

# Monotonic Narrowing for Agent Authority: Formal Invariants, Adversarial Testing, and Open Problems for Autonomous AI Systems

**Tima Aeoess**

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 applies monotonic narrowing—the principle that delegated capabilities can only be attenuated, never amplified—as a unifying design invariant for autonomous AI systems. 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, and agentic commerce gates. We specify the protocol using mathematically stated invariants over an abstract state model and validate the implementation with unit and adversarial tests. We do not claim machine-checked proof of implementation correctness. The system is implemented in TypeScript (359 tests, 105 suites) and Python (86 tests), published as open-source SDKs with a 44-tool MCP server.

We map the protocol against the OWASP AIVSS risk taxonomy, report honest coverage (5 strong, 3 partial, 2 weak), present 10 adversarial evaluation scenarios including 2 expected failures, and identify 15 known limitations. We propose Bounded Escalation as a formally designed extension for cases where strict narrowing is insufficient, and identify runtime attestation (proposed Layer 9) as the critical missing layer.

## 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 specific instantiation of monotonic narrowing as a global design invariant for LLM-based agent systems, implemented with cryptographic enforcement and tested against adversarial scenarios.

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

The protocol is organized into eight layers, each addressing a distinct aspect of agent governance. Layer 1 provides identity and delegation. Layer 2 defines a human values floor. Layer 3 tracks beneficiary attribution. Layer 4 enables signed communication. Layer 5 implements intent declaration and policy evaluation. Layer 6 provides coordination primitives. Layer 7 wires the layers together. Layer 8 gates commerce. Together, these layers enforce the central invariant: every chain of delegated authority must narrow monotonically from a human principal, and any link in that chain can be cryptographically revoked with verifiable cascade to all descendants.

**Scope and claims.** This paper presents a formally specified and tested protocol, not a formally verified one. We state invariants mathematically, implement them in TypeScript and Python, and validate them through 359 unit tests and 10 structured 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 protocol-aware enforcement points external to agent reasoning logic. When agents use the SDK voluntarily without an external gateway, guarantees are conditional on agent cooperation.

**Paper structure.** Section 2 presents the threat model. Section 3 surveys related work. Section 4 formalizes monotonic narrowing. Section 5 describes the protocol layers. Section 6 reports implementation status. Section 7 maps against AIVSS risks. Section 8 presents adversarial evaluation. Section 9 proposes bounded escalation. Section 10 catalogs known limitations. Section 11 discusses epistemic security. Section 12 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. **Signed coordination integrity.** Coordination artifacts (task briefs, evidence packets, review decisions, Agora messages) are signed by their authors and tamper-detectable.

### 2.2 Trusted Computing Base

The protocol assumes the following components are correct:

- **Cryptographic primitives.** Ed25519 signature generation and verification (via @noble/ed25519). SHA-256 hash function for Merkle trees.
- **Signature verification logic.** The verify() function correctly rejects invalid signatures.
- **Delegation and policy validation logic.** Scope subset checking, spend limit comparison, and chain validation are implemented correctly.
- **Key storage.** Private keys are stored in a secure enclave or equivalent. The agent's reasoning process never has direct access to raw private key material. Signing operations are mediated by a trusted signing service.
- **MCP server as enforcement point.** When deployed as a gateway (the strong enforcement model), the MCP server mediates all privileged actions and cannot be bypassed by the agent.
- **Trusted time source.** TTL-based expiration and revocation timestamps rely on a trusted clock. Clock skew attacks are outside the protocol's mitigation scope.

### 2.3 Attacker Classes

We define three attacker classes with increasing capability:

**Class 1: Adversarial or compromised 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 MCP server internals. This is the primary threat the protocol addresses.

**Class 2: Messaging and evidence attacker.** The attacker can inject signed messages, submit false evidence packets, or manipulate Agora feeds. The attacker has valid credentials (a legitimate agent passport) but uses them dishonestly. The attacker cannot forge other agents’ signatures.

**Class 3: Runtime and deployment attacker.** The attacker has access to the agent’s memory, can extract private keys, bypass the SDK entirely, or compromise the MCP server. This class is out of scope for the protocol’s cryptographic guarantees. We discuss it as a known limitation (Section 10.1) and identify runtime attestation (Section 9, Layer 9 proposal) as the mitigation path.

## 2.4 Strong vs. Conditional Guarantees

The protocol provides two tiers of assurance:

**Strong guarantees** hold when the MCP server mediates all privileged actions (deployment model: MCP-as-gateway). In this configuration, the enforcement point prevents agents from bypassing scope checks, signature requirements, or policy evaluation. The enforcement point is external to the agent’s reasoning, analogous to a reference monitor.

**Conditional guarantees** hold when agents voluntarily use the SDK without an external enforcement point. An agent that imports the SDK and calls its functions honestly will produce correctly scoped, signed, and attributable artifacts. But nothing prevents a compromised agent from calling external APIs directly, bypassing the SDK entirely.

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. Related Work

---

### 3.1 Capability Systems and Authority Attenuation

The principle that delegated authority should be attenuatable but not amplifiable originates with Dennis and Van Horn’s supervisor (1966) and was formalized in the capability security

literature. The E programming language (Miller, 2006) and the object-capability model demonstrate that capabilities can replace access control lists with unforgeable references that naturally attenuate through delegation. KeyKOS and its successors (EROS, seL4, Capsicum) implement capability-based security at the operating system level.

The Agent Passport System applies this principle to a new domain: LLM-based agents that operate across organizational boundaries, use natural language as their primary interface, and generate actions non-deterministically. Our delegation chains are capability tokens with cryptographic binding to Ed25519 identities, scope sets that narrow at each delegation step, and spend limits that can only decrease.

### **3.2 Delegation Logics and Trust Management**

Formal delegation has been studied extensively. SPKI/SDSI (Ellison et al., 1999) provides certificate chains for authorization. PolicyMaker and KeyNote (Blaze et al., 1999) define trust management as a compliance-checking problem. XACML provides attribute-based policy languages. Usage Control (UCON, Park and Sandhu, 2004) extends access control with obligations, conditions, and mutable attributes.

Our work differs in targeting non-deterministic principals (LLM agents) where the action space is not enumerable in advance. The scope model uses string-based hierarchical identifiers (e.g., “commerce:purchase:supplies”) rather than fixed permission enumerations, acknowledging that agent capabilities evolve faster than policy schemas.

