

Internet Engineering Task Force  
Internet-Draft  
Intended status: Informational  
Expires: October 10, 2026

J. Michalak  
Nivalto, Inc.  
April 10, 2026

Agent Route Origin Authorization (AgentROA):  
A Cryptographic Policy Enforcement Framework for AI Agent Actions  
over the Model Context Protocol (MCP)

draft-nivalto-agentroa-route-authorization-00

## Abstract

This document specifies the Agent Route Origin Authorization (AgentROA) framework, a cryptographic policy enforcement model for governing the actions of autonomous AI agents operating over the Model Context Protocol (MCP) and Agent-to-Agent (A2A) protocols. AgentROA introduces three core protocol objects: the Agent Route Origin Authorization (ROA) envelope, the Agent Route Attestation (ARA) per-hop receipt, and the Agent Execution Receipt (AER). Together these objects enable: (1) cryptographic binding of an agent's authorized action scope to a signed policy envelope at session initialization, (2) per-hop attestation across multi-agent delegation chains with monotonic scope-narrowing semantics (no policy envelope may be expanded by a downstream delegation), and (3) cryptographic receipts produced intrinsically by the enforcement decision at each MCP tool call boundary. The framework is modeled on the BGP Route Origin Authorization (ROA) concept from RPKI (RFC 6480) applied to the AI agent execution domain.

The Border Gateway enforcement model positions a cryptographic enforcement proxy at the MCP protocol boundary — external to the agent's execution context — reducing the risk that governance decisions are influenced by the governed agent by placing enforcement in a separate process boundary. This document establishes the architectural model, protocol object schemas, and enforcement semantics for the AgentROA framework.

## Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at <https://datatracker.ietf.org/drafts/current/>.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on October 10, 2026.

## Copyright Notice

Copyright (c) 2026 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<https://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document.

## Table of Contents

1. Introduction . . . . .	3
1.1. Motivation . . . . .	3
1.2. BGP RPKI Analogy . . . . .	4
1.3. Scope . . . . .	5
2. Terminology . . . . .	5
3. Architecture Overview . . . . .	7
3.1. System Components . . . . .	7
3.2. Trust Boundary Model . . . . .	8
3.3. Enforcement Proxy Architecture . . . . .	9
4. Agent Route Origin Authorization (ROA) Envelope . . . . .	10
4.1. ROA Envelope Structure . . . . .	10
4.2. Policy Digest Binding . . . . .	12
4.3. Monotonic Scope-Narrowing Semantics . . . . .	13
4.4. ROA Envelope Signing . . . . .	14
5. Agent Route Attestation (ARA) Per-Hop Protocol . . . . .	15
5.1. ARA Object Structure . . . . .	15
5.2. Delegation Chain Construction . . . . .	17
5.3. Chain Verification Algorithm . . . . .	18
6. Agent Execution Receipt (AER) . . . . .	20
6.1. AER Structure . . . . .	20
6.2. Receipt Generation at Enforcement Boundary . . . . .	22
6.3. SCITT Transparency Log Integration . . . . .	23
7. Border Gateway Enforcement Model . . . . .	24
7.1. Enforcement Proxy Position . . . . .	24
7.2. MCP Protocol Integration . . . . .	25
7.3. A2A Protocol Integration . . . . .	27
7.4. Enforcement Decision Algorithm . . . . .	28
8. Security Considerations . . . . .	29
8.1. Threat Model . . . . .	29
8.2. Key Management . . . . .	31
8.3. Implementation Considerations . . . . .	32
8.4. Revocation and Degraded Operation . . . . .	32
8.4.1. Envelope Expiry and Revocation . . . . .	32
8.4.2. Replay Prevention Cache . . . . .	33
8.4.3. Degraded Operation . . . . .	33
9. IANA Considerations . . . . .	34
10. References . . . . .	33
10.1. Normative References . . . . .	33
10.2. Informative References . . . . .	34
Appendix A. JSON Schema Definitions . . . . .	38
Appendix B. Protocol Object Examples . . . . .	44
Author's Address . . . . .	50

## 1. Introduction

### 1.1. Motivation

Autonomous AI agents are increasingly being deployed in enterprise and regulated environments. These systems can investigate incidents across cloud infrastructure, invoke external tools, initiate transactions, and delegate tasks to downstream agents without human review at each step.

The Model Context Protocol (MCP) provides a common mechanism for connecting agents to external tools, data sources, and APIs. The Agent-to-Agent (A2A) protocol provides complementary support for inter-agent communication and task delegation.

These protocols address connectivity and interoperability. They do not, by themselves, define a cryptographic mechanism that binds an agent's authorized action scope to protocol execution, enforces that scope at the protocol boundary, or produces receipts that can be independently verified after the fact.

Existing governance approaches often rely on application-layer monitoring, behavioral detection, or post-execution audit logging. In deployments where such governance mechanisms are co-located with the governed agent, a compromised, manipulated, or misconfigured agent may influence the governance outcome.

AgentROA addresses this concern by positioning enforcement at the protocol boundary, external to the agent's execution context. The Border Gateway enforcement proxy intercepts MCP tool call requests before they reach the target MCP server, validates the caller's ROA envelope, enforces monotonic scope-narrowing semantics across delegation chains, and produces a cryptographic AER receipt as an intrinsic output of the enforcement decision.

The governing question AgentROA answers is: was this agent authorized to perform this specific action under this specific policy envelope at this specific moment — and can that be shown to a third party, including a regulator, auditor, or counter-party, without relying on the agent's own account of execution?

## 1.2. BGP RPKI Analogy

AgentROA draws an analogy from the BGP Route Origin Authorization framework specified in RFC 6480 (RPKI) and RFC 6811 (BGP Prefix Origin Validation).

In BGP RPKI, a Route Origin Authorization (ROA) is a cryptographically signed object that states which Autonomous System (AS) is authorized to originate routes for a specific IP address prefix. Border routers validate incoming route announcements against the ROA database before accepting them into the routing table. This prevents route hijacking by ensuring that only authorized ASes can announce specific prefixes.

AgentROA applies the same model to AI agent execution:

- o An Agent ROA envelope states which agent identity is authorized to invoke specific MCP capabilities under specific policy constraints, signed by the authorizing entity.
- o The Border Gateway validates incoming MCP tool calls against the ROA envelope before forwarding them to the target MCP server. This prevents capability hijacking by ensuring that agents can only invoke capabilities they are explicitly authorized for.
- o The Agent Route Attestation (ARA) provides per-hop attestation across multi-agent delegation chains, analogous to BGP path validation, ensuring that each hop in a delegation chain operated within its declared scope.

The key structural insight from RPKI that AgentROA preserves is the separation of the authorization infrastructure (the ROA envelope and its signing) from the enforcement infrastructure (the Border Gateway). This separation ensures that the enforcement mechanism does not depend on trusting the party being governed.

