Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: 20 October 2026

P. Singla
Independent

18 April 2026

Agent Identity Protocol (AIP): Decentralized Identity and Delegation for AI Agents

draft-singla-agent-identity-protocol-01

Abstract

The Agent Identity Protocol (AIP) defines a decentralized identity, delegation, and authorization framework for autonomous AI agents. AIP combines W3C Decentralized Identifiers (DIDs), capability-based authorization, cryptographic delegation chains, and deterministic validation to enable secure, auditable multi-agent workflows without relying on centralized identity providers.

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 20 October 2026.

Copyright Notice

Table of Contents

1. Introduction

Autonomous AI agents are deployed in production environments to act on behalf of human and organisational principals. An agent may send emails, book appointments, make purchases, access file systems, spawn child agents to complete subtasks, and communicate across multiple platforms, all without explicit human approval for each action.

This creates an identity gap. When an agent presents itself to an API, a payment processor, or another agent, there is no standard mechanism to establish: the agent's persistent identity across interactions; the human or organisation on whose authority it acts; the specific actions it is permitted to take; whether it has been compromised or revoked; or whether it has a trustworthy history.

AIP addresses this gap. It is designed as neutral, open infrastructure analogous to HTTP, OAuth, and JWT, providing infrastructure that any application can build on without vendor dependency.

1.1. Motivation

The absence of an identity standard creates concrete operational problems. Services cannot implement fine-grained agent access control. Humans have no auditable record of what their agents did. Compromised agents cannot be reliably stopped. Agent-to-agent systems have no basis for trust.

Existing deployments typically operate in one of two modes: no access, or full access. AIP introduces fine-grained capability declarations. A principal can grant email.read without email.send, or transactions with a max_daily_total constraint.

1.2. Design Philosophy

AIP is built on five design principles:

Neutral and open. The spec is released under CC0 and the did:aip DID method is registered with the W3C.

Build on existing standards. AIP builds on W3C DID, JWT [RFC7519], JWK [RFC7517], DPoP [RFC9449], and CRL patterns from [RFC5280].

Human sovereignty. Every agent action must trace to a human or authorised organisational principal.

Security proportional to risk. AIP defines three security tiers matching overhead to risk level.

Deterministic verification. The Validation Algorithm (Section 9) is fully deterministic: two independent implementations executing the same steps on the same token will reach the same result.

Zero Trust Architecture. AIP implements the zero trust principles defined in [SP-800-207] and satisfies NIST SP 800-63-4 [SP-800-63-4] AAL2 for agent-to-service interactions. No agent is implicitly trusted by virtue of its origin, network location, or prior successful interaction. Every Credential Token must be validated against the Registry on every interaction. Short-lived Credential Tokens bound by a mandatory TTL limit the blast radius of any credential compromise. Revocation is checked in real time for Tier 2 sensitive operations (Section 11.3). DPoP proof-of-possession (Section 21.3) ensures that possession of a valid Credential Token is insufficient for impersonation without the corresponding private key material, satisfying the anti-replay requirement of [SP-800-207] Section 3.

Integration with Model Context Protocol (MCP). AIP provides the agent identity, authorization, and delegation layer that MCP server implementations require but do not define. An AIP-authenticated agent can obtain scoped access tokens for MCP servers via the token exchange mechanism defined in Section 8.4. AIP does not replace MCP; it provides the identity substrate on which MCP authorization decisions are made. See Section 8.4 (Token Exchange) and Section 17.5 (OAuth Authorization Server) for the concrete integration path.

Least Privilege. Agents are granted only the specific capabilities required for their declared purpose, expressed as signed Capability Manifests (Section 5.3). Capability grants are always additive from principal to agent and never exceed the granting principal’s own capability set. A child agent must not be granted capabilities that the delegating parent does not itself hold (Rule D-1, Section 5.10). The max_delegation_depth field (Section 10) bounds the depth of sub-agent hierarchies, limiting transitive capability propagation. Capability constraints allow fine-grained least-privilege expression within each capability category, consistent with [SP-800-207] Section 3 (Tenets 5 and 6).

Relationship to SPIFFE/SPIRE. SPIFFE is designed for workload identity within enumerable, infrastructure-managed environments. AIP is designed for autonomous agents that are created dynamically, operate across organisational boundaries, and act on behalf of named human principals whose authority must be cryptographically traceable. An enterprise may deploy SPIFFE for its internal service mesh and AIP for its agent fleet without conflict. The two mechanisms are complementary and operate at distinct layers of the identity stack.

Relationship to MCP. The Model Context Protocol [MCP] defines how AI agents discover and invoke tools and data sources. AIP is the agent identity layer that sits beneath MCP's authorisation flow: AIP establishes that an agent is who it claims to be (identification via Credential Token), that it was authorised by a named human principal (delegation chain in the Principal Token), and that it holds specific capabilities (Capability Manifest). AIP does not replace MCP's tool-access OAuth flow - it provides the agent identity that OAuth's "sub" claim cannot supply when the subject is an autonomous agent rather than a human user.

Out of scope - Prompt injection prevention. AIP is an identity and authorisation protocol. Whether the content an agent processes contains adversarial instructions is an application-layer and model-layer concern outside this specification's scope. AIP mitigates the persistence window of a successful prompt injection attack: a compromised agent may be immediately revoked via full_revoke (Section 11.1), and revocation propagates to all child agents within the TTL window defined in Section 8.2. AIP does not prevent the initial injection - it limits the attacker's persistence.

1.3. Architecture Layers

AIP is structured as six ordered layers:

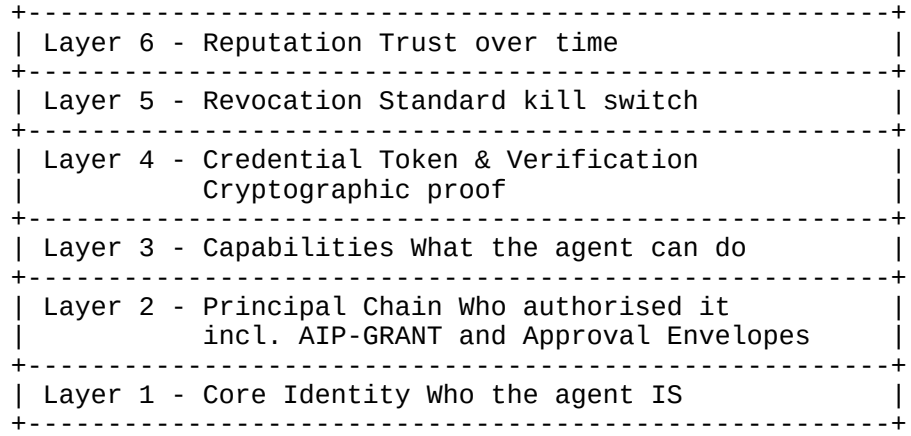


Figure 1: AIP Architecture Layers

2. Conventions and Definitions

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] when, and only when, they appear in all capitals, as shown here.

This document uses "MUST" in preference to "SHALL" throughout for consistency. Where earlier drafts used "SHALL", the normative force is identical; the term has been normalised to "MUST" per this convention.

JSON is used throughout this document as defined in [RFC8259]. URIs are used as defined in [RFC3986]. ABNF notation is used as defined in [RFC5234], including its core rules (ALPHA, DIGIT, HEXDIG, SP, etc.) as defined in RFC5234 Appendix B.1. HTTP status codes, headers, and semantics are as defined in [RFC9110].

All Ed25519 signatures are computed per [RFC8032]. DPoP proofs are constructed per [RFC9449].

2.1. Canonical JSON Serialization

Several objects in this specification are signed outside the JWT framework. For those objects, the following canonical JSON serialization procedure MUST be used when computing or verifying signatures, and when computing Action Hashes:

1 Represent the object as a JSON value per [RFC8259].

2 Recursively sort all object keys in lexicographic ascending order (Unicode code point order). Array element order MUST be preserved; arrays MUST NOT be sorted.

3 Remove all insignificant whitespace (no spaces, tabs, or newlines outside string values).

4 Encode the result as a UTF-8 byte sequence with no BOM.

5 Before computing any signature, set the object's "signature" field to the empty string "". The "signature" field MUST be included in the serialisation at its lexicographically correct position with value "".