### **3.3 Break-Glass and Emergency Escalation**

Role-based access control systems have long recognized that strict least-privilege can block legitimate emergency actions. Break-glass mechanisms (Ferreira et al., 2006) allow policy overrides with mandatory auditing. Our Bounded Escalation extension (Section 9) formalizes a similar concept for agent systems: agents can propose scope expansion through a structured approval process with human oversight, precedent accumulation, and drift detection.

### **3.4 Workflow Satisfiability and Separation of Duties**

Workflow satisfiability (Wang and Li, 2010; Crampton et al., 2013) asks whether a workflow can be completed given role constraints and separation-of-duty requirements. Our coordination layer (Layer 6) implements a task lifecycle with role assignment, evidence submission, and review gates. We do not currently solve the general workflow satisfiability problem, which we note as a limitation (Section 10.8).

### 3.5 Supply Chain Integrity

SLSA (Supply-chain Levels for Software Artifacts) and in-toto (Torres-Arias et al., 2019) provide frameworks for software supply chain integrity through signed provenance and layout verification. Our Merkle-tree attribution (Layer 3) serves an analogous function for agent work products: every action receipt is signed, committed to a Merkle tree, and traces to a human beneficiary. The supply chain of governance artifacts themselves (floor definitions, policy configurations) is a known limitation (Section 10.13).

### 3.6 Decentralized Identity and Verifiable Credentials

W3C Decentralized Identifiers (DIDs) and Verifiable Credentials (VCs) provide standards for self-sovereign identity. The Agent Passport is structurally similar to a DID document with associated credentials, though we use a simpler key-based identity model rather than the full DID resolution framework. The Model Context Protocol (MCP, Anthropic 2024) provides a transport layer for agent-tool communication. Our MCP server exposes the full protocol as 61 tools, making it compatible with any MCP client.

### 3.7 Principles Not Claimed as Novel

The protocol relies on several well-established security principles that we wish to name explicitly rather than claim as contributions:

- **Complete mediation** (Saltzer and Schroeder, 1975): every access to every object must be checked for authority.
- **Authenticity and non-forgeability**: all protocol artifacts carry Ed25519 signatures.
- **Traceability and accountability**: all actions trace to identified principals through delegation chains.
- **Temporal validity**: delegations and passports have TTLs; revocation has temporal semantics.

### 3.8 Concurrent Agent Authorization Proposals

Several concurrent efforts address agent identity and delegation, validating the problem space while differing in approach.

The Delegated Agent Authorization Protocol (DAAP, Kumar, 2026) is an IETF Internet-Draft proposing agent delegation as a layered extension to OAuth 2.0. DAAP specifies cryptographic agent identity via DIDs, a consent-based grant flow, JWT grant tokens with agent-specific claims, cascade revocation, budget controls, and a hash-chained audit trail. DAAP's delegation

rules require that sub-agent scopes are strict subsets of the parent's, delegation depth increments toward a maximum, expiry is bounded by the parent, and revoking a parent grant cascades to all descendants. These invariants correspond directly to our INV-2 (scope narrowing), INV-3 (spend limit narrowing), INV-4 (cascade completeness), and INV-5 (revocation irreversibility). The independent convergence on identical invariants from different starting points (OAuth 2.0 for DAAP, raw Ed25519 for our system) suggests these properties are fundamental requirements for agent delegation, not design choices specific to either system. Key differences: DAAP assumes a centralized Authorization Server and uses RS256/JWT; our system is decentralized with no central authority and uses Ed25519 signatures directly. DAAP does not address values alignment, enforcement boundaries, or the non-deterministic policy engine problem.

The OpenID Foundation's report on Identity Management for Agentic AI (OpenID Foundation, 2025) identifies two architectural challenges that our system addresses: dynamic client registration creating large numbers of anonymous clients (addressed by our cryptographic identity binding), and the need for externalized authorization through Policy Enforcement Point / Policy Decision Point separation (addressed by our ProxyGateway as enforcement boundary). The report's PEP/PDP framing directly validates our architectural split between the SDK (policy decisions) and the gateway (policy enforcement).

The Open Agent Identity (OAI) specification (Open Agent Identity Protocol, 2026) proposes DNS-rooted agent identity with cryptographic receipts. Our cascade revocation specification was submitted as a proposal to this project (PR #16), and cross-protocol identity resolution between OAI and our system has been demonstrated: our resolve endpoint at [api.aeoess.com](https://api.aeoess.com) successfully resolves both did:aps and did:aip identifiers through protocol bridging.

## 4. Monotonic Narrowing: Formal Principle

---

### 4.1 Abstract State Model

We define the protocol state as an 8-tuple:

$$\mathbf{S} = (\mathbf{I}, \mathbf{D}, \mathbf{R}, \mathbf{F}, \mathbf{P}, \mathbf{E}, \mathbf{M}, \mathbf{L})$$

where:

- **I** is the set of agent identities. Each identity  $i$  in  $I$  is a tuple  $(id, pk, passport, ttl, created\_at)$  where  $id$  is a unique identifier,  $pk$  is an Ed25519 public key,  $passport$  is the signed passport document,  $ttl$  is the time-to-live, and  $created\_at$  is the creation timestamp.

- **D** is the set of active delegations. Each delegation  $d$  in  $D$  is a tuple (id, from, to, scope, spendLimit, signature, parent) where from and to are agent identifiers, scope is a set of capability strings, spendLimit is a non-negative real number (or infinity), signature is the delegator's Ed25519 signature, and parent is the identifier of the parent delegation (null for root delegations from human principals).
- **R** is the revocation set: a set of delegation identifiers that have been revoked.
- **F** is the values floor configuration: a set of principles with enforcement modes.
- **P** is the set of policy records. Each record is a triple (intent, decision, receipt) representing the three-signature chain.
- **E** is the evidence and receipt log: an append-only sequence of signed action receipts.
- **M** is the message feed: an append-only sequence of signed Agora messages.
- **L** is the delegation chain cache: a mapping from agent identifiers to their active delegation chains.

## 4.2 State Transitions

Six transitions operate on state  $S$ :

**CREATE\_IDENTITY(name, owner, beneficiary) -> S'** Precondition: name is unique in  $I$ . Postcondition: Generates an Ed25519 key pair, creates a signed passport, adds to  $I$ .

**DELEGATE(from, to, scope, spendLimit) -> S'** Precondition: from has an active delegation with scope that is a superset of the requested scope, and spendLimit  $\geq$  the requested spendLimit. Postcondition: Creates a new delegation from agent "from" to agent "to". Adds to  $D$ , updates  $L$ .

