Internet Engineering Task Force
Internet-Draft
Intended status: Informational
Expires: October 16, 2026

J. Michalak
Nivalto, Inc.

April 16, 2026

# Agent Route Origin Authorization (AgentROA): A Cryptographic Policy Enforcement Framework for AI Agent Actions

draft-nivalto-agentroa-route-authorization-01

## Abstract

This document specifies the Agent Route Origin Authorization (AgentROA) framework, a cryptographic policy enforcement model for governing the actions of autonomous AI agents. 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 capability invocation 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 process at a capability invocation 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. The Border Gateway model is topology-independent: it may be deployed as a protocol-specific proxy in front of Model Context Protocol (MCP) servers, as a service mesh enforcement component covering all inter-service calls, as a network egress gateway covering all outbound capability invocations regardless of protocol, or as a domain-specific execution boundary. The protocol objects defined herein function identically across all deployment topologies. 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 16, 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.

## Change Log (draft-00 to draft-01)

- Revised Abstract and title to remove MCP-specific framing and establish topology-independent scope.
- Added Section 3.4: Deployment Topologies, specifying four Border Gateway deployment configurations (Topology A through D) and establishing that the AgentROA protocol objects are topology-independent.
- Updated Section 1.1 (Motivation) to clarify that enforcement applies across capability invocation boundaries beyond MCP, including service mesh, egress gateway, and domain-specific boundary deployments.
- Updated Section 1.3 (Scope) to state explicitly that the Border Gateway model is topology-independent and that MCP protocol integration is one instantiation, not the exclusive deployment model.
- Updated Section 2 (Terminology) to revise the Border Gateway definition to reflect topology-independence.
- Updated Section 7.1 (Enforcement Proxy Position) to clarify that the reverse proxy arrangement is the MCP-specific instantiation and to reference Section 3.4 for alternative topologies.
- Updated Section 8.3 (Implementation Considerations) to note that Topology B (service mesh) and Topology C (egress gateway) deployments distribute the enforcement function and reduce single-point-of-failure exposure.
- Added Threat T6 (Coverage gap through protocol bypass) to Section 8.1, with mitigation via Topology B or C deployment.

## Table of Contents

# 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. Beyond these protocols, agents may invoke capabilities through shell commands, native SDK integrations, direct database connections, HTTP client libraries, and framework-specific connectors.

Existing connectivity protocols address interoperability. They do not, by themselves, define a cryptographic mechanism that binds an agent's authorized action scope to capability execution, enforces that scope at the invocation 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 a capability invocation boundary, external to the agent's execution context. The Border Gateway enforcement process intercepts capability invocation requests before they reach the target service, 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 Border Gateway may be deployed at the MCP protocol boundary (as a reverse proxy in front of MCP servers), at the service mesh ingress layer (covering all inter-service invocations regardless of protocol), at the network egress boundary (covering all outbound capability invocations), or at a domain-specific execution boundary

such as a payment initiation layer or a production write boundary. The protocol objects defined in this document — the ROA envelope, the ARA chain, and the AER receipt — are identical across all deployment topologies. See Section 3.4 for deployment topology specifications.

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:

- An Agent ROA envelope states which agent identity is authorized to invoke specific capabilities under specific policy constraints, signed by the authorizing entity.
- The Border Gateway validates incoming capability invocation requests against the ROA envelope before forwarding them to the target service. This prevents capability hijacking by ensuring that agents can only invoke capabilities they are explicitly authorized for.
- 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.

The BGP analogy also informs the topology model. Just as BGP border routers enforce routing policy at the boundary between autonomous systems — regardless of the specific traffic type crossing the boundary — the AgentROA Border Gateway enforces capability authorization at the boundary between the agent execution domain and external services — regardless of the specific protocol the agent uses to invoke those services.

