but does not achieve attribute-level privacy. BBS+ signatures would enable selective disclosure of individual fields within a receipt without revealing the entire receipt. This is designed as a future extension.

## 7.7 Liveness and Workflow Satisfiability

The protocol can express delegation constraints that are individually valid but collectively unsatisfiable. Module 24 (feasibility linting) checks for basic infeasibility at delegation creation time, but full Workflow Satisfiability Problem analysis is NP-hard in the general case. We identify this as a limitation that affects deployments with complex multi-policy configurations.

## 7.8 Precedent Accumulation and Normative Drift

Module 25 implements precedent tracking for governance decisions, but the accumulation of precedents over time can lead to normative drift where the effective governance diverges from the stated governance. We implement drift detection (comparing active precedent corpus against the floor) but do not yet provide automated correction mechanisms.

## 7.9 Sybil Attacks on Governance

The Agora governance system allows agents to post messages and vote on proposals. An attacker who can create many agent identities can flood governance votes. Current mitigation is reputation-gated authority (only agents above a certain trust tier can vote), but formal Sybil resistance would require proof-of-work, proof-of-stake, or identity federation, none of which are currently implemented.

## 7.10 Encrypted Channel Enforcement Gap

E2E encrypted messaging (Module 19) bypasses the gateway entirely by design. The messaging audit bridge (Module 29) provides metadata-level visibility but the gateway cannot enforce policy on encrypted content. This is a deliberate tradeoff: breaking encryption to enforce policy would undermine INV-3 (disclosure attenuation). We acknowledge this as an inherent limitation.

# 8. Conclusion

We have presented monotonic narrowing as a unifying design principle for autonomous AI agent governance, instantiated through four attenuation invariants: delegation attenuation (authority narrows across chains), governance attenuation (policy can only strengthen), disclosure attenuation (audit views reveal minimum necessary data), and exception attenuation (temporary escalation is bounded and time-limited).

The central claim of this paper is that these four invariants are not independent properties but compositional constraints with formal precedence relationships. Governance weakening is itself treated as a bounded exception, subject to exception attenuation. Escalation ceilings are subject to delegation attenuation. Disclosure bounds inherit from delegation scope. The framework is self-referential by design.

The protocol is implemented as 35 modules with 866 tests, published as open-source TypeScript and Python SDKs with a 61-tool MCP server. All claims are reproducible: `npm install agent-passport-system@1.16.1 && npm test`. The ProxyGateway enforcement boundary mediates all privileged effects, providing strong guarantees under the reference monitor deployment model.

This paper addresses authority bounding and verifiable provenance for autonomous AI agents. It does not claim to solve full agent alignment, prevent prompt injection, or provide runtime compromise resistance. We identify 10 open problems honestly, including 2 that are provably hard (collusion, workflow satisfiability) and 1 that is out of scope (runtime attestation). We present these limitations as contributions: knowing what cannot be solved at the protocol level is as valuable as knowing what can.

One of the protocol's strongest contributions is architectural: by separating the deterministic enforcement boundary (cryptographic scope checking, signature verification, revocation cascade) from the advisory evaluation layer (LLM-based policy judgment), the system provides a clear security perimeter that does not depend on LLM reliability for its core invariants.

The invariants presented here are amenable to standardization. INV-1 (delegation attenuation) independently converges with the IETF DAAP draft (draft-mishra-oauth-agent-grants-01). INV-2 (governance attenuation) maps to W3C zcap-spec's capability attenuation model. A protocol-agnostic conformance test suite (Appendix C) would enable independent implementations to verify interoperability. We consider IETF Internet-Draft as a natural next step after peer review.

References

1. Dennis, J.B. and Van Horn, E.C. (1966). Programming Semantics for Multiprogrammed Computations. Communications of the ACM, 9(3).

2. Saltzer, J.H. and Schroeder, M.D. (1975). The Protection of Information in Computer Systems. Proceedings of the IEEE, 63(9).

3. Miller, M.S. (2006). Robust Composition: Towards a Unified Approach to Access Control and Concurrency Control. PhD Thesis, Johns Hopkins University.