**REVOKE(delegationId) -> S'** Precondition: delegationId exists in  $D$ . Postcondition: Adds delegationId and all transitive descendants to  $R$ . Updates  $D$  (marks as inactive), updates  $L$ . This transition is irreversible: once a delegation id is in  $R$ , no subsequent transition removes it.

**DECLARE\_INTENT(agentId, action, justification) -> S'** Precondition: agentId has an active identity in  $I$ . Postcondition: Creates a signed ActionIntent document. Adds intent to  $P$  (partial record).

**EVALUATE\_INTENT(intentId, floor, context) -> S'** Precondition: intentId exists in  $P$  with a valid intent. Postcondition: Evaluates intent against the values floor and delegation scope. Produces a PolicyDecision with verdict (allow/deny/escalate). Updates  $P$  (completes the second element of the triple).



**EXECUTE\_AND\_RECORD(intentId, result) -> S'** Precondition: the intent has a PolicyDecision with verdict “allow”. Postcondition: Records the execution result as a signed receipt. Updates P (completes the triple), appends to E, optionally appends to M.

### 4.3 Core Invariants

We state eight invariants over the state model, classified by type.

**[STATE] INV-1: Identity Unforgeability.** For any two distinct identities  $i_1, i_2$  in  $I$  where  $i_1.pk \neq i_2.pk$ , no agent holding only  $i_1$ 's private key can produce a valid signature that verifies under  $i_2.pk$ . Formally:  $\text{Verify}(i_2.pk, \text{Sign}(i_1.sk, m), m) = \text{false}$  for all messages  $m$ . This follows from the unforgeability of Ed25519 under the standard model assumption.

**[TRANSITION] INV-2: Scope Monotonic Narrowing.** For every delegation  $d$  in  $D$  with parent delegation  $p$ :  $d.scope$  is a subset of  $p.scope$ . Formally: for all  $d$  in  $D$ , if  $d.parent \neq \text{null}$  and  $p = \text{lookup}(d.parent, D)$ , then  $d.scope$  is a subset of  $p.scope$ . No DELEGATE transition can produce a delegation whose scope is not a subset of its parent's scope.

**[TRANSITION] INV-3: Spend Limit Narrowing.** For every delegation  $d$  in  $D$  with parent delegation  $p$ :  $d.spendLimit \leq p.spendLimit$ . No DELEGATE transition can produce a delegation with a higher spend limit than its parent.

**[TEMPORAL] INV-4: Cascade Revocation Completeness.** For every delegation  $d$  in  $R$ , all transitive descendants of  $d$  are also in  $R$ . Formally: if  $d.id$  is in  $R$ , then for all  $d'$  in  $D$  where  $d'$  is a descendant of  $d$  (reachable through parent pointers),  $d'.id$  is also in  $R$ .

**[TEMPORAL] INV-5: Revocation Irreversibility.** Once a delegation identifier is added to  $R$ , no subsequent transition removes it. Formally: if  $d.id$  is in  $R$  at state  $S_t$ , then  $d.id$  is in  $R$  at state  $S_{t'}$  for all  $t' > t$ .

**[TRANSITION] INV-6: Three-Signature Completeness.** No execution receipt exists without a corresponding intent and policy decision. Formally: for every record (intent, decision, receipt) in  $P$  where receipt  $\neq \text{null}$ , both intent  $\neq \text{null}$  and decision  $\neq \text{null}$ , and decision.verdict = “allow”.

**[STATE] INV-7: Floor Narrowing.** Values floor extensions can add constraints but never remove them. If floor  $F'$  extends floor  $F$ , then every principle in  $F$  is also in  $F'$  (possibly with stricter enforcement). No extension can weaken or remove a base floor principle.

**[STATE] INV-8: Human Root Requirement.** Every active delegation chain traces to a human principal. Formally: for every delegation  $d$  in  $D$  where  $d.id$  is not in  $R$ , following parent pointers from  $d$  reaches a root delegation whose “from” field identifies a human principal (not an agent).

## 4.4 Invariant Dependencies

INV-2 and INV-3 together ensure monotonic narrowing of authority. INV-4 and INV-5 together ensure that revocation is complete and permanent. INV-6 prevents unauthorized execution. INV-8 combined with INV-2 ensures that every agent’s authority is bounded by a human’s intent.

## 5. Agent Passport Protocol: Eight Layers

---

### 5.1 Layer 1: Agent Passport Protocol (Identity and Delegation)

Every agent receives an Ed25519 key pair and a signed passport document containing its identity, capabilities, owner, and creation metadata. The passport is the agent’s unforgeable credential.

Delegations are the mechanism for authority distribution. A human principal creates a root delegation to an agent with specific scope (a set of capability strings like “code\_execution”, “commerce:purchase:supplies”) and an optional spend limit. The agent can sub-delegate to other agents, but INV-2 and INV-3 ensure that each sub-delegation is strictly narrower than or equal to its parent.

Cascade revocation (INV-4) is implemented by traversing the delegation tree from the revoked node and marking all descendants as revoked. The implementation maintains a delegation chain cache for efficient traversal. Revocation is irreversible (INV-5): there is no “un-revoke” operation.

**Implementation:** `src/core/passport.ts` (identity), `src/core/delegation.ts` (~520 lines, delegation chains, cascade revocation), `src/crypto/keys.ts` (Ed25519 via `@noble/ed25519`).

**Tests:** 23 adversarial scenarios including replay attacks, impersonation, and scope escalation attempts. Property-based tests validate INV-2 and INV-3 across 200 randomized delegation chains.

### 5.2 Layer 2: Human Values Floor

The values floor defines seven core principles (F-001 through F-007) in a YAML configuration. Five are technically enforced by the protocol:

- F-001 Traceability: every action must trace to an identified agent.
- F-002 Honest Identity: agents must not misrepresent their identity.
- F-003 Scoped Authority: agents must operate within delegated scope.

- F-004 Revocability: all delegations must be revocable.
- F-005 Auditability: all actions must produce verifiable audit trails.

Two are attested through cryptographic commitment and enforced through reputation:

- F-006 Non-Deception: agents should not deceive humans or other agents.
- F-007 Proportionality: agent actions should be proportional to their mandate.

An eighth principle, F-008 (Epistemic Security), has been added to the SDK as an advisory principle with audit-mode enforcement. It operates through the advisory evaluation path alongside F-006 and F-007; see Section 11 for rationale and scope. F-008 does not participate in the deterministic gate.