### 1.3. Scope

This document specifies:

- The AgentROA framework architecture and trust boundary model.
- The protocol object schemas for ROA envelopes, ARA per-hop attestations, and AER execution receipts.
- The monotonic scope-narrowing semantics for multi-agent

delegation chains.

- The Border Gateway enforcement model, including four deployment topologies (Section 3.4).

- MCP and A2A protocol integration as specific instantiations of the topology-independent enforcement model.

The Border Gateway enforcement model is topology-independent. While this document illustrates the framework using MCP protocol integration as the primary example, the AgentROA enforcement model applies at any process boundary external to the agent's execution context. The protocol objects defined herein function identically across all deployment topologies specified in Section 3.4. MCP protocol integration (Section 7.2) is one instantiation of the Border Gateway model, not the exclusive deployment architecture.

Future extensions, including voice-triggered action binding 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. Where MCP integration is deployed, 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, making direct API calls, executing shell commands, invoking native SDKs, 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 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. The ROA envelope is topology-independent: it governs capability invocations regardless of the protocol or transport mechanism used by the agent.

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 a capability invocation 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. The AER is topology-independent:

it is structurally identical regardless of the deployment topology in which the Border Gateway is operating.

Border Gateway:
An enforcement process that sits at a capability invocation boundary between an agent and external services. 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. The Border Gateway is topology-independent: it may be deployed as a reverse proxy in front of MCP servers (Topology A), as a service mesh enforcement component (Topology B), as a network egress gateway (Topology C), or as a domain-specific execution boundary (Topology D). See Section 3.4.

Capability Identifier:
A structured identifier for a specific capability invocation. For MCP deployments, capability identifiers take the form mcp:<server-id>.<tool-name>. For non-MCP deployments, implementations MAY define alternative capability identifier schemes using the same structured format, with the scheme prefix indicating the capability type (e.g., api:<service>.<operation>, db:<instance>.<operation>).

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: scope(ARA[n]) ⊆ scope(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 process described in Section 7. The Border Gateway is the critical enforcement point. It MUST be deployed external to the agent's execution context. The Border Gateway is topology-independent: the same enforcement logic and protocol objects operate identically across all deployment topologies described in Section 3.4.

d) 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 capability invocations.

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 process runs in a separate process from the governed agent. Deployment MAY use sidecar containers in a Kubernetes pod, separate virtual machines, hardware-isolated enclaves, service mesh sidecar proxies, or dedicated enforcement infrastructure at the network egress boundary. The architectural model in this document does not assume a policy decision point that is co-located within the agent's execution context.

The trust boundary model is invariant across deployment topologies: regardless of whether the Border Gateway is deployed as a protocol-specific MCP proxy (Topology A), a service mesh component (Topology B), a network egress gateway (Topology C), or a domain-specific

boundary (Topology D), the Agent Execution Domain and the Enforcement Domain remain separate. See Section 3.4.

## 3.3. Enforcement Proxy Architecture

The Border Gateway implements the following request handling model for capability invocations. This model is topology-independent: the steps are identical regardless of the deployment topology, with the understanding that the specific protocol interception mechanism varies by topology as described in Section 3.4.

Step 1: Intercept
The Border Gateway receives the capability invocation 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. The interception mechanism is topology-specific: in Topology A (MCP proxy), the agent sends requests to the Border Gateway endpoint; in Topology B (service mesh), the sidecar proxy intercepts the request transparently; in Topology C (egress gateway), the network enforces routing through the Border Gateway; in Topology D (domain boundary), the domain-specific access path passes through the enforcement process.

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 identifier 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: scope(current-ARA) $\subseteq$ scope(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 capability invocation to the target service. 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.