This procedure is equivalent to the JSON Canonicalization Scheme (JCS) defined in [RFC8785]. Implementations SHOULD use an RFC8785-conformant library to ensure correctness. When this procedure is used to compute Action Hashes, implementations MUST use an RFC8785-conformant library; custom serializer implementations MUST NOT be used for that purpose.

EXAMPLE (informative):

```
{"z": 1, "a": 2, "m": [3,1,2]}  →  {"a":2,"m":[3,1,2],"z":1}
```

NOTE: This canonical serialization applies to Capability Manifests, Agent Identity Objects, Revocation Objects, Endorsement Objects, GrantRequest Objects (when signed), and Approval Envelopes. It does NOT apply to JWT-format objects (Credential Tokens, Principal Tokens), which use standard JWS compact serialization per [RFC7515].

3. Terminology

The following terms are used throughout this specification:

Agent: An autonomous software system powered by a large language model that can perceive inputs, reason about them, and execute actions in the world on behalf of a principal.

Agent Identity Descriptor (AID): A globally unique, persistent identifier for an agent conforming to the did:aip DID method defined in Section 4.1. An AID is a W3C DID.

Principal: The human or organisational entity on whose authority an agent acts. Every AIP delegation chain MUST have a verifiable principal at its root. Principals are identified by W3C DIDs, but NOT by did:aip (which is reserved for agents).

Agent Deployer: The party that provisions an agent on behalf of a principal. May be the principal themselves or a service acting with the principal's explicit consent.

Principal Wallet: Software that holds the principal's DID private key and implements the AIP-GRANT consent and signing ceremony. Throughout this document, "wallet" used alone refers to the Principal Wallet unless the context explicitly states otherwise.

Capability Manifest: A versioned, signed JSON document stored in the Registry that declares the specific permissions (scopes and constraints) granted to an agent. Defined in Section 5.3.

Credential Token: A signed JWT-format token presented by an agent to a Relying Party as proof of identity and authorisation for a specific interaction. Defined in Section 8.1.

Step Execution Token: A short-lived Credential Token issued by the AIP Registry attesting that a specific Approval Envelope step has been claimed by its designated actor. Defined in Section 13.8.

Principal Token: A signed JWT payload encoding one delegation link in the AIP principal chain. Principal Tokens are embedded as compact-serialised JWTs in the aip_chain array of a Credential Token. Defined in Section 5.5.

Relying Party (RP): Any service, API, or agent that receives and verifies an AIP Credential Token. Responsible for implementing the Validation Algorithm (Section 9).

Delegation: The act of granting a subset of capabilities from a principal or parent agent to a child agent. Formally expressed as a Principal Token in the delegation chain.

Delegation Depth: A non-negative integer counter incremented at each delegation step. An agent directly authorised by the root principal has delegation depth 0. A child agent of that agent has delegation depth 1, and so on.

Tier: A threat-model declaration specifying which security properties the deploying principal asserts hold at execution time. Tier determines the applicable revocation-checking mode, maximum Credential Token lifetime, and mandatory security mechanisms. Three Tiers are defined in Section 3.1 below. Tier selection on performance or availability grounds alone, without regard to the security properties declared, is a misconfiguration and a violation of AIP's zero-trust model.

Grant Tier: One of three standardised AIP-GRANT ceremony profiles: G1 (Registry-Mediated), G2 (Direct Deployer), G3 (Full Ceremony). Defined in Section 12.

Cryptographic Admissibility: The property that an agent's Credential Token and delegation chain are cryptographically valid at claim time: signatures verify, claims are well-formed, the token is not expired, and the delegation chain has not been broken.

Capability Overlay: A Registry-stored, issuer-signed document that restricts an agent's effective capability set for a specific engagement or issuer context. Defined in Section 5.11.

Engagement Object: A mutable Registry resource that models a multi-party engagement, serving as the parent container for Capability Overlays, Approval Envelopes, and participant rosters. Defined in Section 5.12.

Approval Envelope: A signed document submitted to the Registry that pre-authorises a complete multi-step workflow in a single principal signing ceremony. Decouples approval from execution. Defined in Section 13.

Action Hash: A SHA-256 hash of the canonical JSON representation of an approval step's action descriptor. Binds approval to a specific operation. Computed per Section 13.7.

Registry: The authoritative store and resolver for AIP agents, Capability Manifests, revocation state, and delegation chains. Implements the Registry Interface (Section 17) and the Validation Algorithm (Section 9). May be operated by a principal, a service, or a consortium.

AIP-GRANT: The principal authorization protocol defining how principals authorize agents through a consent ceremony. Defined in Section 12.

Revocation Object: A signed document declaring that an agent's grant is revoked, either in whole (full_revoke) or in part (scope_revoke or delegation_revoke). Defined in Section 5.7.

Certificate Revocation List (CRL): A Registry-published list of revoked agents. Updated periodically (Tier 1) or on-demand (Tier 2). Defined in Section 11.2.

DPoP Proof: A Demonstration of Proof-of-Possession as defined in

[RFC9449]. Mandatory for Tier 2 sensitive scopes. Binds a Credential Token to a specific HTTP request and the agent's key material.

Endorsement: A signed statement affirming a positive experience or outcome of an interaction with an agent. Contributes to the agent's reputation score. Defined in Section 14.1.

3.1. Architecture Tiers

AIP defines three security Tiers that map threat-model declarations to mandatory protocol behaviors. Tiers are NOT performance classes; they are security property assertions.

* *Tier 1 (Bounded-staleness):* Optimized for high availability and low latency. Revocation is checked against a Registry-published CRL with a mandatory 15-minute update SLA. Principals MAY use did:key for Tier 1 operations.

* *Tier 2 (Real-time):* The default Tier for sensitive operations. Revocation status MUST be checked in real-time against the Registry on every interaction. DPoP proof-of-possession (RFC 9449) is REQUIRED. Principals MUST use did:web to enable Registry trust anchoring.

* *Tier 3 (Regulated/Enterprise):* The highest security level for regulated or high-value environments. Tier 3 supplements Tier 2 with mandatory mutual TLS (mTLS) and OCSP stapling for transport-layer revocation checks.

The complete mapping of Tiers to normative protocol requirements is defined in the Tier Conformance Table in Section 20.1.

4. Core Identity (did:aip)

Every AIP agent has a globally unique, persistent identifier conforming to the W3C Decentralized Identifier (DID) standard [W3C-DID]. The AIP DID method is did:aip, registered with the W3C DID method registry.

4.1. AID Syntax

An Agent Identity Descriptor (AID) conforms to the following ABNF:

```
AID        = "did:aip:" namespace ":" agent-id
namespace  = LOALPHA *( LOALPHA / DIGIT / "-" )
agent-id   = 32LOHEXDIG
LOALPHA    = %x61-7A        ; a-z only, lowercase
DIGIT      = %x30-39
LOHEXDIG   = DIGIT / "a" / "b" / "c" / "d" / "e" / "f"
                            ; lowercase hex digit only
```

Example: did:aip:personal:9f3a1c82b4e6d7f0a2b5c8e1d4f7a0b3

The namespace MUST begin with a lowercase alpha character, MUST NOT end with a hyphen, and MUST NOT contain consecutive hyphens. Implementations MUST reject AIDs containing uppercase hex digits or uppercase namespace characters.

* namespace: A lowercase alphanumeric string (ALPHA followed by any combination of ALPHA, DIGIT, or hyphens) identifying the agent type. Registered namespaces include: personal, enterprise, service, orchestrator, ephemeral. Custom namespaces MAY be registered.

* agent-id: A 32-character hexadecimal string (lowercase) computed as the SHA-256 hash of the agent's public Ed25519 key material (the JWK x value base64url-decoded).

4.2. AID Derivation

An AID is derived deterministically from the agent's Ed25519 public key:

1. Generate an Ed25519 keypair per [RFC8032].

2. Encode the public key as a JWK with kty="OKP", crv="Ed25519", and the public key value in the x field (base64url-encoded).