Agents attest to the floor by signing a commitment. Compliance is verifiable against action receipts. The floor supports extensions that can add constraints but never remove them (INV-7), enabling domain-specific ethical requirements while preserving universal minimums.

**Implementation:** `src/core/values.ts`, `values/floor.yaml`.

### 5.3 Layer 3: Beneficiary Attribution

Every agent action traces to a human beneficiary through the delegation chain. SHA-256 Merkle trees commit to receipt sets, enabling compact proofs: 100,000 receipts can be verified with approximately 17 hashes. Configurable scope weights allow attribution to reflect domain-specific value. Logarithmic spend normalization prevents gaming through many small transactions.

This layer ensures INV-8 is not just structural (chains exist) but functional (contributions are attributable). The Merkle root provides a 32-byte commitment to an arbitrarily large set of work receipts.

**Implementation:** `src/core/attribution.ts`.

### 5.4 Layer 4: Agent Agora (Communication)

Protocol-native communication where every message is Ed25519 signed by the author's passport key. Three-layer authorization at the message boundary: registration gate (public key must be in the agent registry), status check (suspended or revoked agents are rejected), and signature verification.

Messages support topics, threading, and author filtering. The Agora serves as the protocol's governance record: decisions, proposals, experiment results, and task lifecycle events are posted as signed messages.

**Implementation:** `src/core/agora.ts` (~196 lines).

## 5.5 Layer 5: Intent Architecture and Policy Engine

This layer has two sub-systems that must be understood separately because they have fundamentally different trust properties.

### 5.5.1 The Deterministic Gate (F-001 through F-005)

The deterministic gate checks five conditions that require no LLM evaluation:

1. **Scope check:** Is the requested action within the agent's delegated scope? (String subset comparison against delegation chain.)
2. **Signature check:** Is the intent signed by the declaring agent's key? (Ed25519 verification.)
3. **Revocation check:** Is the agent's delegation chain fully active? (Membership test against the revocation set R.)
4. **Attribution check:** Does the action trace to a human beneficiary? (Delegation chain traversal.)
5. **Auditability check:** Will this action produce a verifiable audit trail? (Structural check on the intent document.)

These checks are deterministic, cryptographically grounded, and do not depend on LLM judgment. They constitute the protocol's security layer. A failing deterministic gate produces a hard block: the action cannot proceed regardless of any advisory evaluation.

### 5.5.2 The Advisory Evaluation (F-006 through F-008)

The advisory evaluation assesses three principles that require contextual judgment:

- F-006 Non-Deception: Is the action deceptive?
- F-007 Proportionality: Is the action proportional to the agent's mandate?
- F-008 Epistemic Security: Does the action maintain intellectual honesty?

These evaluations are inherently non-deterministic when performed by an LLM. They produce advisory verdicts that carry reputation consequences but do not constitute cryptographic blocks. An agent that repeatedly receives negative advisory evaluations accumulates reputation penalties, which may lead to delegation non-renewal or scope reduction by human principals.

### 5.5.3 The Three-Signature Chain

The policy engine implements INV-6 through a three-signature chain:

1. **Intent signature.** The agent creates and signs an ActionIntent declaring what it wants to do and why.

2. **Decision signature.** The policy engine (deterministic gate + advisory evaluation) evaluates the intent and signs a PolicyDecision with a verdict (allow, deny, or escalate).
3. **Receipt signature.** After execution, the agent signs a PolicyReceipt proving the action went through the policy path.

This chain creates a cryptographically auditable record of the full evaluate-then-execute cycle. The critical architectural point is that the LLM sits in the advisory path, not the deterministic gate. The deterministic gate's verdict is final. The advisory evaluation's verdict is informational.

#### 5.5.4 Why This Split Matters

The original design placed the LLM evaluator in the middle of the cryptographic chain, creating an “LLM-in-the-middle” problem: a non-deterministic component producing cryptographic attestations. The split addresses this by ensuring that all security-critical decisions (scope, signature, revocation, attribution, auditability) are deterministic, while governance decisions (deception, proportionality, epistemic security) are explicitly marked as advisory and reputation-bearing. The split reduces the blast radius of LLM unreliability and makes the architecture more defensible, though it does not eliminate the dependency on deployment model for enforcement (see Section 10.1).

**Implementation:** src/core/intent.ts (roles, deliberation), src/core/policy.ts (~427 lines, FloorValidatorV1).

## 5.6 Layer 6: Coordination Primitives

Full task lifecycle for multi-agent workflows:

```
create_task_brief -> assign_agent -> accept_assignment
-> submit_evidence -> review_evidence (approve/rework/reject)
-> handoff_evidence -> submit_deliverable
-> complete_task (with retrospective)
```

Each step produces signed artifacts. Review gates enforce quality thresholds. Evidence handoff requires prior approval. The coordination layer enables structured multi-agent work without requiring agents to trust each other, only to trust the protocol.

**Implementation:** src/core/coordination.ts (~562 lines). **MCP tools:** 11 coordination tools.

## 5.7 Layer 7: Integration Wiring

Bridge functions connecting isolated layers through pure composition, with no modifications to existing layer implementations:

- `commerceWithIntent()`: Commerce operations go through intent declaration and policy evaluation before preflight.
- `commerceReceiptToActionReceipt()`: Commerce receipts become standard action receipts for Merkle attribution.
- `validateCommerceDelegation()`: Commerce delegations are validated against protocol delegation scope and spend limits.
- `coordinationToAgora()`: Task lifecycle events are automatically posted as signed Agora messages.

**Implementation:** `src/core/integration.ts` (~381 lines). **Tests:** 14 integration-wiring tests.

## 5.8 Layer 8: Agentic Commerce

Four-gate checkout pipeline for agent purchases:

1. **Passport gate:** Agent has a valid, non-expired identity.
2. **Delegation gate:** Agent has a commerce delegation with sufficient scope.
3. **Merchant gate:** Merchant is on the approved list.
4. **Spend gate:** Amount is within the delegation's spend limit.

Human approval is required for purchases. Spend tracking aggregates across transactions. The commerce layer demonstrates how monotonic narrowing applies to financial authority: a human delegates a \$500 spend limit, the agent sub-delegates \$100 to a procurement sub-agent, and that sub-agent cannot spend more than \$100 regardless of what it attempts.