## 3.4. Deployment Topologies

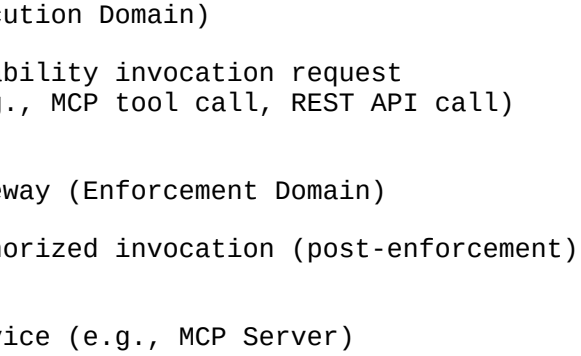
The AgentROA Border Gateway is topology-independent. The protocol objects (ROA envelope, ARA chain, AER receipt) and the enforcement semantics (monotonic scope-narrowing, policy digest validation, pre-execution AER commitment) are identical across all topologies. The topologies differ in the placement of the Border Gateway relative to the agent execution environment and in the classes of capability invocations that fall within the enforcement boundary.

Implementers SHOULD select the topology that provides the coverage appropriate to their threat model and deployment environment. For regulated deployments where comprehensive coverage is required, Topology B or Topology C is RECOMMENDED. Topology A provides the lowest deployment friction and is appropriate where MCP is the primary or exclusive capability invocation mechanism.

#### 3.4.1. Topology A — Protocol-Specific Proxy

Description:
The Border Gateway is deployed as a reverse proxy in front of one or more specific protocol endpoints, such as MCP servers. All agent capability invocations targeting those endpoints are routed through the Border Gateway.

Diagram:




Enforcement scope:
Covers capability invocations directed at the specific protocol endpoints behind the Border Gateway. Invocations using other protocols or targeting other endpoints (e.g., direct SDK calls, shell commands, native library calls) are outside the enforcement boundary in this topology.

Deployment mechanism:
The agent is configured to direct capability invocations to the Border Gateway endpoint rather than directly to the target service. The Border Gateway forwards authorized requests to the target service.

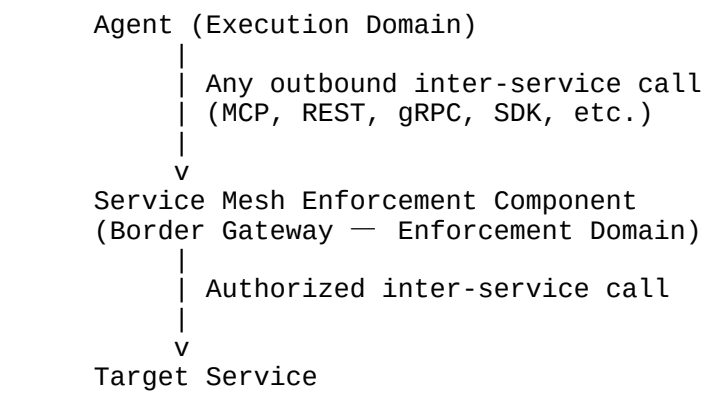
Appropriate use:
Deployments where MCP or a specific named protocol is the primary or exclusive capability invocation mechanism, and where the risk profile of other invocation paths is acceptable. Lowest deployment friction; appropriate for initial deployments and developer environments.

Single-point-of-failure consideration:
The Border Gateway is in the critical path for all covered invocations. Implementations SHOULD deploy multiple Border Gateway instances with consistent state to avoid a single point of failure. See Section 8.3.

#### 3.4.2. Topology B — Service Mesh Enforcement

Description:
The Border Gateway is deployed as a sidecar proxy or ingress enforcement component within a service mesh (e.g., Envoy, Istio, Linkerd). All inter-service calls within the mesh — regardless of application-layer protocol — pass through the enforcement component.

Diagram:




Enforcement scope:
Covers all inter-service capability invocations within the service mesh, regardless of the application-layer protocol. This includes MCP tool calls, REST API calls, gRPC calls, and SDK-based invocations that cross a service boundary. Direct in-process calls and calls that do not cross a service boundary are outside the enforcement boundary.