### 1.3. Scope

This document specifies:

- o The AgentROA framework architecture and trust boundary model.
- o The protocol object schemas for ROA envelopes, ARA per-hop attestations, and AER execution receipts.
- o The monotonic scope-narrowing semantics for multi-agent delegation chains.
- o The Border Gateway enforcement proxy model for MCP and A2A.

Future extensions, including voice-triggered actions and agent identity addressing, are outside the scope of this document.

This document does not specify a wire protocol for MCP or A2A, which are governed by their respective specifications. AgentROA operates as an enforcement layer above the MCP OAuth 2.0 authorization mechanism specified in the MCP Authorization specification.

## 2. Terminology

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in BCP 14 [RFC2119] [RFC8174].

#### Agent:

An autonomous software system that perceives its environment, makes decisions, and takes actions — including invoking tools via MCP and delegating tasks to other agents via A2A — without constant human oversight at each step.

#### Agent Route Origin Authorization (ROA) Envelope:

A cryptographically signed data structure that specifies the authorized action scope for a specific agent session, including the set of permitted MCP capabilities, the policy digest binding, the session identity, the device attestation reference, and the approval state. Analogous to a BGP ROA for IP prefix origin authorization.

#### Agent Route Attestation (ARA):

A cryptographically signed per-hop attestation produced at each delegation boundary in a multi-agent execution chain. Each ARA object references the upstream ROA or ARA, the downstream agent's declared scope, and certifies that the downstream scope is a proper subset of the upstream scope (monotonic scope-narrowing).

#### Agent Execution Receipt (AER):

A cryptographically signed artifact produced by the Border Gateway at the moment an MCP tool call enforcement decision is made. The AER is produced intrinsically by the enforcement decision, not generated by a separate logging system after execution. The AER includes the policy digest active at decision time, the capability identifier, the session identity, and the enforcement outcome.

#### Border Gateway:

An enforcement proxy that sits at the MCP protocol boundary between an agent and one or more MCP servers. The Border Gateway is external to the agent's execution context, operating in a

separate process with its own trust domain. The Border Gateway validates ROA envelopes, enforces monotonic scope-narrowing semantics, and produces AER receipts.

**Capability Identifier:**

A structured identifier for a specific MCP tool invocation, consisting of the MCP server identifier and the tool name, formatted as `mcp:<server-id>.<tool-name>`.

**Delegation Chain:**

An ordered sequence of ARA objects representing the authorization path from the root ROA envelope to the current executing agent. Each element in the chain represents one agent-to-agent delegation hop.

**Monotonic Scope-Narrowing:**

The property of a delegation chain where each downstream agent's authorized scope is a proper subset of — or equal to — its upstream delegator's scope. The scope **MUST NOT** be expanded at any delegation boundary. This is the "no-loosen" invariant.

**No-Loosen Invariant:**

The fundamental enforcement rule of AgentROA: a downstream agent in a delegation chain **MAY NOT** be authorized for capabilities or policy scopes broader than those of its upstream delegator. Formally:  $\text{scope}(\text{ARA}[n]) \subseteq \text{scope}(\text{ARA}[n-1])$  for all  $n$ .

**Policy Digest:**

A cryptographic hash of the policy document governing an agent's authorized behavior at a specific version. The policy digest binds the ROA envelope to the specific policy in effect at the time of authorization.

**Session Identity:**

A cryptographically bound identifier for a specific agent execution session, including the agent's principal identity, the device or infrastructure attestation, and the session inception timestamp.

**SCITT:**

Supply Chain Integrity, Transparency and Trust [SCITT-ARCH]. AgentROA receipts are designed to be compatible with SCITT transparency logs.

### 3. Architecture Overview

#### 3.1. System Components

The AgentROA framework consists of the following components:

a) Agent Identity Registry

A registry mapping agent identifiers to cryptographic public keys and capability declarations. Agents register their identity and declared capability scope prior to session initiation.

b) Policy Engine

A system that evaluates agent action requests against policy documents and produces ROA envelopes at session initiation. The policy engine is operated by the enterprise or regulated entity deploying the agent.

c) Border Gateway

The enforcement proxy described in Section 7. The Border Gateway is the critical enforcement point. It MUST be deployed external to the agent's execution context.

e) SCITT Transparency Log (optional)

An append-only transparency log compatible with [SCITT-ARCH] that records AER receipts. When deployed, the SCITT log provides independent third-party verifiability of the agent's execution history.

### 3.2. Trust Boundary Model

The fundamental trust boundary in AgentROA separates two domains:

The Agent Execution Domain:

The process or container in which the AI agent framework executes. This domain includes the agent's model inference, tool orchestration, and application-layer code. This domain is considered untrusted from the perspective of the enforcement infrastructure.

The Enforcement Domain:

The process or container in which the Border Gateway executes. This domain is separate from and external to the Agent Execution Domain. The Enforcement Domain holds the signing keys for AER receipts and has sole authority to authorize MCP tool calls.

This separation is the critical architectural distinction from application-layer governance approaches in which the policy engine and the agents it governs run in the same process. When a governance layer runs in the same process as the agent, a compromised agent may influence the governance outcome.

AgentROA assumes that the Border Gateway enforcement proxy runs in a separate process from the governed agent. Deployment MAY use sidecar containers in a Kubernetes pod, separate virtual machines, or hardware-isolated enclaves. The architectural model in this document does not assume a policy decision point that is co-located within the agent's execution context.

### 3.3. Enforcement Proxy Architecture

The Border Gateway implements the following request handling model for MCP tool calls:

Step 1: Intercept

The Border Gateway receives the MCP tool call request from the agent. The agent MUST present its current ROA envelope or a valid ARA chain in an authorization header or equivalent protocol metadata field.

Step 2: Validate Envelope

The Border Gateway validates:

- a) The ROA envelope signature (EdDSA verification against the agent's registered public key in the Agent Identity Registry).
- b) The policy digest (the hash of the policy document in the envelope MUST match the current policy version for this agent's scope).
- c) The session identity (the session MUST be active and not expired).
- d) The device attestation reference (if required by policy).

Step 3: Validate Scope

The Border Gateway validates that the requested capability (mcp:<server-id>.<tool-name>) is within the authorized scope declared in the ROA envelope or ARA chain.

#### Step 4: Apply No-Loosen Check

If the request originates from a delegation chain (ARA present), the Border Gateway validates the monotonic scope-narrowing invariant:  $\text{scope}(\text{current-ARA}) \subseteq \text{scope}(\text{parent-ARA-or-ROA})$ . A delegation chain that attempts to expand scope MUST be rejected.