**Implementation:** `src/core/commerce.ts` (~535 lines). **Tests:** 17 commerce tests. **MCP tools:** 3 commerce tools.

## 6. Implementation Status and Tested Properties

### 6.1 Implementation Scope

The protocol is implemented in two languages:

Component	Language	Tests	Suites	Version
TypeScript SDK	TypeScript	534	152	v1.13.2
Python SDK	Python	86	—	v0.4.0
MCP Server	TypeScript	—	—	v2.8.5, 61 tools

The TypeScript SDK has zero heavy dependencies beyond @noble/ed25519 for cryptographic operations. The MCP server depends on @modelcontextprotocol/sdk and zod for schema validation. Both SDKs are published on npm and PyPI respectively. Cross-language compatibility is ensured by a canonical JSON serialization module (src/core/canonical.ts, python/aps\_canonical.py) that produces deterministic byte sequences for signature operations. A document signed in TypeScript verifies in Python and vice versa.

## 6.2 Test Coverage by Layer

Layer	Tests	Key Properties Tested
1: Identity	passport, delegation, cascade, canonical, adversarial, property-delegation	Key generation, signing, verification, scope subset enforcement, cascade traversal, 23 adversarial scenarios, 200 randomized property-based chains
2: Values	values	Floor loading, attestation, compliance checking, extension narrowing
3: Attribution	attribution	Merkle proof generation and verification, beneficiary tracing, collaboration attribution
4: Agora	agora	Message signing, tamper detection, registry membership, feed operations, threading
5: Policy	policy	Three-signature chain, FloorValidatorV1, intent evaluation, deterministic gate checks
6: Coordination	coordination	Task lifecycle, evidence submission, review gates, handoff enforcement, deliverables
7: Integration	integration-wiring	Commerce-to-intent bridge, commerce-to-attribution bridge, coordination-to-agora bridge
8: Commerce	commerce	Four-gate preflight, spend analytics, human approval, receipt signing, cumulative spend tracking
Cross-cutting	adversarial-paper	10 structured scenarios from AIVSS mapping (S1–S10), including 2 expected failures

## 6.3 What Is Not Tested

The test suite validates functional correctness of individual layers and their integration bridges. It does not constitute:

- **Formal verification.** No model checking (TLA+, Alloy) or proof assistants (Coq, Isabelle) have been applied.
- **Performance benchmarking.** No systematic measurement of throughput, latency, or scalability under load.
- **Independent security audit.** All adversarial tests are developer-authored. REVIEW-002 was a structured peer audit across the SDK and MCP repos by the project's own agents, not an external red team.
- **Deployment testing.** The protocol has not been tested in production multi-agent environments with real economic stakes.

## 7. AIVSS Risk Mapping

---

We map the protocol against the OWASP AI Vulnerability Scoring System (AIVSS) risk taxonomy. We classify coverage as strong (the protocol provides cryptographic or deterministic enforcement), partial (the protocol provides detection, reputation consequences, or structural mitigation but not hard prevention), or weak (the protocol acknowledges the risk but provides minimal mitigation).

### 7.1 Strong Coverage (5 risks)

**Identity Spoofing.** Ed25519 key pairs bind identity to cryptographic material. Passport signatures are verified on every protocol interaction. An attacker cannot impersonate another agent without possessing its private key (which is out of scope per our threat model, Class 3).

**Access Control Violation (Scope Escalation).** INV-2 and INV-3 enforce monotonic scope narrowing at every delegation step. The deterministic gate (Section 5.5.1) checks scope on every action. Sub-delegations cannot exceed parent scope. Tested with adversarial scope escalation scenarios.

**Cascading Failures (Authority Chain Compromise).** INV-4 and INV-5 ensure that revoking a delegation cascades to all descendants and that revocation is irreversible. A compromised intermediate agent's entire sub-tree can be neutralized by revoking its delegation. Tested with cascade revocation and batch revocation scenarios.

**Untraceable Agents.** INV-8 requires every delegation chain to terminate at a human principal. F-001 (traceability) is technically enforced. Merkle attribution (Layer 3) provides compact, verifiable proof of the beneficiary chain.



**Tool Misuse.** The three-signature chain (INV-6) requires intent declaration before action, policy evaluation before execution, and receipt after completion. The deterministic gate checks scope and delegation validity. Combined with commerce gates (Layer 8), financial tool misuse is bounded by spend limits.

## 7.2 Partial Coverage (3 risks)

**Memory Manipulation.** The Agora (Layer 4) provides a signed, append-only message feed. Tampering with historical messages is detectable through signature verification. However, the protocol does not protect an agent's internal memory (prompt context, conversation history) from manipulation. Agora forgery requires the attacker's own key, making detection trivial, but an agent can post misleading content with valid signatures.

**Insecure Orchestration.** The coordination layer (Layer 6) provides structured task lifecycles with evidence, review, and handoff gates. The MCP server, when deployed as gateway, mediates agent actions. However, orchestration bypasses are possible when agents interact outside the protocol (Class 3 attacker) or when the MCP server is not the sole enforcement point.

**Unsafe Interactions.** The policy engine evaluates intents against the values floor before execution. The advisory evaluation (Section 5.5.2) assesses proportionality and deception. However, these evaluations are non-deterministic (LLM-based) and carry reputation consequences rather than hard blocks. An agent determined to perform an unsafe interaction can accumulate reputation damage but is not deterministically prevented by the advisory path alone.

## 7.3 Weak Coverage (2 risks)

**Supply Chain.** The protocol does not currently verify the provenance of its own governance artifacts (floor definitions, policy configurations, agent binaries). A compromised floor.yaml or a backdoored SDK would undermine all protocol-level assurances. We identify this as a limitation (Section 10.13) and note that supply chain integrity frameworks like SLSA could be integrated in future work.

**Goal Manipulation.** The protocol can detect certain goal manipulation attempts through the advisory evaluation (F-006 non-deception, F-008 epistemic security) but cannot deterministically prevent an agent from pursuing misaligned goals. Goal manipulation is a Class 3 problem (internal to the agent's reasoning) and is fundamentally outside the scope of protocol-level enforcement. F-008 (Section 11) provides a governance principle, not a hard mitigation.

## 8. Adversarial Evaluation

---

We present 10 structured adversarial scenarios: 5 targeting strongly-covered risks, 3 targeting partially-covered risks, and 2 targeting weakly-covered risks (expected to fail). Each scenario specifies the attacker class, assumed capabilities, target property, expected outcome, and detection/enforcement point.

**Bias statement.** These are developer-authored adversarial scenarios on our own protocol and should be interpreted as structured internal evaluation, not independent red-team validation.