Deployment mechanism:
AgentROA enforcement logic is integrated into the service mesh sidecar or ingress component. The ROA envelope or ARA chain is presented in a service mesh metadata header. No application-layer code changes are required in the agent.

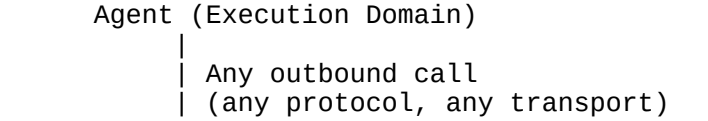
Appropriate use:
Deployments where broad protocol coverage is required and where a service mesh infrastructure is already deployed. Covers capability invocations that would bypass a protocol-specific proxy, including invocations using REST, gRPC, or SDK-native transports.

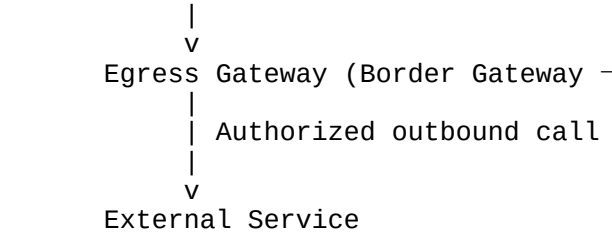
Single-point-of-failure consideration:
Service mesh enforcement distributes the enforcement function across mesh nodes rather than centralizing it in a single proxy. This topology reduces single-point-of-failure exposure relative to Topology A while maintaining consistent enforcement across all covered invocations.

### 3.4.3. Topology C — Egress Gateway Enforcement

Description:
The Border Gateway is deployed at the network egress boundary. All outbound capability invocations from the agent execution environment to external services — regardless of protocol, transport, or application-layer mechanism — are routed through the egress gateway.

Diagram:






Enforcement scope:
Covers all outbound capability invocations from the agent execution environment. This is the broadest enforcement topology. Shell commands that invoke external endpoints, SDK calls, MCP tool calls, direct HTTP requests, and all other outbound invocations pass through the enforcement boundary. In-process operations that do not produce outbound network calls are outside the enforcement boundary.

Deployment mechanism:
Network routing is configured to direct all egress traffic from the agent execution environment through the Border Gateway infrastructure. This MAY be implemented using network policy (e.g., Kubernetes NetworkPolicy, firewall rules, VPN egress routing) that enforces routing through the Border Gateway for all outbound connections.

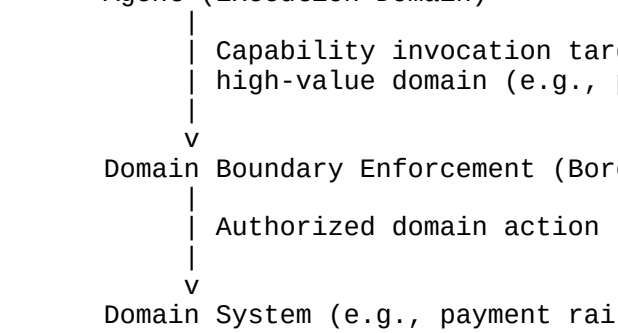
Appropriate use:
High-assurance deployments in regulated environments where comprehensive coverage of all outbound capability invocations is required. Appropriate for Tier 1 regulated deployments (financial services, healthcare, government/defense) where the threat model includes agents using non-MCP invocation paths. Provides the strongest enforcement boundary of the four topologies.

Single-point-of-failure consideration:
As with Topology A, the egress gateway is in the critical path. Implementations SHOULD deploy redundant egress gateway instances. The broader coverage of this topology means that a gateway failure affects all outbound agent capability invocations, not just MCP-targeted ones.

#### 3.4.4. Topology D — Domain-Specific Execution Boundary

Description:
The Border Gateway is deployed at a specific high-value capability boundary within an enterprise system, such as a payment initiation layer, a production write boundary, a PHI data access boundary, or a regulatory filing submission point. Enforcement is concentrated at the boundary where the specific high-risk capability class is invoked.