#### Step 5: Produce AER

The Border Gateway produces an Agent Execution Receipt (AER) as an intrinsic output of the enforcement decision. The AER is signed by the Border Gateway's private key. The AER is produced regardless of whether the enforcement outcome is PERMIT or DENY.

#### Step 6: Forward or Reject

If the outcome is PERMIT, the Border Gateway forwards the MCP tool call to the target MCP server. If the outcome is DENY, the Border Gateway returns an authorization error to the agent and records the denial event.

#### Step 7: Write to Ledger

The AER is written to the local audit store and, if configured, to the SCITT transparency log.

## 4. Agent Route Origin Authorization (ROA) Envelope

### 4.1. ROA Envelope Structure

The AgentROA ROA Envelope is a JSON object with the following top-level fields. All fields are REQUIRED unless noted.

schema\_version:

String. MUST be "1.0".

envelope\_id:

String. A unique identifier for this envelope instance, formatted as "env:<random-hex-16>".

issued\_at:

String. ISO 8601 datetime at which this envelope was issued.

expires\_at:

String. ISO 8601 datetime at which this envelope expires. Enforcement decisions MUST fail for expired envelopes.

session:

Object. Session context.

- session\_id: String. Unique session identifier.
- channel: String. One of: "api", "mcp\_client", "voice", "browser", "mobile\_app".
- agent\_id: String. The agent identifier of the governed agent.
- device\_attestation\_ref: String, OPTIONAL. Reference to a device attestation artifact (e.g., WebAuthn assertion).

authorized\_scope:

Object. The authorized capability scope.

- capabilities: Array of Strings. Each element is a capability identifier of the form mcp:<server-id>.<tool-name> or the wildcard form mcp:<server-id>.\*.
- max\_delegation\_depth: Integer. Maximum number of delegation hops permitted. MUST be  $\geq 0$ .

- cross\_org\_permitted: Boolean. Whether delegation across organizational boundaries is permitted.
- data\_classification\_ceiling: String, OPTIONAL. Maximum data classification level accessible.
- budget\_ceiling: Number, OPTIONAL. Maximum total spend or resource budget permitted for this session, in the unit declared by budget\_unit. Downstream delegations MUST NOT declare a budget\_ceiling exceeding this value.
- budget\_unit: String, OPTIONAL. Unit for budget\_ceiling (e.g., "USD", "tokens", "api\_calls"). Required when budget\_ceiling is present.
- price\_class: Integer, OPTIONAL. Maximum price tier permitted (lower value = lower cost). Downstream delegations MUST NOT declare a price\_class exceeding this value.
- slo\_class: Integer, OPTIONAL. Minimum required service level (higher value = stricter SLO). Downstream delegations MUST NOT declare an slo\_class below this value.

#### policy:

- Object. Policy binding.
- policy\_id: String. Identifier of the governing policy.
  - policy\_version: String. Version of the governing policy.
  - policy\_digest: String. SHA-256 hash of the policy document, formatted as "sha256:<hex>".
  - policy\_uri: String, OPTIONAL. URI at which the policy document may be retrieved.

#### authorization:

- Object. Authorization requirements.
- auth\_strength: String. One of: "session\_only", "device\_bound", "device\_bound\_with\_attestation", "dual\_control".
  - approval\_state: String. One of: "pending", "granted", "not\_required".
  - approval\_artifact\_ref: String, OPTIONAL. Reference to the signed approval artifact.

#### evidence:

- Object. Evidence references.
- session\_hash: String. Hash of the session establishment event.
  - model\_provenance: Array of Strings. Identifiers of the AI models that may execute within this session.

#### signatures:

- Array of Objects. One or more signatures.
- signer: String. Identifier of the signing entity.
  - alg: String. Signature algorithm. MUST be "EdDSA".
  - sig: String. Base64url-encoded signature over the canonical JSON serialization of all fields except "signatures".

## 4.2. Policy Digest Binding

The policy\_digest field binds the ROA envelope to a specific version of the governing policy document. This binding serves two purposes:

- Enforcement time validation:** The Border Gateway MUST verify that the policy\_digest in the ROA envelope matches the hash of the current policy document for this agent's scope. If the policy has been updated since the envelope was issued, the envelope MUST be rejected and a new envelope issued against the current policy.
- Receipt auditability:** The AER receipt includes the policy\_digest



from the envelope at the time of the enforcement decision. This allows an auditor reviewing the receipt to retrieve the specific policy version that governed the action, even if the policy has subsequently changed.

The canonical policy digest is computed as:

```
policy_digest = "sha256:" || hex(SHA-256(policy_document_bytes))
```

where `policy_document_bytes` is the UTF-8 encoding of the policy document in its canonical JSON serialization.

#### 4.3. Monotonic Scope-Narrowing Semantics

This section specifies the no-loosen invariant, which is the foundational enforcement property of AgentROA.

DEFINITION (Monotonic Scope-Narrowing):

For any two consecutive elements `ARA[n-1]` and `ARA[n]` in a delegation chain, the full policy envelope of `ARA[n]` MUST be no broader than the policy envelope of `ARA[n-1]` across all constrained dimensions.

AgentROA applies a tighten-only partial-order algebra across three distinct constraint dimensions. Each dimension has its own ordering operator, and a violation of ANY dimension MUST cause the Border Gateway to reject the delegation with a DENY AER receipt.

Dimension 1 — Capability Scope (set containment):

Let  $C(x)$  denote the set of capabilities authorized by envelope or attestation  $x$ . Then for all  $n > 0$ :

$$C(ARA[n]) \subseteq C(ARA[n-1])$$

where `ARA[0]` is the root ROA envelope.

An ARA that claims capabilities outside the authorized set of its parent violates this dimension. The denial reason code MUST be `SCOPE_EXPANSION_DENIED`.

Dimension 2 — Budget and Price Class (less-than-or-equal):

Let  $B(x)$  denote the `budget_ceiling` declared in envelope  $x$  and  $P(x)$  denote the `price_class`. Then for all  $n > 0$ :

$$\begin{aligned} B(ARA[n]) & \leq B(ARA[n-1]) \\ P(ARA[n]) & \leq P(ARA[n-1]) \end{aligned}$$

A downstream delegation that raises the budget ceiling or escalates the price class above the parent's declared maximum violates this dimension. The denial reason code MUST be `BUDGET_EXPANSION_DENIED`.

Dimension 3 — Service Level Objective (greater-than-or-equal):

Let  $S(x)$  denote the `slo_class` declared in envelope  $x$ , where higher numerical values denote stricter service levels. Then for all  $n > 0$ :

$$S(ARA[n]) \geq S(ARA[n-1])$$

A downstream delegation that relaxes the required service level below the parent's declared floor violates this dimension. The denial reason code MUST be `SLO_RELAXATION_DENIED`.