3. Compute SHA-256(base64url_decode(x)) to obtain a 32-byte hash.

4. Hex-encode the hash (lowercase) to form the agent-id.

5. Combine with the chosen namespace to form the complete AID: did:aip:<namespace>:<agent-id>.

This derivation is deterministic and cryptographically self-verifying: possession of the private key is both necessary and sufficient to claim ownership of the AID.

4.3. Registered Namespaces

The following namespaces are registered:

personal: An AI assistant serving a single human principal. Personal agents declare this namespace.

enterprise: An agent deployed by an organisation for use within the organisation.

service: A service-owned agent providing functionality to principals (e.g., payment processor, communication provider).

orchestrator: An agent that spawns and manages sub-agents. Typically has spawn_agents.create and spawn_agents.manage scopes.

ephemeral: A short-lived agent created for a single task. Ephemeral agents MUST include a task_id in their Principal Token and MUST have a limited delegation_valid_for_seconds (typically < 1 hour).

registry: The registry namespace is reserved for Registry-owned agent identities when a Registry deployment exposes agent-like actors. It MUST NOT be used as the stable Registry service identifier. The stable Registry service identifier is registry_id as defined in Section 7.3.4.

Custom namespaces MAY be registered with the W3C and the AIP registry. Namespaces are immutable once registered.

4.4. Agent Identity Object Schema

The Agent Identity Object is the base document for an agent. It includes the AID, public key material, and metadata. The schema is defined in Section 5.2 (see also agent-identity.schema.json in the schemas directory).

Key fields:

* aid: The agent’s AID (REQUIRED)

* name: Human-readable agent name (REQUIRED)

* type: The namespace component of the AID (REQUIRED, MUST match)

* model: AI model information including provider and model_id (REQUIRED)

* public_key: JWK format Ed25519 public key (REQUIRED)

* created_at: ISO 8601 timestamp (REQUIRED)

* version: Identity version number for key rotation (REQUIRED, minimum 1)

* previous_key_signature: EdDSA signature by previous key when rotating (REQUIRED if version > 1)

4.5. Uniqueness and Immutability

An AID MUST be globally unique. Once an agent is registered in a Registry, its AID is immutable. An AID cannot be reused or transferred. Two agents with identical public keys will derive identical AIDs, so the cryptographic relationship is deterministic.

5. Resource Model and Data Structures

This section defines the core data structures in AIP: JSON Schema representations of agents, principals, capabilities, tokens, and enrollment objects. All objects conform to the canonical JSON serialization rules of Section 2.1.

5.1. Resource Naming

Agent Identities (AIDs) are W3C Decentralized Identifiers [W3C-DID] using the did:aip method. The ABNF grammar uses core rules from [RFC5234] Appendix B.1 (DIGIT) and defines two additional terminal rules:

```
aip-did    = "did:aip:" namespace ":" unique-id
namespace  = LOALPHA *( LOALPHA / DIGIT )
             *( "-" 1*( LOALPHA / DIGIT ) )
unique-id  = 32LOHEXDIG
LOALPHA    = %x61-7A        ; a-z only, lowercase
LOHEXDIG   = DIGIT / "a" / "b" / "c" / "d" / "e" / "f"
                            ; lowercase hex digit only
```

The namespace MUST begin with a lowercase alpha character, MUST NOT end with a hyphen, and MUST NOT contain consecutive hyphens. Implementations MUST reject AIDs containing uppercase hex digits or uppercase namespace characters.

The following namespace values are defined:

Namespace	Description
personal	An agent acting for a single human principal
enterprise	An agent acting within an organisational deployment
service	A persistent agent providing a capability as a service
ephemeral	An agent created for a single task; revoked on completion
orchestrator	An agent whose primary function is spawning child agents
registry	An AIP Registry instance; at most one active AID per deployment

Table 1: Defined Namespace Values

A registry-namespace AID MUST NOT be registered via the standard POST /v1/agents endpoint. It MUST be created exclusively via the Registry Genesis procedure.

Compound typed identifiers use prefixed UUID v4 values:

Object Type	Prefix	Example
Capability Manifest	cm:	cm:550e8400-e29b-41d4-a716-...
Revocation Object	rev:	rev:6ba7b810-9dad-11d1-80b4-...
Endorsement Object	end:	end:6ba7b811-9dad-11d1-80b4-...
Grant Request	gr:	gr:550e8401-e29b-41d4-a716-...
Approval Envelope	apr:	apr:550e8402-e29b-41d4-a716-...

Table 2: Typed Identifier Prefixes

5.2. Agent Identity Object

The Agent Identity Object is the canonical signed JSON document that establishes an agent's identity. Canonical signing field order: aid, name, type, model, created_at, version, public_key, previous_key_signature.

aid Type: string. Required: REQUIRED. Constraints: MUST match did:aip ABNF; pattern is lowercase namespace plus 32 lowercase hexadecimal characters.

name Type: string. Required: REQUIRED. Constraints: minLength: 1, maxLength: 64.

type Type: string. Required: REQUIRED. Constraints: MUST exactly match namespace component of aid; pattern is lowercase alphanumeric with optional hyphen-separated segments.

model Type: object. Required: REQUIRED. Constraints: See sub-fields below.

model.provider Type: string. Required: REQUIRED. Constraints: minLength: 1, maxLength: 64.

model.model_id Type: string. Required: REQUIRED. Constraints: minLength: 1, maxLength: 128.

model.attestation_hash Type: string. Required: OPTIONAL. Constraints: pattern ^sha256:[0-9a-f]{64}$.

created_at Type: string. Required: REQUIRED. Constraints: ISO 8601 UTC; format: date-time; immutable after registration.

version Type: integer. Required: REQUIRED. Constraints: minimum: 1; MUST increment by exactly 1 on key rotation.

public_key Type: object. Required: REQUIRED. Constraints: JWK per [RFC7517]; Ed25519 (kty=OKP, crv=Ed25519) per [RFC8037].

public_key.kty Type: string. Required: REQUIRED. Constraints: const: "OKP".

public_key.crv Type: string. Required: REQUIRED. Constraints: const: "Ed25519".

public_key.x Type: string. Required: REQUIRED. Constraints: pattern ^[A-Za-z0-9_-]{43}$ (32 bytes base64url, no padding).

public_key.kid Type: string. Required: REQUIRED. Constraints: DID URL using the agent AID followed by #key-N where N is a positive integer.

previous_key_signature Type: string. Required: OPTIONAL (version=1); REQUIRED (version>=2). Constraints: base64url EdDSA signature of new object (with this field set to ""); pattern ^[A-Za-z0-9_-]+$.

Normative requirements:

* The aid, type, and created_at fields MUST NOT change after initial registration.

* The type field MUST exactly match the namespace component of the aid field.

* The version field MUST start at 1 and MUST increment by exactly 1 on each key rotation.

* The public_key.kid MUST use format <aid>#key-<n> where <n> starts at 1.

* When version is 2 or greater, previous_key_signature MUST be present and MUST be a non-empty base64url string.

5.3. Capability Manifest

The Capability Manifest is a versioned, signed JSON document that declares the specific permissions granted to an agent. Canonical signing field order: manifest_id, aid, granted_by, version, issued_at, expires_at, capabilities, signature.

Field	Type	Required	Constraints
manifest_id	string	REQUIRED	pattern: ^cm:[0-9a-f]{8}-[0-9a-f]{4}-4[0-9a-f]{3}-[89ab][0-9a-f]{3}-[0-9a-f]{12}$
aid	string	REQUIRED	MUST match did:aip ABNF
granted_by	string	REQUIRED	MUST be a valid W3C DID; pattern: any valid DID method and suffix
version	integer	REQUIRED	minimum: 1; MUST increment on every update including scope_revoke
issued_at	string	REQUIRED	ISO 8601 UTC; format: date-time
expires_at	string	REQUIRED	ISO 8601 UTC; MUST be after issued_at
capabilities	object	REQUIRED	See Section 5.9 for all sub-fields
signature	string	REQUIRED	base64url EdDSA signature; pattern: ^[A-Za-z0-9_-]+$

Table 3: Capability Manifest Fields

A new manifest_id MUST be generated on every update, including scope_revoke operations. Relying Parties MUST verify the manifest signature field using the public key of the granted_by DID before trusting any capability declared within it.