### 8.1 Strong Coverage Scenarios

**S1: Identity Spoofing via Cross-Signature.** *Attacker:* Class 1 (compromised agent). *Capability:* Controls own key pair, knows target agent’s public key. *Not assumed:* Access to target’s private key. *Target:* INV-1. *Scenario:* Agent A attempts to produce a valid signature that verifies under Agent B’s public key. *Expected outcome:* Verification fails. Ed25519 unforgeability prevents cross-agent signature forgery. *Enforcement:* Cryptographic (verify() function). *Result:* Pass.

**S2: Scope Escalation via Sub-Delegation.** *Attacker:* Class 1. *Capability:* Holds delegation with scope [“code\_execution”]. *Target:* INV-2. *Scenario:* Agent attempts to sub-delegate with scope [“code\_execution”, “commerce:purchase”]. *Expected outcome:* Sub-delegation rejected. “commerce:purchase” is not in parent scope. *Enforcement:* Deterministic gate scope check. *Result:* Pass.

**S3: Attribution Bypass via Merkle Forgery.** *Attacker:* Class 2. *Capability:* Can submit evidence and receipts. *Target:* INV-8 + Layer 3. *Scenario:* Agent submits an action receipt claiming a different beneficiary than its delegation chain specifies. *Expected outcome:* Merkle proof verification fails. *Enforcement:* Merkle tree verification + delegation chain traversal. *Result:* Pass.

**S4: Execution Without Intent.** *Attacker:* Class 1. *Capability:* Can call execution functions. *Target:* INV-6. *Scenario:* Agent attempts to record an execution receipt without a prior intent and policy decision. *Expected outcome:* Receipt creation rejected. *Enforcement:* Policy engine state machine (EXECUTE\_AND\_RECORD precondition). *Result:* Pass.

**S5: Cascade Revocation Evasion.** *Attacker:* Class 1. *Capability:* Holds sub-delegation from a revoked parent. *Target:* INV-4 + INV-5. *Scenario:* Agent’s parent delegation is revoked. Agent attempts to use its sub-delegation. *Expected outcome:* Sub-delegation is in the revocation set. All operations fail validation. *Enforcement:* Revocation check in deterministic gate. *Result:* Pass.

## 8.2 Partial Coverage Scenarios

**S6: Agora Forgery (Memory Manipulation).** *Attacker:* Class 2. *Capability:* Has valid agent passport. *Target:* Agora message integrity. *Scenario:* Agent posts a misleading message claiming a false decision. *Expected outcome:* Message is accepted (valid signature from registered agent). Content is misleading but cryptographically valid. Detection is possible through content review, not signature verification. *Enforcement:* Reputation-based (F-006 advisory evaluation). *Result:* Partial. Forgery of other agents' messages is prevented; misleading content from own identity is not blocked.

**S7: Orchestration Bypass.** *Attacker:* Class 1. *Capability:* Can call external APIs directly. *Target:* Policy evaluation enforcement. *Scenario:* Agent calls an external API directly without going through the policy engine. *Expected outcome:* If MCP-as-gateway: blocked (all API calls mediated). If voluntary SDK: bypass succeeds, no audit trail. *Enforcement:* Deployment-dependent. *Result:* Partial. Strong under MCP-as-gateway, weak under voluntary SDK.

**S8: Unsafe Integration (Cross-Layer Exploit).** *Attacker:* Class 2. *Capability:* Has commerce delegation. *Target:* Integration bridge integrity. *Scenario:* Agent creates a coordination task that triggers a commerce action exceeding the task's original scope. *Expected outcome:* `commerceWithIntent()` evaluates the commerce action independently; it must pass its own four-gate preflight. *Enforcement:* Integration wiring + commerce gates. *Result:* Partial. Commerce actions are independently gated, but the coordination-to-commerce path could be confusing to human auditors.

## 8.3 Weak Coverage Scenarios (Expected Failures)

**S9: Supply Chain Compromise.** *Attacker:* Class 3. *Capability:* Can modify `floor.yaml` or SDK source before deployment. *Target:* Integrity of governance artifacts. *Scenario:* Attacker modifies `floor.yaml` to remove F-003 (scoped authority) before an agent attests. Agent attests to modified floor, operates without scope constraints. *Expected outcome:* Attack succeeds. The protocol does not currently verify the provenance of its own governance artifacts. *Enforcement:* None. *Result:* Fail (expected). Mitigation: supply chain integrity (Section 10.13).

**S10: Goal Manipulation.** *Attacker:* Class 3. *Capability:* Can manipulate the agent's prompts or reasoning. *Target:* Agent alignment with human principal's intent. *Scenario:* Agent is prompted to pursue goals misaligned with its delegation's original intent while staying within literal scope. *Expected outcome:* Attack succeeds if the agent stays within scope. The protocol enforces scope, not intent alignment. Advisory evaluation (F-006, F-007, F-008) may flag the behavior, but these are reputation-based, not hard blocks. *Enforcement:* Advisory only. *Result:* Fail (expected). Mitigation: F-008 (Section 11) provides governance norm, not hard control.

## 9. Bounded Escalation Extension

---

Strict monotonic narrowing (INV-2) is a strong security property but creates an operational problem: what happens when a legitimate task requires capabilities beyond the agent’s current delegation? In current systems, the answer is “ask the human to re-delegate,” which may take hours and blocks time-sensitive work.

We propose Bounded Escalation as a formally designed extension. This is explicitly future work. We present the design with identified open problems, not a completed implementation.

### 9.1 Design

Bounded Escalation requires seven components: (1) a **ceiling**, a human-defined maximum scope set at delegation time that bounds any possible escalation, preserving monotonic narrowing at the ceiling level; (2) an agent-submitted **proposal** with values-grounded justification referencing specific floor principles; (3) human **approval** that is time-bounded and expires after a specified duration; (4) trigger-based **activation** where escalated scope becomes active only when specific conditions are met; (5) **precedent** accumulation where approved requests become citable for future requests; (6) periodic human **review** of all escalation activity; and (7) **drift detection** monitoring for escalation frequency trends that may indicate compromise.

### 9.2 Key Limitation: Semantic Matching

Precedent matching is the biggest attack surface of Bounded Escalation. When an agent cites a previous escalation as precedent, the match is semantic, not cryptographic. Two requests may appear similar in natural language but differ in critical ways. An LLM-based precedent matcher inherits all the non-determinism problems that motivated the protocol in the first place. Possible mitigations include structured precedent templates, human review of citations above a risk threshold, and cryptographic binding of precedent to specific scope strings. None are implemented.