Combined invariant:

The Border Gateway MUST evaluate all three dimensions at each delegation boundary. A delegation envelope that satisfies Dimension 1 but violates Dimension 2 or 3 MUST be rejected. Partial conformance is not sufficient.

#### IMPLEMENTATION REQUIREMENT:

The Border Gateway MUST evaluate this invariant at each delegation boundary. An ARA that violates any dimension of the tighten-only partial-order algebra MUST be rejected. The rejection MUST produce a DENY AER receipt containing the applicable reason code.

This invariant prevents privilege escalation through agent delegation chains. An agent that delegates a task to a downstream agent CANNOT grant the downstream agent permissions, budget, or relaxed service levels that the delegating agent does not itself possess.

Scope comparison algorithm:

The Border Gateway compares capability sets using the following procedure:

1. Resolve each capability identifier to its canonical form. Wildcard capabilities (`mcp:<server-id>.*`) are expanded to include all capabilities advertised by the named MCP server's capability manifest.
2. For each capability in the downstream ARA, verify that the capability is present in or subsumed by a wildcard in the parent's capability set.
3. If any downstream capability is not subsumed, reject the delegation with a `SCOPE_EXPANSION_DENIED` error.
4. If the downstream ARA declares a `budget_ceiling`, verify that:  
`budget_ceiling(ARA[n]) budget_ceiling(ARA[n-1])`  
If violated, reject with `BUDGET_EXPANSION_DENIED`.
5. If the downstream ARA declares a `price_class`, verify that:  
`price_class(ARA[n]) price_class(ARA[n-1])`  
If violated, reject with `BUDGET_EXPANSION_DENIED`.
6. If the downstream ARA declares an `slo_class`, verify that:  
`slo_class(ARA[n]) slo_class(ARA[n-1])`  
If violated, reject with `SLO_RELAXATION_DENIED`.

#### 4.4. ROA Envelope Signing

ROA envelopes MUST be signed using EdDSA with the Ed25519 curve as specified in [RFC8032]. The signature is computed over the canonical JSON serialization of the envelope excluding the "signatures" field.

Canonical JSON serialization follows the JSON Canonicalization Scheme (JCS) as specified in [RFC8785].

The signing key MUST correspond to the public key registered in the Agent Identity Registry for the authorizing entity. The key identifier MUST be included in the "signer" field of the signature object.

#### 5. Agent Route Attestation (ARA) Per-Hop Protocol

## 5.1. ARA Object Structure

An Agent Route Attestation (ARA) object is produced at each agent-to-agent delegation boundary. The ARA attests that:

- a) The delegating agent (the upstream agent) authorized the downstream agent to act on its behalf.
- b) The downstream agent's authorized scope is within the delegating agent's scope (monotonic scope-narrowing).
- c) The delegation occurred within a valid session.

The ARA object has the following structure:

schema\_version:

String. MUST be "1.0".

ara\_id:

String. Unique identifier for this ARA, formatted as "ara:<random-hex-16>".

issued\_at:

String. ISO 8601 datetime at which this ARA was issued.

upstream\_ref:

Object. Reference to the parent in the delegation chain.

- ref\_type: String. One of: "roa\_envelope", "ara".
- ref\_id: String. The envelope\_id or ara\_id of the parent.
- ref\_digest: String. SHA-256 hash of the parent object's canonical serialization. This creates a hash chain over the delegation path.

delegating\_agent:

Object. Identity of the delegating agent.

- agent\_id: String. agent identifier of the delegating agent.
- session\_id: String. Session identifier of the delegating agent's current session.

delegated\_agent:

Object. Identity of the agent receiving the delegation.

- agent\_id: String. agent identifier of the delegated agent.
- capability\_declaration\_ref: String, OPTIONAL. Reference to the delegated agent's declared capability manifest.

delegated\_scope:

Object. The scope being delegated. MUST satisfy the monotonic scope-narrowing invariant with respect to the parent scope.

- capabilities: Array of Strings. Capability identifiers authorized for the delegated agent.
- task\_context: String, OPTIONAL. Plain-language description of the task being delegated.
- max\_delegation\_depth: Integer. Remaining permitted delegation depth. MUST be strictly less than the parent's max\_delegation\_depth.

policy:

Object. Policy binding for this delegation.

- policy\_digest: String. MUST match the policy\_digest of the root ROA envelope in the delegation chain.
- policy\_version: String.

signatures:

Array of Objects. Signatures as specified in Section 4.4.

The ARA MUST be signed by the delegating agent.

## 5.2. Delegation Chain Construction

A delegation chain is constructed as follows:

1. The root element is the ROA envelope issued by the Policy Engine at session initiation (ref\_type: "roa\_envelope").
2. Each subsequent element is an ARA object produced at a delegation boundary (ref\_type: "ara").
3. The chain forms a singly-linked list through the upstream\_ref fields. The ref\_digest field creates a cryptographic hash chain: each ARA commits to the exact bytes of its parent.
4. The chain terminates at the agent currently requesting a MCP tool call.

When a Border Gateway receives a MCP tool call, the requesting agent presents its delegation chain (an ordered array of serialized ARA objects, from root to current). The Border Gateway validates the entire chain before making an enforcement decision.

Chain length:

The chain length MUST NOT exceed the max\_delegation\_depth specified in the root ROA envelope.

Chain integrity:

The Border Gateway MUST verify:

- a) The root ROA envelope signature.
- b) Each ARA signature in the chain.
- c) The hash chain integrity (ref\_digest field).
- d) The monotonic scope-narrowing invariant at each hop.

## 5.3. Chain Verification Algorithm

The following algorithm MUST be implemented by the Border Gateway:

FUNCTION VerifyChain(chain, requested\_capability):

INPUT:

chain: ordered array [root\_envelope, ara\_1, ..., ara\_n]  
requested\_capability: capability identifier being requested

OUTPUT:

PERMIT or DENY, with reason

STEP 1: Verify root envelope

```
IF NOT VerifySignature(chain[0]) THEN
    RETURN DENY, "invalid_root_signature"
IF chain[0].expires_at < NOW() THEN
    RETURN DENY, "envelope_expired"
current_scope = chain[0].authorized_scope.capabilities
current_budget = chain[0].authorized_scope.budget_ceiling
current_slo = chain[0].authorized_scope.slo_class
```