5.4. Credential Token

An AIP Credential Token is a compact JWT with the following format:

```
aip-token = JWT-header "." JWT-payload "." JWT-signature
```

The token MUST be a valid JWT as defined by [RFC7519].

JWT Header fields:

Field	Requirement	Value / Constraints
typ	MUST	"AIP+JWT"
alg	MUST	"EdDSA" (REQUIRED); "ES256" (OPTIONAL); "RS256" (OPTIONAL, legacy enterprise only)
kid	MUST	DID URL identifying the signing key; MUST match the kid in the signing agent's Agent Identity

Table 4: JWT Header Fields

Credential Token Payload fields:

Field	Type	Required	Constraints
aip_version	string	REQUIRED	MUST be "0.3" for this spec
iss	string	REQUIRED	MUST match did:aip ABNF; MUST equal sub of last aip_chain element
sub	string	REQUIRED	MUST match did:aip ABNF; MUST equal iss for non-delegated tokens
aud	string/ array	REQUIRED	Single string or array; MUST include the Relying Party's identifier
iat	integer	REQUIRED	Unix timestamp; MUST NOT be in the future (30-second clock skew tolerance)
exp	integer	REQUIRED	Unix timestamp; MUST

			be strictly greater than iat; TTL limits per Section 8.2
jti	string	REQUIRED	UUID v4 canonical lowercase
aip_scope	array	REQUIRED	minItems: 1; uniqueItems: true; each item matches ^[a-z_]+([.][a-z_]+)*$
aip_chain	array	REQUIRED	minItems: 1; maxItems: 11; each element is a compact-serialised signed Principal Token JWT
aip_registry	string	OPTIONAL	URI of AIP Registry
aip_approval_id	string	OPTIONAL	pattern: ^apr:[0-9a-f]{8}-...$; REQUIRED when token is a step-claim token
aip_approval_step	integer	OPTIONAL	Positive integer (>= 1); REQUIRED when aip_approval_id is present. Uses 1-based indexing.
aip_engagement_id	string	OPTIONAL	pattern: ^eng:[0-9a-f]{8}-...$; present when token is scoped to an Engagement Object

Table 5: Credential Token Payload Fields

5.5. Principal Token

A Principal Token is a JWT payload encoding one delegation link in the AIP principal chain. Principal Tokens are embedded as compact-serialised JWTs in the aip_chain array of a Credential Token.

Field	Type	Required	Constraints
iss	string	REQUIRED	W3C DID or did:aip AID; for delegation_depth 0: MUST equal principal.id; for delegation_depth > 0: MUST equal delegated_by
sub	string	REQUIRED	MUST match did:aip ABNF
principal	object	REQUIRED	See sub-fields below
principal.type	string	REQUIRED	enum: ["human", "organisation"]
principal.id	string	REQUIRED	W3C DID; MUST NOT use did:aip method; MUST be byte-for-byte identical across all chain elements
delegated_by	string/ null	REQUIRED	null when delegation_depth is 0; MUST be a did:aip AID when delegation_depth > 0
delegation_depth	integer	REQUIRED	minimum: 0, maximum: 10; MUST equal the array index of this token in aip_chain
max_delegation_depth	integer	REQUIRED	minimum: 0, maximum: 10; default: 3 when absent; only the value from

			aip_chain[0] governs the chain
issued_at	string	REQUIRED	ISO 8601 UTC; format: date-time
expires_at	string	REQUIRED	ISO 8601 UTC; MUST be strictly after issued_at
purpose	string	OPTIONAL	maxLength: 128
task_id	string/ null	OPTIONAL; REQUIRED for ephemeral agents	minLength: 1, maxLength: 256 when non-null
scope	array	REQUIRED	minItems: 1; uniqueItems: true
acr	string	OPTIONAL; REQUIRED when grant ceremony is G3	Authentication context class reference
amr	array	OPTIONAL; REQUIRED when grant ceremony is G3	Authentication method reference values per [RFC8176]

Table 6: Principal Token Fields

The principal.id field MUST be byte-for-byte identical across every Principal Token in the same aip_chain array. The principal.id MUST NOT begin with did:aip:.

5.6. Registration Envelope

The Registration Envelope is the request body submitted to POST /v1/agents to register a new AIP agent.

Field	Type	Required	Constraints
identity	object	REQUIRED	MUST conform to agent-identity schema; version MUST be 1; previous_key_signature MUST NOT be present
capability_manifest	object	REQUIRED	MUST conform to capability-manifest schema; version MUST be 1; aid MUST equal identity.aid
principal_token	string	REQUIRED	Compact JWT (header.payload.signature); delegation_depth MUST be 0; delegated_by MUST be null
grant_tier	string	REQUIRED	enum: ["G1", "G2", "G3"]; MUST be consistent with principal_token's delegation_depth and the scopes in capability_manifest

Table 7: Registration Envelope Fields

5.7. Revocation Object

Revocation is performed by submitting a signed Revocation Object to POST /v1/revocations. Canonical signing field order: revocation_id, target_id, type, issued_by, reason, timestamp, propagate_to_children, scopes_revoked, signature.

Field	Type	Required	Constraints
revocation_id	string	REQUIRED	pattern: ^rev:[0-9a-f]{8}-[0-9a-f]{4}-4[0-9a-f]{3}-[89ab][0-9a-f]{3}-[0-9a-f]{12}$
target_id	string	REQUIRED	MUST be a did:aip AID (for full_revoke, scope_revoke, delegation_revoke) or a Principal DID (for principal_revoke)
type	string	REQUIRED	enum: ["full_revoke", "scope_revoke", "delegation_revoke", "principal_revoke"]
issued_by	string	REQUIRED	W3C DID; in target's chain
reason	string	REQUIRED	enum of defined reasons
timestamp	string	REQUIRED	ISO 8601 UTC
propagate_to_children	boolean	OPTIONAL	default: false
scopes_revoked	array	REQUIRED when scope_revoke; MUST NOT otherwise	minItems: 1
signature	string	REQUIRED	base64url EdDSA

Table 8: Revocation Object Fields

Revocation Types:

* *full_revoke* - Permanently revokes the AID.

* *scope_revoke* - Invalidates specific scopes for the agent. For Tier 1 and 2, this MUST cause Relying Parties to reject Credential Tokens containing the revoked scopes (Step 7, Section 9).

* *delegation_revoke* - Invalidates delegation chains rooted at the target. Child agents lose authority; the target AID remains valid for direct principal interactions.

* *principal_revoke* - Issued by the root principal to revoke their entire authorisation of the target agent (via AID) or all agents under their authority (via Principal DID).

5.8. Endorsement Object

Any Relying Party or agent MAY submit a signed Endorsement Object to the Registry after a completed interaction. Canonical signing field order: endorsement_id, from_aid, to_aid, task_id, outcome, notes, timestamp, signature.

Field	Type	Required	Constraints
endorsement_id	string	REQUIRED	pattern: ^end:[0-9a-f]{8}-[0-9a-f]{4}-4[0-9a-f]{3}-[89ab][0-9a-f]{3}-[0-9a-f]{12}$
from_aid	string	REQUIRED	MUST NOT equal to_aid
to_aid	string	REQUIRED	MUST NOT equal from_aid
task_id	string	REQUIRED	minLength: 1, maxLength: 256
outcome	string	REQUIRED	enum: ["success", "partial", "failure"]
notes	string/ null	OPTIONAL	maxLength: 512
timestamp	string	REQUIRED	ISO 8601 UTC
signature	string	REQUIRED	base64url EdDSA of from_aid

Table 9: Endorsement Object Fields

The Registry MUST verify every submitted Endorsement Object signature. Only success and partial outcomes increment endorsement_count. failure increments incident_count.

5.9. Field Constraints and Defined Scope Identifiers

Capability capabilities sub-fields are defined in full in schemas/v0.3/capability-manifest.schema.json.