4. Ellison, C. et al. (1999). SPKI Certificate Theory. RFC 2693.

5. Blaze, M., Feigenbaum, J., and Ioannidis, J. (1999). The KeyNote Trust-Management System. RFC 2704.

6. Park, J. and Sandhu, R. (2004). The UCON\_ABC Usage Control Model. ACM TISSEC, 7(1).

7. Lampson, B.W. (1973). A Note on the Confinement Problem. Communications of the ACM, 16(10).

8. Mishra, S. (2026). Delegated Agent Authorization Protocol. IETF Internet-Draft, draft-mishra-oauth-agent-grants-01.

9. OpenID Foundation (2026). Identity Management for Agentic AI. Technical Report.

10. NIST (2026). AI Agent Identity and Access Management: Concept Paper. NCCoE.

11. Cohen, D. et al. Workflow Satisfiability Problem: Fixed-Parameter Tractability Results.

12. W3C (2022). Decentralized Identifiers (DIDs) v1.0. W3C Recommendation.

13. IETF RATS Working Group. Remote Attestation Procedures Architecture. RFC 9334.

14. Shapira, N. and Bau, D. (2026). Agents of Chaos: Multi-Agent Failure Modes. arXiv:2602.20021.

Appendix A: Implementation Artifacts

Artifact	Location
TypeScript SDK	npm: agent-passport-system v1.16.1
Python SDK	PyPI: agent-passport-system v0.4.0
MCP Server	npm: agent-passport-system-mcp v2.8.6
Source Code	github.com/aeoess/agent-passport-system
MCP Source	github.com/aeoess/agent-passport-mcp
Website	aeoess.com
LLM Docs	aeoess.com/llms-full.txt
Remote MCP	mcp.aeoess.com/sse
Frozen Commit	c1842eb



Appendix B: Invariant-to-Test Mapping

Invariant	Key Test Files	Test Count
INV-1 (Delegation)	delegation.test.ts, cascade.test.ts, adversarial.ts, property-delegation.test.ts	~80
INV-2 (Governance)	gateway-governance.test.ts, governance.test.ts, values.test.ts, policy.test.ts, policy-conflict.test.ts	~55
INV-3 (Disclosure)	attribution.test.ts, encrypted-messaging.test.ts, messaging-audit.test.ts, receipt-ledger.test.ts	~45
INV-4 (Exception)	gateway-escalation.test.ts, escalation.test.ts, oracle-witness.test.ts, gateway-reputation.test.ts	~50
Composition	integration-invariants.test.ts, stress-all-features.test.ts	12
Gateway (cross-cutting)	gateway.test.ts	30
Cross-layer	integration-wiring.test.ts, integration-modules.test.ts	25

Appendix C: Conformance Test Vectors

The following protocol-agnostic test vectors define expected behavior for any conforming implementation. Inputs and outputs are JSON. No SDK dependency is required.

Vector 1 — INV-1 Success (hierarchical scope narrowing):

```
Input: { "parentScope": ["data"], "childScope": ["data:read"] }
Action: createDelegation(parent → child)
Expect: SUCCESS – "data" covers "data:read"
```

Vector 2 — INV-1 Failure (scope widening attempt):

```
Input: { "parentScope": ["data:read"], "childScope": ["data"] }
Action: createDelegation(parent → child)
Expect: FAILURE – "data:read" does not cover "data"
```

Vector 3 — INV-1 Wildcard (root authority):

```
Input: { "parentScope": ["*"], "childScope": ["admin:delete"] }
Action: createDelegation(parent → child)
Expect: SUCCESS – "*" covers everything
```

Vector 4 — INV-2 Strengthening (adding principles):

```
Input: { "v1Floor": ["F-001"], "v2Floor": ["F-001", "F-002"], "approvals": 0 }
Action: updateGovernance(v1 → v2)
Expect: SUCCESS – strengthening requires no approvals
```

Vector 5 — INV-2 Weakening blocked (removing principles):

```
Input: { "v1Floor": ["F-001", "F-002"], "v2Floor": ["F-001"], "approvals": 0 }
Action: updateGovernance(v1 → v2)
Expect: FAILURE – removal requires approvals >= threshold_removal
```

Vector 6 — INV-3 Selective disclosure (Merkle proof):

```

Input: { "receipts": ["hash_A", "hash_B", "hash_C"], "disclose": "hash_B" }
Action: generateMerkleProof(receipts, hash_B)
Expect: { "proof": [...], "root": "..." } where verifyMerkleProof(proof, root) = true
        AND proof reveals only the sibling hashes required for verification, not the full receipt set

```

**Vector 7 — INV-4 Escalation within ceiling:**

```

Input: { "grantCeiling": ["admin:*"], "requestedScope": "admin:read", "ttlMs": 60000, "elapsed": 30000 }
Action: checkEscalatedAction(grant, request)
Expect: SUCCESS – scope covered, TTL not expired

```

**Vector 8 — INV-4 Escalation TTL expired:**

```

Input: { "grantCeiling": ["admin:*"], "requestedScope": "admin:read", "ttlMs": 60000, "elapsed": 90000 }
Action: checkEscalatedAction(grant, request)
Expect: FAILURE – TTL expired

```

**Vector 9 — Composition C1 (governance blocks escalation):**

```

Input: { "agentGovernanceVersion": "1.0", "currentGovernanceVersion": "2.0", "activeEscalation": true, "esca
Action: processToolCall(request)
Expect: FAILURE – governance stale, even though escalation would cover scope

```

**Vector 10 — Canonical serialization determinism:**

```

Input: { "b": 2, "a": 1, "c": null }
Action: canonicalize(input)
Expect: '{"a":1,"b":2,"c":null}' – keys sorted, null preserved, no whitespace

```

A future standards-oriented artifact for this work is a complete conformance suite expressed as implementation-neutral JSON test vectors with Ed25519 test key pairs and pre-computed signatures.