STEP 2: Verify hash chain and monotonic narrowing

```
FOR i = 1 TO length(chain) - 1:
    ara = chain[i]
    parent = chain[i-1]
    parent_digest = SHA-256(canonical_json(parent))

    IF ara.upstream_ref.ref_digest != parent_digest THEN
        RETURN DENY, "chain_integrity_violation"
```

```

IF NOT VerifySignature(ara) THEN
    RETURN DENY, "invalid_ara_signature_at_hop_" + i

IF NOT Subset(ara.delegated_scope.capabilities, current_scope) THEN
    RETURN DENY, "scope_expansion_violation_at_hop_" + i

IF ara.delegated_scope.budget_ceiling IS PRESENT AND
    current_budget IS PRESENT AND
    ara.delegated_scope.budget_ceiling > current_budget THEN
    RETURN DENY, "budget_expansion_denied_at_hop_" + i

IF ara.delegated_scope.slo_class IS PRESENT AND
    current_slo IS PRESENT AND
    ara.delegated_scope.slo_class < current_slo THEN
    RETURN DENY, "slo_relaxation_denied_at_hop_" + i

current_scope = ara.delegated_scope.capabilities
current_budget = ara.delegated_scope.budget_ceiling
current_slo = ara.delegated_scope.slo_class

STEP 3: Verify requested capability is in current scope
IF NOT In(requested_capability, current_scope) THEN
    RETURN DENY, "capability_not_in_scope"

STEP 4: Verify policy digest consistency
FOR i = 1 TO length(chain) - 1:
    IF chain[i].policy.policy_digest != chain[0].policy.policy_digest THEN
        RETURN DENY, "policy_digest_mismatch_at_hop_" + i

RETURN PERMIT

END FUNCTION

```

## 6. Agent Execution Receipt (AER)

### 6.1. AER Structure

The Agent Execution Receipt (AER) is produced by the Border Gateway as an intrinsic output of each enforcement decision. The AER is a cryptographically signed artifact intended to provide independently verifiable evidence that a specific enforcement decision was made for a specific agent action under a specific policy at a specific time.

The AER is not a log entry. It is a signed commitment produced by the enforcement layer at the moment of the decision. The distinction is architecturally significant: a log entry is produced after the fact by a logging system that may or may not have observed the exact enforcement logic. An AER is produced by the enforcement logic itself, committed before the action executes (for PERMIT) or when the action is rejected (for DENY).

AER structure:

schema\_version:  
String. MUST be "1.0".

aer\_id:  
String. Unique receipt identifier, formatted as  
"aer:<random-hex-16>".

produced\_at:  
String. ISO 8601 datetime at which this receipt was produced.  
This MUST be the timestamp of the enforcement decision, not

the timestamp of execution completion.

enforcement\_outcome:

String. One of: "permit", "deny".

enforcement\_mode:

String. One of: "normal", "degraded". "degraded" indicates that the enforcement decision was made against locally cached materials because the Agent Identity Registry or revocation infrastructure was unavailable. Relying parties that require non-degraded receipts SHOULD reject receipts with enforcement\_mode "degraded". See Section 9.4.3.

denial\_reason:

String. Present only when enforcement\_outcome is "deny". One of: "invalid\_signature", "envelope\_expired", "envelope\_revoked", "replay\_detected", "chain\_integrity\_violation", "scope\_expansion\_violation", "budget\_expansion\_denied", "slo\_relaxation\_denied", "capability\_not\_in\_scope", "policy\_digest\_mismatch", "approval\_required", "auth\_strength\_insufficient".

session:

Object. Session context.

- session\_id: String.
- agent\_id: String. agent identifier of the requesting agent.
- device\_attestation\_ref: String, OPTIONAL.

action:

Object. The action subject to enforcement.

- capability: String. The capability identifier requested.
- mcp\_server\_id: String. Target MCP server identifier.
- mcp\_tool\_name: String. Target tool name.
- input\_hash: String. SHA-256 hash of the canonical serialization of the tool call inputs. The inputs themselves are NOT included in the AER.

policy:

Object. Policy binding at enforcement time.

- policy\_id: String.
- policy\_digest: String. The policy digest from the ROA envelope. This records the exact policy version that governed this enforcement decision.

chain\_summary:

Object. Summary of the delegation chain.

- chain\_depth: Integer. Number of hops in the chain.
- root\_envelope\_id: String. The envelope\_id of the root ROA envelope.
- chain\_digest: String. SHA-256 hash of the canonical serialization of the complete delegation chain.

border\_gateway:

Object. Border Gateway identity.

- gateway\_id: String. Identifier of the producing Border Gateway instance.
- gateway\_version: String. Software version.

plan\_hash:

String, OPTIONAL. A cryptographic hash of the frozen artifacts that governed this enforcement decision, enabling deterministic or tolerance-bounded replay. The plan\_hash binds at minimum: the prompt template identifier and version, the model identifier and version, the operator and governance library versions, and the policy-pack identifiers active at enforcement time. When present, it is formatted as "sha256:<hex>" and computed over

the canonical JSON serialization of the frozen artifact manifest. Auditors or compliance systems MAY use `plan_hash` to re-execute the enforcement decision logic against the same frozen inputs and verify that the original allow/deny outcome is reproduced within declared tolerances. Implementations that support deterministic replay SHOULD populate this field.

signatures:

Array of Objects. The AER MUST be signed by the Border Gateway's private key using EdDSA/Ed25519. The Border Gateway signing key MUST be registered in the Agent Identity Registry.

## 6.2. Receipt Generation at Enforcement Boundary

The following invariants apply to AER generation:

Invariant 1 (Pre-execution commitment):

For enforcement\_outcome "permit", the AER MUST be produced and persisted to the local audit store BEFORE the MCP tool call is forwarded to the target MCP server. This ensures that the AER exists regardless of whether the tool call subsequently succeeds or fails.

Invariant 2 (Denial receipt):

For enforcement\_outcome "deny", the AER MUST be produced and persisted before returning the authorization error to the requesting agent.

Invariant 3 (Policy digest capture):

The policy\_digest in the AER MUST be captured from the ROA envelope at enforcement time. If the policy has been updated between envelope issuance and enforcement time, the enforcement decision MUST fail with denial reason "policy\_digest\_mismatch".

Invariant 4 (Input hash binding):

The input\_hash field MUST be computed from the actual tool call inputs presented at enforcement time. This ensures that the AER receipt is bound to the specific inputs, not merely to the capability identifier.

## 6.3. SCITT Transparency Log Integration

AgentROA AER receipts are designed to be submitted to SCITT-compatible transparency logs [SCITT-ARCH]. When a SCITT log is configured:

- a) The Border Gateway submits each AER as a SCITT Statement.
- b) The SCITT log returns a SCITT Receipt (a countersignature over the log entry).
- c) The SCITT Receipt is appended to the AER's signatures array.
- d) The combined AER (with SCITT Receipt) is returned to the requesting agent and stored in the local audit store.