Defined Scope Identifiers:

email.read email.write email.send
email.delete calendar.read calendar.write
calendar.delete filesystem.read filesystem.write
filesystem.execute filesystem.delete web.browse
web.forms_submit web.download transactions
communicate.whatsapp communicate.telegram communicate.sms
communicate.voice spawn_agents.create spawn_agents.manage

NOTE: The bare scope spawn_agents is retired as of v0.3 and is NOT a defined scope identifier. Validators MUST reject tokens containing spawn_agents without a sub-scope qualifier with invalid_scope. spawn_agents.create and spawn_agents.manage are Tier 2 scopes with a maximum TTL of 300 seconds; DPoP is REQUIRED.

Where this document references transactions.* or communicate.*, this means the bare capability key OR any scope beginning with that prefix OR capabilities.transactions.enabled: true / capabilities.communicate.enabled: true in the Capability Manifest.

Empty arrays [] for filesystem.read or filesystem.write MUST be interpreted as deny-all. A require_confirmation_above value above max_single_transaction is vacuous and MUST be rejected. When communicate.enabled is true, at least one channel MUST be explicitly set to true.

5.10. Delegation Rules

Rule D-1. A delegated agent MUST NOT grant scopes or looser constraint values than its own Capability Manifest contains.

Rule D-2. A delegated agent MUST NOT issue a Principal Token with max_delegation_depth greater than its remaining depth (max_delegation_depth - delegation_depth).

Rule D-3. Implementations MUST reject any Credential Token where the delegation_depth of any chain token exceeds the root token’s max_delegation_depth.

Rule D-4. The root Principal Token (index 0) MUST have delegation_depth = 0. Each subsequent token at index i MUST have delegation_depth = i. No gaps, skips, or repeated values are permitted.

Rule D-5. Each delegation chain token MUST be signed by the private key of the delegated_by AID (or root principal for depth 0).

5.11. Capability Overlays

A Capability Overlay is a signed restriction document stored in the Registry. It narrows an agent’s effective capability set for operations within a specific engagement or issuer context.

Rule CO-1 (Attenuation Only): An overlay MUST NOT expand any constraint value beyond what the base Capability Manifest permits. The effective capability set is always the intersection of the base manifest and all active overlays scoped to the engagement or issuer.

Field	Type	Required	Constraints
overlay_id	string	REQUIRED	Pattern: ^co:[0-9a-f]{8}-[0-9a-f]{4}-4[0-9a-f]{3}-[89ab][0-9a-f]{3}-[0-9a-f]{12}$
aid	string	REQUIRED	The target agent's AID
engagement_id	string	OPTIONAL	Pattern: ^eng:...$; links to Engagement Object
issued_by	string	REQUIRED	DID of the overlay issuer; MUST be did:web or did:aip (MUST NOT be did:key)
overlay_type	string	REQUIRED	MUST be "restrict"
issued_at	string	REQUIRED	ISO 8601 UTC timestamp
expires_at	string	REQUIRED	ISO 8601 UTC; MUST be strictly after issued_at
version	integer	REQUIRED	Positive integer; monotonically increasing per (aid, engagement_id, issued_by) tuple
constraints	object	REQUIRED	Uses the same schema as Capability Manifest capabilities sub-object
signature	string	REQUIRED	Base64url EdDSA signature by issued_by over JCS canonical JSON

Table 10: Capability Overlay Fields

Overlay Rules:

1. *CO-1 (Attenuation Only):* For every field in constraints, the value MUST be equal to or more restrictive than the corresponding value in the base Capability Manifest.

2. *CO-2 (Issuer DID Method):* The issued_by DID MUST use did:web or did:aip. Overlays signed by did:key MUST be rejected with overlay_issuer_invalid.

3. *CO-3 (Version Monotonicity):* A new overlay for the same (aid, engagement_id, issued_by) tuple MUST have a strictly higher version than the current active overlay.

4. *CO-4 (Expiry Handling):* Expired overlays MUST be treated as absent.

5. *CO-5 (Engagement Termination):* If an Engagement Object associated with an overlay's engagement_id is terminated, all overlays for that engagement MUST be invalidated atomically by the Registry.

6. *CO-6 (Multiple Overlays):* When multiple overlays apply to the same agent, the effective capability set is the intersection of ALL applicable overlays with the base manifest.

Effective Capability Computation:

```
effective = base_manifest.capabilities
for each active_overlay in applicable_overlays:
    effective = intersect(effective, active_overlay.constraints)
```

Where intersect applies per-field: boolean fields use AND (false wins); numeric limits use min(base, overlay); path arrays use set intersection; scope set to false removes that scope.

5.12. Engagement Objects

An Engagement Object is a mutable Registry resource that models a multi-party engagement. It is the parent container for Capability Overlays scoped via engagement_id, Approval Envelopes scoped via engagement_id, and participant roster and approval gate state.

Field	Type	Required	Constraints
engagement_id	string	REQUIRED	Pattern: ^eng:[0-9a-f]{8}-[0-9a-f]{4}-4[0-9a-f]{3}-[89ab][0-9a-f]{3}-[0-9a-f]{12}$
title	string	REQUIRED	maxLength: 256

hiring_operator	string	REQUIRED	DID; MUST be did:web or did:aip
deploying_principal	string	REQUIRED	DID of the deploying principal
created_at	string	REQUIRED	ISO 8601 UTC
expires_at	string	REQUIRED	ISO 8601 UTC; MUST be after created_at
status	string	REQUIRED	One of: "proposed", "active", "suspended", "completed", "terminated"
participants	array	REQUIRED	Array of Participant objects
approval_gates	array	OPTIONAL	Array of Approval Gate objects
change_log	array	REQUIRED	Append-only array of Change Log Entry objects
hiring_operator_signature	string	REQUIRED	Base64url EdDSA signature by hiring_operator over the top-level engagement signing input defined below
deploying_principal_signature	string	REQUIRED	Base64url EdDSA countersignature by deploying_principal over the same top-level engagement signing input defined below
version	integer	REQUIRED	Monotonically increasing

Table 11: Engagement Object Fields

Participant object fields: aid (REQUIRED), role (REQUIRED, maxLength: 64), capability_overlay_id (OPTIONAL), added_at (REQUIRED, ISO 8601), added_by (REQUIRED, DID), removed_at (OPTIONAL, ISO 8601).

Approval Gate object fields: gate_id (REQUIRED, pattern: ^gate:[a-z0-9_-]+$), name (REQUIRED, maxLength: 128), required_approver (REQUIRED, DID), trigger (REQUIRED, string), status (REQUIRED, one of: "pending", "approved", "rejected"), approved_at (OPTIONAL, ISO 8601).

Top-level engagement signature input: Both hiring_operator_signature and deploying_principal_signature MUST be computed over the same JCS-canonical JSON serialization of the Engagement Object after removing both top-level signature fields (hiring_operator_signature and deploying_principal_signature). This signing input includes all other Engagement Object fields, including change_log and version. The countersignature does not sign the other party's signature value.

Change Log Entry fields: seq (REQUIRED, monotonically increasing from 1), timestamp (REQUIRED, ISO 8601), action (REQUIRED), actor (REQUIRED, DID), payload (OPTIONAL, object), signature (REQUIRED, Base64url EdDSA by actor over JCS canonical JSON excluding signature).

Defined change log actions: engagement_created, engagement_countersigned, participant_added, participant_removed, participant_role_changed, gate_added, gate_approved, gate_rejected, engagement_suspended, engagement_resumed, engagement_completed, engagement_terminated.

The Registry MUST reject any request that modifies or deletes an existing change log entry. Only appends are permitted.

Engagement Lifecycle:

```
  proposed --> active --> completed
                 |
                 +--> suspended --> active (resumed)
                 |
                 +--> terminated
```

Engagement Lifecycle Transitions:

* proposed to active: Requires both hiring_operator_signature and deploying_principal_signature.

* active to suspended / completed: Hiring operator only.

* active/suspended to terminated: Either party.

Termination Cascade: When an Engagement is terminated:

1. The Registry MUST set status: "terminated" atomically.

2. All Capability Overlays linked to this engagement_id MUST be invalidated (per Rule CO-5).

3. All Approval Envelopes in pending_approval, approved, or executing status scoped to this engagement_id MUST be transitioned to cancelled.