Diagram:



Enforcement scope:
Covers capability invocations targeting the specific domain system protected by this boundary. Other capability invocations targeting other systems are outside the enforcement boundary.

Deployment mechanism:
The domain system (e.g., payment initiation service, EHR access layer) is configured to require an AgentROA AER receipt as a precondition for processing any agent-initiated request. No agent action in the domain may proceed without a valid AER. This MAY be expressed as a hard invariant: "no receipt, no action."

Appropriate use:
Deployments where specific high-value capability classes require the strongest authorization evidence regardless of how the agent arrived at the boundary. Particularly appropriate for payment authorization (in conjunction with AAX-Pay domain protocol extensions), PHI access under HIPAA, and production system write operations under SOX Section 404. This topology may be combined with Topology A or B to provide defense-in-depth for high-risk capability classes.

#### 3.4.5. Topology Combinations

The four topologies are not mutually exclusive. Implementations MAY deploy multiple topologies simultaneously to provide defense-in-depth. A RECOMMENDED combination for regulated enterprise deployments is:

- Topology A for MCP servers (protocol-specific, low friction)
- Topology D for payment initiation and PHI access (domain-specific, highest assurance at high-value boundaries)

A higher-assurance alternative:

- Topology B or C for broad coverage
- Topology D for domain-specific high-value boundaries

In all multi-topology deployments, the protocol objects (ROA envelope, ARA chain, AER receipt) are identical. A single ROA envelope issued at session initiation governs the agent's capability scope across all deployment topologies active in the session.

## 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. For MCP deployments, identifiers take the form mcp:<server-id>.<tool-name> or the wildcard form mcp:<server-id>.*. For non-MCP deployments, implementations MAY use alternative identifier schemes as noted in the Capability Identifier terminology definition.
- max_delegation_depth: Integer. Maximum number of delegation hops permitted. MUST be >= 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:

a) 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.

b) 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:

B(ARA[n])  B(ARA[n-1])
P(ARA[n])  P(ARA[n-1])

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])  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, regardless of deployment topology. 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. For MCP deployments, wildcard capabilities (mcp:<server-id>.*) are expanded to include all capabilities advertised by the named MCP server's capability manifest. For non-MCP deployments, implementations MUST define equivalent wildcard resolution semantics for their capability identifier scheme.

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 capability invocation.

When a Border Gateway receives a capability invocation request, 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. This algorithm is topology-independent and applies identically across all deployment topologies specified in Section 3.4.

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).

The AER is topology-independent. The structure and semantics of the AER are identical regardless of whether the Border Gateway is deployed in Topology A (MCP proxy), Topology B (service mesh), Topology C (egress gateway), or Topology D (domain boundary). The action.capability field uses the capability identifier scheme appropriate to the deployment topology.

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 8.4.3.

deployment_topology:
   String, OPTIONAL. The deployment topology of the Border Gateway that produced this receipt. One of: "topology_a_protocol_proxy", "topology_b_service_mesh", "topology_c_egress_gateway", "topology_d_domain_boundary". When present, allows relying parties to determine the enforcement scope of this receipt.

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. Uses the capability identifier scheme appropriate to the deployment topology. For MCP topologies, this takes the form mcp:<server-id>.<tool-name>.
- target_service_id: String. Identifier of the target service. For MCP deployments this is the MCP server identifier. For other topologies this is the identifier of the target service or endpoint class.
- operation: String. The specific operation or tool name. For MCP deployments this is the tool name.
- input_hash: String. SHA-256 hash of the canonical serialization of the invocation 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. These invariants apply identically across all deployment topologies.

Invariant 1 (Pre-execution commitment):
For enforcement_outcome "permit", the AER MUST be produced

and persisted to the local audit store BEFORE the capability invocation is forwarded to the target service. This ensures that the AER exists regardless of whether the invocation 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 invocation 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.