SCITT integration provides three additional properties:

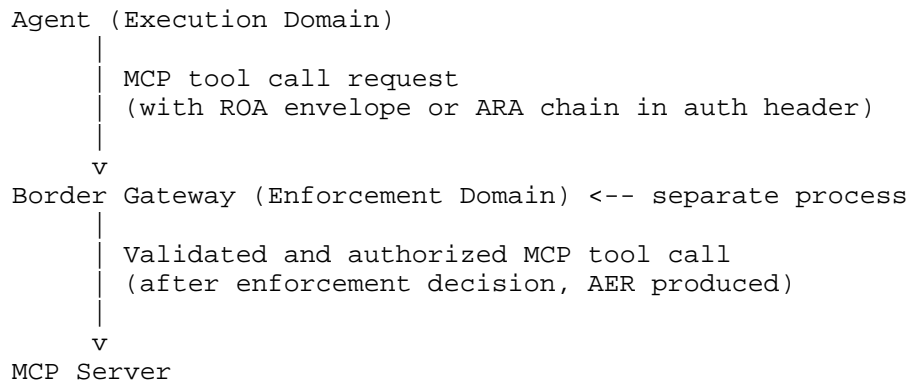
1. Third-party verifiability: Any party with the SCITT log's public key can verify the AER without trusting the Border Gateway operator.
2. Tamper evidence: The append-only SCITT log structure prevents modification of historical receipts.

3. Transparency: Deployments MAY use transparency logs to support independent verification or audit requirements.

## 7. Border Gateway Enforcement Model

### 7.1. Enforcement Proxy Position

The Border Gateway MUST be positioned as follows:



The Border Gateway acts as a reverse proxy for MCP connections. From the MCP server's perspective, all requests arrive from the Border Gateway. From the agent's perspective, the Border Gateway is the target endpoint for MCP tool calls.

This architecture ensures:

- a) The MCP server never receives unauthorized tool calls.
- b) The enforcement decision is made by infrastructure the agent cannot influence.
- c) Every tool call — authorized or denied — produces a signed receipt in the enforcement domain.

### 7.2. MCP Protocol Integration

MCP uses OAuth 2.0 for authorization as specified in the MCP Authorization specification. AgentROA operates as an additional layer above MCP's OAuth authorization mechanism.

AgentROA DOES NOT replace MCP's OAuth authorization. Both mechanisms operate in sequence:

Step 1 (MCP OAuth): The agent obtains an OAuth access token scoped to the target MCP server. This token is validated by the MCP server's authorization server.

Step 2 (AgentROA envelope): The agent presents its ROA envelope or ARA chain to the Border Gateway in an authorization header or equivalent protocol metadata field.

Step 3 (Border Gateway validation): The Border Gateway performs the chain verification algorithm specified in Section 5.3 and produces an AER receipt.

Step 4 (Forwarding): If the enforcement outcome is PERMIT, the Border Gateway forwards the tool call to the MCP server with the OAuth token. The AER receipt identifier or receipt object is returned to the agent in response metadata.

OAuth token audience validation:

The Border Gateway MUST validate that the OAuth token's audience claim (aud) matches the target MCP server's



registered identifier. This prevents token reuse across MCP servers.

#### MCP server registration:

MCP servers that participate in AgentROA enforcement MUST register their capability manifests with the Agent Identity Registry. The capability manifest declares all tools the server exposes, used by the Border Gateway for wildcard scope resolution.

### 7.3. A2A Protocol Integration

When agents communicate using the A2A protocol, AgentROA governs the delegation of tasks between agents.

#### Task delegation over A2A:

When an orchestrating agent delegates a task to a downstream agent over A2A, the orchestrating agent MUST produce an ARA object for the delegation (see Section 5) and include it in the A2A task request.

#### Cross-organization delegation:

When a delegation crosses an organizational boundary, the ROA envelope's `cross_org_permitted` field MUST be true. Additional cross-organization agreement semantics are outside the scope of this document.

### 7.4. Enforcement Decision Algorithm

The complete enforcement decision algorithm is:

FUNCTION Enforce(request):

INPUT:

request: MCP tool call request with auth headers

OUTPUT:

PERMIT or DENY, with AER receipt

STEP 1: Extract credentials

envelope\_or\_chain = ExtractFromHeader(request)

STEP 2: Determine chain type

IF envelope\_or\_chain is ROA envelope:

chain = [envelope\_or\_chain]

ELSE:

chain = envelope\_or\_chain (ARA chain)

STEP 3: Resolve capability

capability = mcp:<request.server\_id>.<request.tool\_name>

STEP 4: Execute chain verification

result = VerifyChain(chain, capability)

STEP 5: Check approval state if required

IF chain[0].authorization.auth\_strength IN

["device\_bound", "device\_bound\_with\_attestation"]:

IF chain[0].authorization.approval\_state != "granted":

result = DENY, "approval\_required"

STEP 6: Produce AER

aer = ProduceAER(  
session = ExtractSession(chain),  
action = ExtractAction(request),  
policy = ExtractPolicy(chain[0]),  
chain\_summary = SummarizeChain(chain),

```

        outcome = result.outcome,
        denial_reason = result.reason
    )
    SignAER(aer, border_gateway_private_key)
    PersistAER(aer)

```

STEP 7: Submit to SCITT (if configured)

```

    IF scitt_enabled:
        scitt_receipt = SubmitToSCITT(aer)
        AppendSignature(aer, scitt_receipt)

```

STEP 8: Return

```

    IF result.outcome == PERMIT:
        ForwardToMCPServer(request)
        RETURN response WITH receipt metadata referencing aer.aer_id
    ELSE:
        RETURN 403 Forbidden WITH receipt metadata referencing
            aer.aer_id

```

END FUNCTION

## 8. Security Considerations

### 8.1. Threat Model

AgentROA is designed to protect against the following threats:

T1 - Capability escalation through delegation:

A malicious or compromised agent attempts to grant a downstream agent capabilities it does not possess.

Mitigation: No-loosen invariant (Section 4.3).

T2 - Prompt injection leading to unauthorized tool calls:

An adversary embeds malicious instructions in data the agent processes, causing the agent to attempt tool calls outside its declared scope.

Mitigation: Border Gateway scope enforcement (Section 7). The agent cannot invoke capabilities outside its envelope regardless of what instructions it receives.

T3 - Replay attacks using captured envelopes:

An attacker captures a valid ROA envelope and replays it to authorize actions not intended by the original issuer.

Mitigation: Session identity binding and envelope expiry. The session\_id in the envelope MUST be unique per session, and the Border Gateway MUST maintain a replay prevention cache for recently-seen envelope identifiers.

T4 - Governance bypass through in-process compromise:

A compromised agent attempts to modify or disable the governance layer.

Mitigation: Trust boundary separation (Section 3.2). The Border Gateway runs in a separate process that the agent cannot access.

T5 - Policy drift — agent scope expands over time:

An agent's effective capability scope expands through learned skills or updated configuration without the policy envelope being reissued.

Mitigation: Policy digest binding (Section 4.2). The policy digest in the ROA envelope is validated against the current policy at each enforcement decision.