4. Subsequent Credential Token validation for operations scoped to this engagement MUST fail with engagement_terminated.

6. Registration Protocol

6.1. Registration Envelope

An agent is registered by submitting a Registration Envelope to ‘POST /v1/agents‘. The fields are defined in Section 5.6.

The ‘principal_token‘ MUST have ‘delegation_depth‘ equal to 0 and ‘delegated_by‘ equal to null. The ‘sub‘ field of the decoded principal token payload MUST equal ‘identity.aid‘.

6.2. Registration Validation

The Registry MUST perform the following checks in order before accepting a Registration Envelope. The Registry MUST either accept all or reject all — partial registration MUST NOT be possible.

1. Check 1. ‘identity‘ MUST be present and MUST be valid JSON conforming to the Agent Identity schema.

2. Check 2. ‘identity.aid‘ MUST match the ‘did:aip‘ ABNF grammar defined in Section 4.1.

3. Check 3. ‘identity.type‘ MUST equal the namespace component of ‘identity.aid‘. A mismatch MUST be rejected.

4. Check 4. ‘identity.aid‘ MUST NOT already exist in the Registry.

5. Check 5. ‘identity.public_key‘ MUST be an Ed25519 JWK with ‘kty="OKP"‘, ‘crv="Ed25519"‘, and a 43-character base64url ‘x‘ value.

6. Check 6. ‘capability_manifest.expires_at‘ MUST be in the future at the time of registration.

7. Check 7. ‘capability_manifest.aid‘ MUST equal ‘identity.aid‘.

8. Check 8. The ‘principal_token‘ string MUST be decodable as a compact-serialised JWT and MUST conform to the Principal Token schema.

9. Check 9. The decoded ‘principal_token‘ payload ‘sub‘ MUST equal ‘identity.aid‘.

10. Check 10. The decoded ‘principal_token‘ payload ‘principal.id‘ MUST NOT begin with ‘did:aip:‘.

11. Check 11. If ‘identity.type‘ is ‘ephemeral‘, the decoded ‘principal_token‘ payload ‘task_id‘ MUST be non-null and non-empty.

12. Check 12. ‘capability_manifest.signature‘ MUST be verifiable against the ‘granted_by‘ DID’s public key.

13. Check 13. If ‘identity.version‘ is 2 or greater, ‘identity.previous_key_signature‘ MUST be present and MUST verify using the key at the previous version.

14. Check 14. ‘grant_tier‘ MUST be present and MUST be one of ‘"G1"‘, ‘"G2"‘, or ‘"G3"‘. If any scope in ‘capability_manifest.capabilities‘ is a Tier 2 scope (per Section 3.1), ‘grant_tier‘ MUST be ‘"G2"‘ or ‘"G3"‘. If any scope is Tier 3, ‘grant_tier‘ MUST be ‘"G3"‘. If ‘grant_tier‘ is absent, the Registry MUST reject with ‘registration_invalid‘.

The Registry MUST record the ‘delegated_by‘ field from the decoded ‘principal_token‘ payload (or the principal’s DID if ‘delegated_by‘ is null) in the parent-child delegation index, for use in revocation propagation (Section 11.3).

6.3. Error Responses

All AIP error responses MUST use the format defined in Section 18.1. Implementations MUST NOT return HTTP 200 for error conditions.

7. Agent Resolution

7.1. DID Resolution

Every AID MUST have a corresponding W3C DID Document resolvable by any standard DID resolver. The DID Document is derived deterministically from the Agent Identity stored in the Registry.

The Registry MUST generate and serve DID Documents from GET /v1/agents/{aid} when the Accept: application/did+ld+json or Accept: application/did+json header is present.

AIP implementations MUST support DID resolution for at minimum did:key and did:web DID methods. Resolved DID Documents MAY be cached for a maximum of 300 seconds. If resolution fails or times out, implementations MUST treat the result as registry_unavailable and MUST reject the operation.

If the AID has been revoked, the Registry MUST return deactivated: true in _didDocumentMetadata_, per W3C-DID Section 7.1.2.

7.2. DID Document Structure

A conformant did:aip DID Document MUST include @context, id, verificationMethod, authentication, and controller. See the canonical example in the repository at _examples/latest/did-document.json_.

A principal’s DID Document that authorises agents for Tier 2 operations MUST include an AIPRegistry service entry in its service array. This requirement applies to principal DID methods other than did:aip; principals MUST NOT use the did:aip method anywhere in the protocol.

```
{
  "id": "did:web:acme.com#aip-registry",
  "type": "AIPRegistry",
  "serviceEndpoint": "https://registry.acme.com"
}
```

The serviceEndpoint MUST be an HTTPS URI pointing to the base URL of the authoritative AIP Registry for agents delegated by this principal. The type MUST be exactly "AIPRegistry" (case-sensitive).

For principals using did:key: an AIPRegistry service entry cannot be declared (DID Key documents have no service array). Per Section 12, did:key principals MUST NOT authorise Tier 2 operations.

For principals using did:web or another non-did:aip method that supports service entries: the AIPRegistry service entry is REQUIRED when the principal authorises any agent with Tier 2 scopes. It is OPTIONAL for principals authorising only Tier 1 agents.

The CRUD operations for did:aip are:

Operation	Mechanism
Create	POST /v1/agents with Registration Envelope
Read	GET /v1/agents/{aid} with DID Accept header
Update	PUT /v1/agents/{aid} for key rotation only
Deactivate	POST /v1/revocations with _full_revoke_ object

Table 12

7.3. Registry Genesis

Registry Genesis is the one-time initialisation procedure by which a new AIP Registry establishes its stable service identifier, generates its initial trust keys, and publishes discovery and trust metadata. It MUST be performed before the Registry serves any requests.

7.3.1. Key Generation

The Registry MUST generate a fresh Ed25519 trust keypair at first boot. The private key MUST be stored encrypted at rest (AES-256-GCM or equivalent). The corresponding public key is published in the initial Registry Trust Record. Separate active verification keys for CRL documents (Section 11.2) and Step Execution Tokens are RECOMMENDED so that routine operational-key rotation does not require trust-root rotation.

7.3.2. Registry ID Establishment

The Registry MUST establish a stable registry_id for the Registry service. The registry_id MUST be an HTTPS URI under the Registry operator’s control. The RECOMMENDED value is the origin that serves /.well-known/aip-registry.

```
https://registry.example.com
```

The registry_id identifies the Registry service, not a single signing key. It MUST remain stable across planned key rotation. At most one active registry_id MUST exist per Registry deployment at any time.

7.3.3. Self-Registration Exemption

The Registry service identifier MUST NOT be created via POST /v1/agents. Instead, the Registry MUST persist its registry_id and trust keys directly to its own data store during genesis. The following Registration Envelope checks (Section 6.2) are explicitly inapplicable to the Registry service:

* *Check 10* (principal_token validation) — the Registry has no human principal and MUST NOT carry a principal delegation chain.

* *Check 4* (duplicate AID rejection) — genesis is idempotent; if a registry_id already exists in the data store the existing record MUST be used and genesis MUST NOT overwrite it.

7.3.4. Well-Known Publication

The Registry MUST publish discovery metadata at:

GET /.well-known/aip-registry

The response MUST be a JSON object containing:

Field	Type	Required	Description
registry_id	string	REQUIRED	Stable HTTPS identifier for the Registry service
registry_name	string	REQUIRED	Human-readable Registry name (maxLength: 128)
aip_version	string	REQUIRED	The aip_version this Registry conforms to
registry_trust_uri	string	REQUIRED	URI of the current Registry Trust Record
endpoints	object	REQUIRED	Map of service names to relative or absolute URI paths

Table 13

The _endpoints_ object MUST include at minimum:

Key	Description
agents	Base path for agent endpoints (e.g., /v1/agents)
crl	CRL endpoint path (e.g., /v1/crl)
revocations	Revocation submission path (e.g., /v1/revocations)

Table 14

First-contact bootstrapping: On first contact with a Registry, Relying Parties MUST:

* Fetch /.well-known/aip-registry over HTTPS.