SCITT integration operates identically across all deployment topologies.

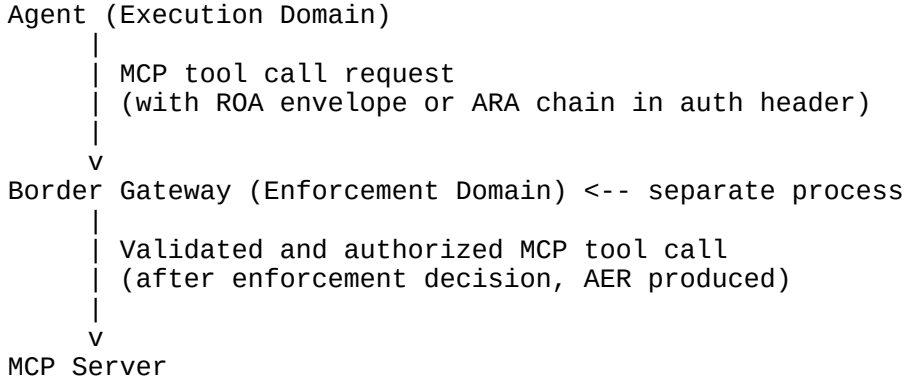
## 7. Border Gateway Enforcement Model

### 7.1. Enforcement Proxy Position

The Border Gateway MUST be positioned at a capability invocation boundary external to the Agent Execution Domain. The specific positioning depends on the deployment topology selected per Section 3.4.

The following diagram illustrates the Topology A (MCP proxy) instantiation, in which the Border Gateway acts as a reverse proxy for MCP connections. This is one instantiation of the Border

Gateway model. For alternative topologies (service mesh, egress gateway, domain boundary), see Section 3.4.




In Topology A, 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.

In Topology B (service mesh), the Border Gateway component is the service mesh sidecar or ingress proxy. The agent does not direct requests explicitly to the Border Gateway; the service mesh intercepts all inter-service calls transparently.

In Topology C (egress gateway), the Border Gateway is the network egress enforcement point. All outbound connections from the agent execution environment are routed through the enforcement infrastructure regardless of protocol.

In Topology D (domain boundary), the Border Gateway is the access control point for the specific domain system. No agent action in the domain may be initiated without a valid AER.

Across all topologies, the following properties hold:
a) The target service never receives unauthorized capability invocations.
b) The enforcement decision is made by infrastructure the agent cannot influence.
c) Every capability invocation — authorized or denied — produces a signed receipt in the enforcement domain.

## 7.2. MCP Protocol Integration

This section specifies AgentROA integration with the Model Context Protocol. This is the Topology A instantiation of the Border Gateway model. MCP integration is one deployment pattern; the AgentROA protocol objects and enforcement semantics are identical for non-MCP deployments under other topologies.