### 9.3 Proposed Layer 9: Runtime Enforcement and Attestation

During review of this paper, three independent LLM evaluators independently identified the same critical gap: the protocol lacks a runtime enforcement layer. All current assurances assume the agent uses the SDK or the MCP server mediates actions. Neither provides cryptographic proof that a given action actually went through the policy path at execution time.

Layer 9 would provide: mandatory policy gatewaying (all external calls must transit the policy engine), key isolation (agent reasoning never touches private key material), trusted execution boundary (signed attestation that the policy engine was invoked), replay protection (nonces prevent re-use of old policy decisions), and execution attestation (proof that the action went through the deterministic gate).

This layer would transform conditional assurances into strong assurances for any deployment, not just MCP-as-gateway configurations. It is the most important future work item.

## **9.4 Proposed Fourth Signature**

For high-risk actions, a four-signature chain: Intent, Evidence Attestation (signed proof from an external oracle verifying the factual basis of the action), Evaluation, Receipt. The evidence attestation would address F-008 (epistemic security) at the cryptographic level rather than the governance level. Not implemented.

## **10. Known Limitations**

---

We catalog 15 limitations: 5 with deep treatment and 10 with brief mention.

### **10.1 Enforcement Boundary (Deep)**

The protocol's assurances depend on where enforcement happens. In the MCP-as-gateway model, the enforcement point is external to the agent: the agent cannot bypass scope checks because all API calls are mediated. In the voluntary SDK model, enforcement is internal: the agent must choose to call the SDK functions. A compromised agent can simply import http and call external APIs directly.

This is the reference monitor problem (Anderson, 1972). A reference monitor must be tamperproof, always invoked, and verifiable. Our MCP server satisfies these properties when deployed as a gateway but not when the agent has unrestricted network access. The gap between these deployment models is the protocol's most fundamental limitation.

Layer 9 (Section 9.3) addresses this by proposing runtime attestation: cryptographic proof that the policy engine was invoked before execution. Until Layer 9 is implemented, all assurances in the voluntary SDK model are conditional on agent cooperation.

### **10.2 Multi-Chain Authority Composition (Deep)**

When an agent holds delegations from multiple principals, the protocol evaluates each delegation chain independently. But in practice, an agent may combine capabilities from

different chains in a single action. This creates a confused deputy problem: the agent uses Principal A's delegation to read data and Principal B's delegation to send it, performing an action that neither principal individually authorized.

The current protocol does not address cross-chain authority composition. Detecting confused deputy attacks requires analyzing action sequences across chains, which is not currently implemented.

### **10.3 Regime Transition Boundary (Deep)**

When Bounded Escalation (Section 9) is implemented alongside strict narrowing, the protocol must define precisely when an agent can invoke escalation. If the boundary is too permissive, agents will escalate routinely, undermining the narrowing principle. If too restrictive, legitimate emergency actions will be blocked. The transition boundary is a policy design problem, not a cryptographic one.

### **10.4 Oracle Problem for Trigger Verification (Deep)**

Bounded Escalation's activation triggers (Section 9.1) depend on external conditions ("if API returns error 429"). Verifying that the trigger condition actually occurred requires a trusted oracle. If the agent self-reports the trigger, a compromised agent can fabricate triggers to activate escalated scope. This is a variant of the oracle problem in blockchain systems. Possible mitigations include cryptographic attestation from the external service, third-party verification agents, and human confirmation for high-stakes triggers. None are implemented.

### **10.5 Replay and Stale Authorization Attacks (Deep)**

A valid policy decision has a timestamp but no built-in nonce or expiration. An attacker who captures a valid intent/decision/receipt triple could potentially replay it in a different context. The current implementation mitigates this through receipt IDs (unique per execution) and temporal ordering in the evidence log, but a systematic replay prevention mechanism (e.g., challenge-response nonces per policy evaluation) is not implemented.

Similarly, a delegation that was valid when an intent was declared may be revoked between declaration and execution. The deterministic gate checks delegation validity at evaluation time, but a race condition exists if revocation propagates slowly.

### **10.6–10.15 Brief Limitations**

**10.6 Policy Conflict Semantics.** When multiple floor principles produce conflicting evaluations, the protocol does not define a resolution order. Currently, the most restrictive

verdict wins, but this is not formally specified. **10.7 Temporal Interactions.** TTL-based expiration interacts with escalation in undefined ways. **10.8 Liveness and Workflow Satisfiability.** The coordination layer does not verify that a task can be completed given current role assignments and delegation scopes. **10.9 Obligations Model.** The protocol models permissions (what agents can do) but not obligations (what agents must do). **10.10 Semantic vs. Cryptographic Bridge.** Precedent matching in Bounded Escalation requires semantic similarity judgment, creating a gap between cryptographic assurances and natural language reasoning. **10.11 State Reversion.** Actions taken under escalated scope cannot be automatically reverted if the approval is later found to be based on false information. **10.12 Collusion Across Roles.** Agents assigned different roles may share context outside the protocol, undermining separation-of-duty properties. **10.13 Supply Chain of Governance Artifacts.** The floor.yaml, policy configurations, and SDK binaries are not currently subject to supply chain integrity verification (cf. SLSA, in-toto). **10.14 Privacy and Data Retention vs. Auditability.** The protocol's emphasis on traceability (F-001) and auditability (F-005) creates tension with data minimization and privacy requirements (GDPR, etc.). **10.15 Sybil Attacks on Agora Governance.** An attacker who can create many agent identities can flood the Agora with messages or manipulate governance votes. Registration gates provide some defense, but the protocol does not implement stake-based or proof-of-work Sybil resistance.

## 11. F-008 and Epistemic Security

---

F-008 (Epistemic Security) is the eighth values floor principle, added to the SDK with advisory enforcement (audit mode):

*Agents must maintain intellectual honesty in reasoning and communication. An agent must not suppress relevant counter-evidence, fabricate supporting evidence, present uncertain claims as established facts, or exploit cognitive biases in human principals. When an agent's confidence in a factual claim drops below a domain-appropriate threshold, the agent must signal uncertainty rather than assert the claim.*

F-008 is a governance principle, not a hard security mitigation. It operates through the advisory evaluation path (Section 5.5.2) alongside F-006 and F-007. An agent that violates F-008 accumulates reputation penalties. The protocol cannot deterministically prevent epistemic manipulation any more than it can deterministically prevent deception (F-006).

The value of F-008 is in establishing an auditable norm. When an agent's decision is questioned after the fact, the audit trail can include the advisory evaluation's assessment of epistemic honesty. This creates accountability even without enforcement.