* Verify that the response registry_id matches the Registry URI established via the root principal DID Document when such a DID binding is available. Unless otherwise specified, this comparison MUST use origin comparison per [RFC6454] (scheme + host + port).

* Fetch the Registry Trust Record from registry_trust_uri.

* Pin the registry_id, the trusted Registry Trust Record version, and the trust keys from that record locally.

* On subsequent contacts, use the versioned trust-update procedure in Section 7.3.6 before updating the local trust state.

The registry_id identifies the Registry instance and MUST remain stable across planned Registry key rotation. Planned key rotation updates the Registry Trust Record and the Registry's active verification keys while preserving the same registry_id. Emergency compromise recovery is handled separately under Section 7.3.6.

The well-known document is discovery metadata. The canonical trust state for a Registry is defined by its Registry Trust Record, not by the well-known document itself.

7.3.5. Registry Trust Record

A Registry Trust Record is the canonical versioned trust metadata object for a Registry. It MUST be published at:

Method	Path	Description
GET	/v1/registry-trust/current	Current Registry Trust Record
GET	/v1/registry-trust/{version}	Immutable Registry Trust Record for the specified version

Table 15

The Registry Trust Record MUST use a detached-signature structure consisting of a signed object and a signatures array. The signed object MUST include at minimum:

* registry_id

* version

* issued_at

* expires_at

* discovery_uri

* endpoints

* trust_signature_threshold

* trusted_keys

* active_verification_keys

The canonical signing input is the RFC 8785 JCS serialisation of the signed object only. Signature verification MUST ignore the signatures array except as the container for detached signatures over that signed object.

Verifiers use the latest trusted Registry Trust Record for current Registry interactions. For historical verification of Step Execution Tokens, CRLs, or signed notifications, verifiers MUST use the Registry Trust Record version that was current at artifact issuance time. The trusted_keys set governs trust-record acceptance, while active_verification_keys governs which runtime keys are valid for Registry-signed artifacts issued under that trust-record version.

Historical Registry Trust Records MUST remain retrievable for at least the maximum lifetime of Step Execution Tokens and CRL artifacts plus a 30-day audit margin.

7.3.6. Single-Instance Constraint

Each Registry deployment MUST have exactly one active Registry ID. Provisioning a second Registry ID within the same deployment MUST be treated as a fatal configuration error. Horizontal scaling and high-availability deployments MUST share a single Registry ID and trust state.

7.3.7. Registry Key Rotation

AIP defines two Registry key rotation paths:

* *Planned rotation:* continuity-preserving rotation in which the Registry keeps the same registry_id and publishes a new Registry Trust Record version.

* *Emergency re-bootstrap:* continuity-breaking recovery for suspected key compromise or loss of the retiring private key.

Validators bootstrapping trust in a Registry MUST fetch and pin the current Registry Trust Record before processing any Step Execution Tokens or CRL documents signed by that Registry. Cached trust state MUST NOT be used beyond the trust record's expires_at without refresh.

7.3.7.1. Planned Rotation

During planned rotation, the Registry MUST publish a new immutable Registry Trust Record whose signed.version is exactly one greater than the previously trusted version and whose signed.registry_id matches the existing pinned registry_id.

The signed.trust_signature_threshold value defines the minimum number of signatures from signed.trusted_keys required for ordinary acceptance of a Registry Trust Record. Implementations MAY require a higher threshold than 1 for high-assurance deployments, but they MUST still preserve the overlapping-signature rule below during planned rotation.

Planned rotation MUST use overlapping trust signatures. A new Registry Trust Record version MUST satisfy the threshold rules in both the currently pinned and the candidate new trust record. At a minimum, a planned rotation update MUST be signed by:

1. at least one key trusted in the currently pinned Registry Trust Record; and

2. at least one key trusted in the new Registry Trust Record.

If the discovery document's registry_trust_uri does not share the same origin as registry_id, the Relying Party MUST treat the Registry as untrusted unless an explicit trust policy permits that alternate trust-record origin.

A Relying Party that has pinned trust version N MUST update sequentially. It MUST fetch and validate version N+1 before accepting version N+2 or later.

This sequential trust-update model is intentionally similar to repository root update patterns in The Update Framework [TUF], where clients advance trust state one version at a time to resist rollback and mix-and-match attacks.

1. The fetched record's signed.registry_id MUST match the pinned value.

2. The fetched record’s signed.version MUST equal the pinned version plus exactly 1.

3. The fetched record’s signatures MUST satisfy the overlapping-signature rule above.

4. The fetched record’s expires_at MUST be in the future.

5. The fetched record MUST NOT be older than or equal to any previously accepted version for that registry_id.

If all checks succeed, the Relying Party MAY replace its pinned trust state with the new Registry Trust Record while retaining the prior trust records for historical verification.

Historical Step Execution Tokens and CRL documents signed before the rotation remain valid for their stated _exp_ period only. Relying Parties MUST verify such historical artifacts against the Registry Trust Record version that was current at issuance time; they MUST NOT re-verify them against the newest trust record automatically.

7.3.7.2. Emergency Re-Bootstrap

If a Registry Trust Record update cannot satisfy the planned rotation checks, or if the retiring trust key is suspected compromised or unavailable, the event MUST be treated as an emergency re-bootstrap.

This distinction between planned rollover and emergency continuity-breaking recovery is similar in spirit to DNSSEC trust anchor rollover guidance in [RFC5011], where automated trust continuity and exceptional recovery are treated as different operational cases.

1. Relying Parties that have pinned Registry trust state and subsequently receive discovery or trust metadata that does not satisfy the planned-rotation checks above MUST treat this as a potential MITM condition.

2. Relying Parties MUST NOT use the new trust state automatically. They MUST halt Registry-dependent operations and require explicit re-bootstrap or out-of-band operator confirmation before re-pinning.

3. Emergency re-bootstrap is continuity-breaking by design. Automatic verifier acceptance rules for planned rotation do not apply.

4. Historical Step Execution Tokens and CRL documents signed by the previous Registry trust state remain valid for their stated _exp_ period only. Relying Parties MUST verify historical artifacts against the trust record version that was current at issuance time; they MUST NOT re-verify them against the emergency replacement trust state.

8. Credential Tokens

8.1. Token Structure

An AIP Credential Token is transmitted as an HTTP Authorization header:

EXAMPLE (informative):

```
Authorization: AIP <token>
X-AIP-Version: 0.3
```

For interactions requiring Proof-of-Possession:

EXAMPLE (informative):

```
Authorization: AIP <token>
DPoP: <dpop-proof>
X-AIP-Version: 0.3
```

The X-AIP-Version HTTP header MUST carry the full protocol version string, identical to the aip_version JWT claim . For this specification, the value MUST be "0.3". Implementations MUST reject requests where the X-AIP-Version header carries a version value that does not match a supported aip_version.

8.2. Token Issuance

Implementations MUST enforce the following maximum token lifetimes:

Scope Category	Maximum TTL
All standard scopes (email.*, calendar.*, web.*, etc.)	3600 seconds (1 hour)
transactions.* or communicate.*	300 seconds (5 minutes)
filesystem.execute	300 seconds (5 minutes)
spawn_agents.create or spawn_agents.manage	300 seconds (5 minutes)

Table 16

When a token contains scopes from multiple categories, the most restrictive TTL applies. Zero-duration and negative-duration tokens (where exp <= iat) MUST be rejected.

REMINDER: A token's Tier is determined by its highest-risk scope (Section 3.1). A token containing one Tier 2 scope and any number of Tier 1 scopes is a Tier 2 token in its entirety. Implementations MUST NOT derive Tier from the majority of scopes or from the first scope in the array.

8.3. Token Refresh and Long-Running Tasks

8.3.1. Agent Self-Refresh

An AIP agent holds its own Ed25519 private key and MAY issue new Credential Tokens at any time, provided the Principal Token(s) in its delegation chain (aip_chain) remain valid. There is no refresh token in AIP - the agent's signing key IS the refresh credential.

An agent self-refresh involves: issuing a new Credential Token with a new jti, a fresh iat, and a new exp within TTL limits. The aip_chain content remains unchanged until the Principal Tokens within it expire.

Agents MUST NOT re-use the same jti when issuing a fresh token. Each issued Credential Token MUST have a unique jti.