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 topology-independent. The specific implementation of Step 1 (request interception) varies by topology as described in Section 3.4; all subsequent steps are identical.

```
FUNCTION Enforce(request):

  INPUT:
    request: capability invocation 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 = ResolveCapabilityIdentifier(request)
```

```
      // For MCP: mcp:<request.server_id>.<request.tool_name>
      // For other topologies: per capability identifier scheme

    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,
        topology        = CurrentDeploymentTopology()
      )
      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:
        ForwardToTargetService(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 capability invocations:
An adversary embeds malicious instructions in data the agent processes, causing the agent to attempt capability invocations 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. The enforcement decision is made against the capability identifier, not against the content of the agent's reasoning. The sophistication of the injection is irrelevant to the enforcement outcome.

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. This mitigation applies identically across all deployment topologies.

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.

T6 - Coverage gap through protocol bypass:
An agent invokes capabilities using a protocol or mechanism not covered by the deployed Border Gateway topology (e.g., a shell command in a Topology A deployment that covers only MCP servers).
Mitigation: Topology selection. Deployments with broad coverage requirements SHOULD deploy Topology B (service mesh) or Topology C (egress gateway), which enforce at the inter-service boundary or network egress boundary respectively, covering capability invocations regardless of application-layer protocol. The appropriate topology MUST be selected based on the deployment's threat model and the range of capability invocation mechanisms available to the governed agent. Relying parties that require comprehensive coverage SHOULD verify that the deployment_topology field in received AER receipts is consistent with the enforcement scope their compliance requirements demand.

## 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 in Topology A (protocol-specific proxy) and Topology D (domain boundary) deployments, the Border Gateway processes every capability invocation in the covered class. The Border Gateway is therefore a potential bottleneck and single point of failure for those invocation classes. 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 invocation boundary rather than in application middleware.

c) Audit store durability (AER receipts MUST survive Border Gateway failures).

In Topology B (service mesh) deployments, enforcement is distributed across mesh nodes rather than concentrated in a single proxy. This topology reduces single-point-of-failure exposure while maintaining consistent enforcement. The same AER structure and signing semantics apply, with the Border Gateway logic integrated into the mesh sidecar or ingress component.

In Topology C (egress gateway) deployments, the enforcement is concentrated at the network egress boundary. Implementations SHOULD deploy redundant egress enforcement infrastructure, noting that a failure in this topology affects all outbound capability invocations from the agent execution environment.

Implementers SHOULD select the deployment topology based on the threat model and the range of capability invocation mechanisms available to the governed agent. For agents that exclusively use MCP for external capability invocation, Topology A provides adequate coverage with the lowest deployment friction. For agents that use multiple invocation mechanisms, Topology B or C is RECOMMENDED.

### 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" },
        "budget_ceiling": { "type": "number" },
        "budget_unit": { "type": "string" },
        "price_class": { "type": "integer", "minimum": 0 },
        "slo_class": { "type": "integer", "minimum": 0 }
```

```
      }
    },
    "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, Topology A)

```
{
  "schema_version": "1.0",
  "aer_id": "aer:2f5a8c1d4e7b0f3a",
  "produced_at": "2026-04-08T14:02:18Z",
  "enforcement_outcome": "permit",
  "enforcement_mode": "normal",
  "deployment_topology": "topology_a_protocol_proxy",
  "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",
    "target_service_id": "github",
    "operation": "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": "1.1.0"
  },
  "signatures": [{
    "signer": "nivalto-bgw:prod-us-east-1-001",
    "alg": "EdDSA",
    "sig": "base64url:MEYCIQDs...AA=="
  }]
}
```

B.4. Example AER Receipt (DENY, Topology C — Egress Gateway)

```
{
  "schema_version": "1.0",
  "aer_id": "aer:7d3b1e9f5a2c8d4f",
  "produced_at": "2026-04-08T14:05:42Z",
  "enforcement_outcome": "deny",
  "enforcement_mode": "normal",
  "deployment_topology": "topology_c_egress_gateway",
  "denial_reason": "capability_not_in_scope",
  "session": {
    "session_id": "sess:8b3d0e7f2a1c9b4e",
```

```
    "agent_id": "aha:acme-corp/engineering/coding-agent-7"
  },
  "action": {
    "capability": "api:payments.wire-transfer.initiate",
    "target_service_id": "payments-service",
    "operation": "initiate",
    "input_hash":
      "sha256:a1b2c3d4e5f60718192021222324252627282 92a2b2c2d2e2f30313233"
  },
  "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-egress:prod-us-east-1-001",
    "gateway_version": "1.1.0"
  },
  "signatures": [{
    "signer": "nivalto-bgw-egress:prod-us-east-1-001",
    "alg": "EdDSA",
    "sig": "base64url:MEUCIQCx...Bg=="
  }]
}
```

## Author’s Address


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