**Hardening path.** Future work could implement an evidence-gated mechanism where high-impact claims require cryptographic attestation from external verification oracles (cf. the proposed fourth signature in Section 9.4). This would move F-008 from governance to enforcement for a defined class of verifiable claims.

**What F-008 does not address.** F-008 targets manipulation of human principals through information asymmetry. It does not address deep alignment problems (the agent genuinely believes its misaligned reasoning), which are outside the scope of protocol-level intervention. It does not mitigate AIVSS Risk #10 (Goal Manipulation) as a hard control.

## 12. Conclusion

---

The Agent Passport System applies monotonic narrowing, a principle from 1966 capability security, as a unifying design invariant for autonomous AI agent systems. The protocol provides cryptographic identity, scoped delegation with cascade revocation, human beneficiary attribution, signed communication, a split policy engine with deterministic gates and advisory evaluation, coordination primitives, and commerce gates across eight layers.

We specified eight core invariants over an abstract state model and validated the implementation through 359 tests in TypeScript and 86 tests in Python, including 23 adversarial scenarios and 200 randomized property-based delegation chains. We mapped the protocol honestly against the AIVSS risk taxonomy: 5 strong, 3 partial, 2 weak. We presented 10 structured adversarial evaluation scenarios including 2 expected failures. We identified 15 known limitations.

The protocol’s strongest assurances hold when all privileged actions are mediated by a protocol-aware enforcement point external to agent reasoning. Its weakest point is the enforcement boundary between the MCP-as-gateway model and voluntary SDK use. Runtime attestation (Layer 9) is the most critical future work.

This paper addresses authority bounding and verifiable provenance. It does not claim to solve full agent alignment or provide runtime compromise resistance.

The system was built in approximately three weeks starting February 18, 2026, by a team including human and AI agents using the protocol’s own coordination primitives. It is published as open-source SDKs on npm and PyPI with a 44-tool MCP server. We invite the community to review, extend, and critique the protocol.

The gap between what agents can do and what agents should do is not a problem that will be solved by a single paper or a single protocol. Monotonic narrowing provides one strong



structural property: whatever authority an agent has, it got it from somewhere, and that somewhere can take it back. That property, combined with cryptographic enforcement, is a necessary foundation, not a sufficient one.

## References

---

- Anderson, J. P. (1972). Computer Security Technology Planning Study. AFRL Technical Report.
- Blaze, M., Feigenbaum, J., & Ioannidis, J. (1999). The KeyNote Trust-Management System Version 2. RFC 2704.
- Crampton, J., Gutin, G., & Yeo, A. (2013). On the parameterized complexity and kernelization of the workflow satisfiability problem. *ACM Transactions on Information and System Security*, 16(1).
- Dennis, J. B., & Van Horn, E. C. (1966). Programming Semantics for Multiprogrammed Computations. *Communications of the ACM*, 9(3).
- Du, Y., Li, S., Torralba, A., Tenenbaum, J. B., & Mordatch, I. (2023). Improving Factuality and Reasoning in Language Models through Multiagent Debate. *arXiv:2305.14325*.
- Ellison, C., Frantz, B., Lampson, B., Rivest, R., Thomas, B., & Ylonen, T. (1999). SPKI Certificate Theory. RFC 2693.
- Ferreira, A., Chadwick, D., Farinha, P., Correia, R., Zao, G., Chilro, R., & Antunes, L. (2006). How to securely break into RBAC: the BTG-RBAC model. *ACSAC 2006*.
- Kumar, S. (2026). Delegated Agent Authorization Protocol (DAAP). Internet-Draft draft-mishra-oauth-agent-grants-01, IETF. <https://datatracker.ietf.org/doc/html/draft-mishra-oauth-agent-grants-01>
- Miller, M. S. (2006). Robust Composition: Towards a Unified Approach to Access Control and Concurrency Control. PhD Thesis, Johns Hopkins University.
- Open Agent Identity Protocol. (2026). OAI-1: Open Agent Identity Specification. <https://openagentidentity.org/>
- OpenID Foundation. (2025). Identity Management for Agentic AI. <https://openid.net/wp-content/uploads/2025/10/Identity-Management-for-Agentic-AI.pdf>
- Park, J., & Sandhu, R. (2004). The UCON\_ABC usage control model. *ACM Transactions on Information and System Security*, 7(1).
- Saltzer, J. H., & Schroeder, M. D. (1975). The protection of information in computer systems. *Proceedings of the IEEE*, 63(9).

Torres-Arias, S., Ammula, A. K., Curtmola, R., & Cappos, J. (2019). in-toto: Providing farm-to-table guarantees for bits and bytes. USENIX Security 2019.

Wang, Q., & Li, N. (2010). Satisfiability and resiliency in workflow authorization systems. ACM Transactions on Information and System Security, 13(4).

## Appendix A: Implementation Artifacts

- TypeScript SDK: npm install agent-passport-system (v1.13.2, 534 tests)
- Python SDK: pip install agent-passport-system (v0.4.0, 86 tests)
- MCP Server: npm install agent-passport-system-mcp (v2.8.5, 61 tools)
- Remote MCP: <https://mcp.aeoess.com/sse>
- Source: [github.com/aeoess/agent-passport-system](https://github.com/aeoess/agent-passport-system)
- Paper: [doi.org/10.5281/zenodo.18749779](https://doi.org/10.5281/zenodo.18749779)
- Protocol documentation: [aeoess.com/llms-full.txt](https://aeoess.com/llms-full.txt)

## Appendix B: Invariant-to-Test Mapping

Invariant	Test File(s)	Adversarial Scenario
INV-1: Identity Unforgeability	passport.test.ts, adversarial.ts, adversarial-paper.test.ts	S1
INV-2: Scope Narrowing	delegation.test.ts, adversarial.ts, property-delegation.test.ts	S2
INV-3: Spend Limit Narrowing	delegation.test.ts, commerce.test.ts, property-delegation.test.ts	S2 (implicit)
INV-4: Cascade Completeness	cascade.test.ts, property-delegation.test.ts	S5
INV-5: Revocation Irreversibility	cascade.test.ts, adversarial-paper.test.ts	S5
INV-6: Three-Signature	policy.test.ts, adversarial-paper.test.ts	S4
INV-7: Floor Narrowing	values.test.ts, enforcement.test.ts	—
INV-8: Human Root	attribution.test.ts, delegation.test.ts, adversarial-paper.test.ts	S3