### 8.2. Key Management

Private keys used for signing ROA envelopes, ARA objects, and AER receipts MUST be stored in hardware security modules or equivalent secure key storage, or in device-bound key storage for device-bound approval flows.

Key rotation:

Implementations SHOULD support periodic rotation of ROA envelope signing keys and Border Gateway signing keys according to local deployment policy and risk requirements.

### 8.3. Implementation Considerations

Implementers SHOULD note that this framework requires the Border Gateway to process every MCP tool call. The Border Gateway is therefore a potential bottleneck and single point of failure. Implementations SHOULD provide:

- a) High availability deployment (multiple Border Gateway instances with consistent state).
- b) Low-latency enforcement appropriate to the deployment, noting that enforcement occurs at the protocol boundary rather than in application middleware.
- c) Audit store durability (AER receipts MUST survive Border Gateway failures).

### 8.4. Revocation and Degraded Operation

This section specifies requirements for ROA envelope revocation and for continued operation when the Border Gateway cannot reach the Agent Identity Registry or policy infrastructure.

#### 8.4.1. Envelope Expiry and Revocation

ROA envelopes include an `expires_at` field (Section 4.1). The Border Gateway MUST reject any envelope whose `expires_at` timestamp is in the past. The denial reason code MUST be `"envelope_expired"`.

In addition to time-based expiry, implementations SHOULD support explicit revocation. The Agent Identity Registry SHOULD publish revocation information for ROA envelopes whose corresponding sessions have been administratively terminated or whose signing keys have been compromised. Revocation information SHOULD be distributed as delta-encoded updates identified by a monotonically increasing epoch and sequence number, enabling validators to maintain current revocation state without full re-download.

When a validator processes a revocation delta, it MUST update its local revocation cache and MUST reject any subsequently presented envelope whose `envelope_id` or signing key appears in the revocation cache. The denial reason code for a revoked envelope MUST be `"envelope_revoked"`. AER receipts for revoked envelopes SHOULD include the revocation epoch and sequence number at which the revocation was applied.

#### 8.4.2. Replay Prevention Cache

To prevent replay attacks using captured but unexpired envelopes, the Border Gateway MUST maintain a replay prevention cache of recently-seen (`envelope_id`, `session_id`) pairs. The cache MUST cover at minimum the full validity window of the longest-lived envelope the deployment issues. On restart, the Border Gateway MUST either restore the cache from durable storage or refuse to process envelopes issued before the restart until their natural expiry time has elapsed.

When a duplicate (`envelope_id`, `session_id`) pair is detected, the

Border Gateway MUST reject the request with denial reason "replay\_detected" and produce a DENY AER receipt.

#### 8.4.3. Degraded Operation

When the Border Gateway cannot reach the Agent Identity Registry to verify an agent's public key or to retrieve current revocation data, it MAY continue enforcement against locally cached materials subject to the following constraints:

- a) The Border Gateway MUST mark AER receipts produced during degraded operation with an enforcement\_mode field set to "degraded". Relying parties that require non-degraded receipts MAY reject degraded receipts.
- b) The Border Gateway MUST NOT serve stale revocation data beyond its declared cache TTL without marking the resulting receipts as degraded.
- c) The Border Gateway SHOULD record the reason for degraded operation (e.g., "registry\_unreachable", "revocation\_cache\_stale") in the AER receipt for audit purposes.
- d) Implementations SHOULD provide configuration to fail-closed (reject all requests) rather than fail-open (permit with degraded receipts) during registry outages, particularly for deployments in regulated industries where degraded receipts may not satisfy compliance requirements.

#### 9. IANA Considerations

This document has no IANA actions.

#### 10. References

##### 10.1. Normative References

- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.
- [RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, May 2017.
- [RFC8032] Josefsson, S. and I. Liusvaara, "Edwards-Curve Digital Signature Algorithm (EdDSA)", RFC 8032, January 2017.
- [RFC8785] Rundgren, A., Jordan, B., and S. Erdtman, "JSON Canonicalization Scheme (JCS)", RFC 8785, June 2020.
- [RFC6480] Lepinski, M. and S. Kent, "An Infrastructure to Support Secure Internet Routing", RFC 6480, February 2012.
- [RFC6811] Mohapatra, P., Scudder, J., Ward, D., Bush, R., and R. Austein, "BGP Prefix Origin Validation", RFC 6811, January 2013.

##### 10.2. Informative References

- [MCP-SPEC] Anthropic, "Model Context Protocol Specification", 2024, <<https://spec.modelcontextprotocol.io/>>.
- [A2A-SPEC]

Google, "Agent-to-Agent (A2A) Protocol Specification", 2025.

[SCITT-ARCH]

Birkholz, H., et al., "An Architecture for Trustworthy and Transparent Digital Supply Chains", draft-ietf-scitt-architecture, work in progress.

[OWASP-AGENTIC]

OWASP, "OWASP Agentic AI Top 10 2026", December 2025.

[FIDO-PASSKEY]

FIDO Alliance, "Passkey Technical Specification", 2023.

## Appendix A. JSON Schema Definitions

### A.1. ROA Envelope Schema

```
{
  "$schema": "https://json-schema.org/draft/2020-12/schema",
  "$id": "https://nivalto.com/schemas/agentroa/roa-envelope.json",
  "title": "AgentROA ROA Envelope",
  "type": "object",
  "additionalProperties": false,
  "required": [
    "schema_version", "envelope_id", "issued_at", "expires_at",
    "session", "authorized_scope", "policy", "authorization",
    "evidence", "signatures"
  ],
  "properties": {
    "schema_version": { "type": "string", "const": "1.0" },
    "envelope_id": { "type": "string",
      "pattern": "^env:[a-f0-9]{16}$" },
    "issued_at": { "type": "string", "format": "date-time" },
    "expires_at": { "type": "string", "format": "date-time" },
    "session": {
      "type": "object",
      "required": ["session_id", "channel", "agent_id"],
      "properties": {
        "session_id": { "type": "string" },
        "channel": { "type": "string",
          "enum": ["api", "mcp_client", "voice",
            "browser", "mobile_app" ] },
        "agent_id": { "type": "string",
          "pattern": "^aha:[a-zA-Z0-9_-]+/[a-zA-Z0-9_-]+/[a-zA-Z0-9_-]+$" },
        "device_attestation_ref": { "type": "string" }
      }
    },
    "authorized_scope": {
      "type": "object",
      "required": ["capabilities", "max_delegation_depth",
        "cross_org_permitted"],
      "properties": {
        "capabilities": {
          "type": "array",
          "items": { "type": "string" },
          "minItems": 1
        },
        "max_delegation_depth": {
          "type": "integer", "minimum": 0
        },
        "cross_org_permitted": { "type": "boolean" },
        "data_classification_ceiling": { "type": "string" }
      }
    }
  }
}
```

```

    },
    "policy": {
      "type": "object",
      "required": ["policy_id", "policy_version", "policy_digest"],
      "properties": {
        "policy_id": { "type": "string" },
        "policy_version": { "type": "string" },
        "policy_digest": { "type": "string",
          "pattern": "^sha256:[a-f0-9]{64}$" },
        "policy_uri": { "type": "string", "format": "uri" }
      }
    },
    "authorization": {
      "type": "object",
      "required": ["auth_strength", "approval_state"],
      "properties": {
        "auth_strength": { "type": "string",
          "enum": ["session_only", "device_bound",
            "device_bound_with_attestation",
            "dual_control"] },
        "approval_state": { "type": "string",
          "enum": ["pending", "granted", "not_required"] },
        "approval_artifact_ref": { "type": "string" }
      }
    },
    "evidence": {
      "type": "object",
      "required": ["session_hash", "model_provenance"],
      "properties": {
        "session_hash": { "type": "string" },
        "model_provenance": {
          "type": "array",
          "items": { "type": "string" }
        }
      }
    },
    "signatures": {
      "type": "array", "minItems": 1,
      "items": {
        "type": "object",
        "required": ["signer", "alg", "sig"],
        "properties": {
          "signer": { "type": "string" },
          "alg": { "type": "string", "const": "EdDSA" },
          "sig": { "type": "string" }
        }
      }
    }
  }
}

```

## Appendix B. Protocol Object Examples

### B.1. Example ROA Envelope

```

{
  "schema_version": "1.0",
  "envelope_id": "env:4a7c9f2b1e8d3a6f",
  "issued_at": "2026-04-08T14:00:00Z",
  "expires_at": "2026-04-08T14:10:00Z",
  "session": {
    "session_id": "sess:8b3d0e7f2a1c9b4e",
    "channel": "mcp_client",
    "agent_id": "aha:acme-corp/operations/devops-agent-1",
    "device_attestation_ref": "att:ref:k8s-pod-sidecar-001"
  }
}

```

```

},
"authorized_scope": {
  "capabilities": [
    "mcp:aws-cloudwatch.get_metric_data",
    "mcp:aws-cloudwatch.describe_alarms",
    "mcp:github.get_pull_request",
    "mcp:github.list_commits",
    "mcp:pagerduty.get_incident"
  ],
  "max_delegation_depth": 2,
  "cross_org_permitted": false,
  "data_classification_ceiling": "internal"
},
"policy": {
  "policy_id": "devops-incident-investigation-v4",
  "policy_version": "4.2.1",
  "policy_digest":
    "sha256:a3f2c1b9e8d7a6f5e4c3b2a1908f7e6d5c4b3a2918f7e6d5c4b3a291"
},
"authorization": {
  "auth_strength": "session_only",
  "approval_state": "not_required"
},
"evidence": {
  "session_hash":
    "sha256:1b2c3d4e5f6a7b8c9d0e1f2a3b4c5d6e7f8a9b0c1d2e3f4a5b6c7d8e",
  "model_provenance": [
    "anthropic:claude-3-7-sonnet",
    "nivalto:action-classifier:v2"
  ]
},
"signatures": [{
  "signer": "nivalto:policy-engine:prod",
  "alg": "EdDSA",
  "sig": "base64url:MEUCIQDe...Ag=="
}]
}

```

## B.2. Example ARA Object (Delegation Hop)

```

{
  "schema_version": "1.0",
  "ara_id": "ara:9c4e1f8a2b7d3e0f",
  "issued_at": "2026-04-08T14:02:15Z",
  "upstream_ref": {
    "ref_type": "roa_envelope",
    "ref_id": "env:4a7c9f2b1e8d3a6f",
    "ref_digest":
      "sha256:c3b2a1908f7e6d5c4b3a2918f7e6d5c4b3a291a3f2c1b9e8d7a6f5e4"
  },
  "delegating_agent": {
    "agent_id": "aha:acme-corp/operations/devops-agent-1",
    "session_id": "sess:8b3d0e7f2a1c9b4e"
  },
  "delegated_agent": {
    "agent_id": "aha:acme-corp/engineering/coding-agent-7"
  },
  "delegated_scope": {
    "capabilities": [
      "mcp:github.get_pull_request",
      "mcp:github.list_commits"
    ],
    "task_context": "Analyze code changes related to Lambda config",
    "max_delegation_depth": 0
  },
  "policy": {

```

```

    "policy_digest":
      "sha256:a3f2c1b9e8d7a6f5e4c3b2a1908f7e6d5c4b3a2918f7e6d5c4b3a291",
    "policy_version": "4.2.1"
  },
  "signatures": [{
    "signer": "aha:acme-corp/operations/devops-agent-1",
    "alg": "EdDSA",
    "sig": "base64url:MEQCIB...Q=="
  }]
}

```

### B.3. Example AER Receipt (PERMIT)

```

{
  "schema_version": "1.0",
  "aer_id": "aer:2f5a8c1d4e7b0f3a",
  "produced_at": "2026-04-08T14:02:18Z",
  "enforcement_outcome": "permit",
  "session": {
    "session_id": "sess:8b3d0e7f2a1c9b4e",
    "agent_id": "aha:acme-corp/engineering/coding-agent-7",
    "device_attestation_ref": null
  },
  "action": {
    "capability": "mcp:github.get_pull_request",
    "mcp_server_id": "github",
    "mcp_tool_name": "get_pull_request",
    "input_hash":
      "sha256:f1e2d3c4b5a6978889706a5b4c3d2e1f0a9b8c7d6e5f4a3b2c1d0e9f"
  },
  "policy": {
    "policy_id": "devops-incident-investigation-v4",
    "policy_digest":
      "sha256:a3f2c1b9e8d7a6f5e4c3b2a1908f7e6d5c4b3a2918f7e6d5c4b3a291"
  },
  "chain_summary": {
    "chain_depth": 1,
    "root_envelope_id": "env:4a7c9f2b1e8d3a6f",
    "chain_digest":
      "sha256:9f8e7d6c5b4a3918f7e6d5c4b3a2918f7e6d5c4b3a291a3f2c1b9e8d"
  },
  "border_gateway": {
    "gateway_id": "nivalto-bgw:prod-us-east-1-001",
    "gateway_version": "1.0.0"
  },
  "signatures": [{
    "signer": "nivalto-bgw:prod-us-east-1-001",
    "alg": "EdDSA",
    "sig": "base64url:MEYCIQDs...AA=="
  }]
}

```

### Author's Address

Joseph Michalak  
 Nivalto, Inc.  
 Tampa, Florida  
 United States  
 jmichalak@nivalto.com  
<https://nivalto.com>