8.3.2. Pre-emptive Refresh Requirements

Agents MUST implement pre-emptive refresh to avoid mid-task token expiry. An agent MUST begin issuing a replacement Credential Token before the current token expires:

Token TTL Category	Begin refresh when remaining TTL <=
Standard (3600s)	300 seconds (5 minutes)
Sensitive (300s)	30 seconds

Table 17

Implementations MUST NOT wait for a token rejection (token_expired) before refreshing. Waiting for rejection creates a gap in execution continuity and may leave in-progress Tier 2 operations without valid authority.

Relying Parties MUST NOT reject a token solely because a newer token exists for the same agent. Each token is independently valid for its own iat to exp window.

For real-time streaming interactions (e.g., a long-running web socket), the agent SHOULD renegotiate the session with a fresh Credential Token before the current token's expiry rather than waiting for mid-stream rejection.

8.3.3. Delegation Chain Expiry

When a Principal Token in the aip_chain expires, the Credential Token becomes structurally invalid at validation Step 8h regardless of the Credential Token's own exp. This is because the delegation authority itself has lapsed.

When a delegation chain expires:

1 The agent MUST NOT issue new Credential Tokens referencing the expired aip_chain (even if the agent's own exp is still in the future).

2 The agent MUST obtain a fresh delegation from its parent (or from the root principal for depth-0 agents) via the AIP-GRANT flow or sub-agent delegation flow.

3 Once a fresh delegation is established and registered, the agent may resume issuing Credential Tokens.

Relying Parties that receive a token where Step 8h fails MUST return chain_token_expired. Agents receiving this error MUST treat it as delegation_chain_refresh_required - they must re-establish their delegation rather than merely refreshing their Credential Token.

Anticipatory chain refresh: Agents SHOULD monitor the expires_at timestamps of all Principal Tokens in their aip_chain. When the nearest expiry is within 10% of the total delegation validity period (or 24 hours, whichever is smaller), the agent SHOULD proactively initiate a delegation renewal.

8.3.4. Interaction with Approval Envelopes

Approval Envelopes are specifically designed to decouple human approval timing from token TTL constraints. The following rules govern their interaction:

1 An Approval Envelope's approval_window_expires_at is independent of any Credential Token TTL. Envelopes may remain in pending_approval status for hours while normal TTLs of 300s or 3600s apply only at execution time.

2 When an agent claims an Approval Step, it MUST present a Credential Token that is valid at the time of the claim. The token TTL for a step-claim follows the same rules as for any other interaction involving those scopes (300s for Tier 2 scopes, 3600s for Tier 1).

3 Long-running workflows where steps are separated by hours or days require the agent to issue a fresh Credential Token for each step claim. This is intentional - the agent's authority must be re-verified at each step, not just at envelope creation time.

4 If an agent's delegation chain expires between envelope approval and step execution, the agent MUST renew its delegation before claiming any remaining steps. The Approval Envelope itself remains valid - only the execution credential needs renewal.

5 An agent MUST NOT pre-issue step-claim Credential Tokens for all steps at envelope approval time. Tokens MUST be issued at execution time so that revocation checks (Section 9 Step 7) are performed against the current Registry state.

8.4. Token Exchange for MCP

8.4.1. Overview

An AIP agent MAY exchange its Credential Token for a scoped access token targeting a specific MCP server or OAuth-protected resource. The exchange is performed at the Registry’s token endpoint .

8.4.2. Exchange Request

The agent sends an RFC 8693 token exchange request:

```
POST /v1/oauth/token HTTP/1.1
Content-Type: application/x-www-form-urlencoded
DPoP: <DPoP proof JWT>

grant_type=urn:ietf:params:oauth:grant-type:token-exchange
&subject_token=<AIP Credential Token>
&subject_token_type=urn:ietf:params:oauth:token-type:jwt
&resource=https://mcp-server.example.com/
&scope=urn:aip:scope:email.read urn:aip:scope:calendar.read
```

Normative requirements:

1. grant_type MUST be urn:ietf:params:oauth:grant-type:token-exchange.
2. subject_token MUST be a valid AIP Credential Token.
3. subject_token_type MUST be urn:ietf:params:oauth:token-type:jwt.
4. resource MUST be present per RFC 8707 [RFC8707], identifying the target resource server.
5. scope MUST use AIP scope URIs from the urn:aip:scope: namespace. The requested scopes MUST be a subset of the Credential Token’s aip_scope.
6. DPoP proof [RFC9449] MUST be included, binding the exchange to the agent’s key material.

8.4.3. Exchange Validation

The Registry (as OAuth AS) MUST:

1. Validate the subject_token using the full validation algorithm.
2. Verify the DPoP proof binds to the agent’s public key.

3 Verify the requested scope is a subset of the subject_token's aip_scope (attenuation only, never expansion).

4 Verify the resource is a registered MCP server or resource. Fail: invalid_target.

5 Issue an access token with:

* aud set to the resource URI

* scope set to the granted scopes

* sub set to the agent's AID

* act.sub set to the root principal's DID (actor chain per [RFC8693] § 4.1)

* TTL <= the remaining TTL of the subject_token

6 The access token MUST be a JWT per RFC 9068 [RFC9068].

8.4.4. Exchange Response

EXAMPLE (informative):

```
 {
   "access_token": "<JWT>",
   "token_type": "DPoP",
   "expires_in": 300,
   "scope": "urn:aip:scope:email.read urn:aip:scope:calendar.read",
"issued_token_type": "urn:ietf:params:oauth:token-type:access_token"
 }
```

9. Credential Token Validation

The Credential Token Validation Algorithm is the normative heart of AIP. A Relying Party MUST execute the following steps in the order presented. Validation is deterministic: two independent implementations executing the same steps on the same token with the same Registry state MUST reach the same outcome.

The Relying Party MUST reject the token at the first step that fails. Steps marked 6a, 6b, 9e, and 10a are conditional sub-steps introduced in v0.3; they are part of the numbered step at which they appear and do not change the base step numbering.

9.1. Step 1: Parse

Parse the Authorization header value as a JWT per [RFC7519]. If parsing fails or the token is malformed, reject with invalid_token.

9.2. Step 2: Header Validation

Verify the JWT header contains:

* typ: MUST be "AIP+JWT"

* alg: MUST be one of the approved algorithms per Section 21.2 (EdDSA REQUIRED; ES256 and RS256 optional per Section 20.1)

* kid: MUST be present and MUST be a DID URL in the form did:aip:<namespace>:<32-hex>#key-<n>

If any check fails, reject with invalid_token.

9.3. Step 3: Identify Agent and Retrieve Public Key

Extract the kid header parameter. This is a full DID URL identifying the agent's public key. Contact the Registry to retrieve the historical public key matching this kid. The Registry MUST return the public key material along with its validity period (valid_from and valid_until timestamps).

If the kid is not found or the key is not valid at the time of the JWT's iat claim, reject with unknown_aid.

9.4. Step 4: Verify Signature

Verify the JWT signature using the public key retrieved in Step 3. The signature verification MUST use constant-time comparison. If signature verification fails, reject with invalid_token.

9.5. Step 5: Validate Claims

Validate the following claims from the decoded JWT payload:

5a. iat (Issued-At Time): iat MUST NOT be in the future. Allow a 30-second clock skew tolerance. If iat is in the future (beyond skew), reject with invalid_token.

5b. exp (Expiration Time): exp MUST be strictly greater than iat (zero-duration tokens are not permitted). If exp <= iat, reject with invalid_token.

5c. Token Not Expired: exp MUST be in the future (current time must be < exp). If exp is in the past, reject with token_expired.

5d. aud (Audience): The aud claim MUST match the Relying Party's identifier. If aud is a string, it MUST match exactly. If aud is an array, the Relying Party's identifier MUST be present in the array. If no match, reject with invalid_token.

5e. jti (JWT ID) Replay Check: jti MUST be a UUID v4 in canonical form (lowercase, hyphenated). The Relying Party MUST maintain a replay cache keyed by (iss, jti) for each token's validity window. If (iss, jti) has been seen before, reject with token_replayed. After the token expires, the replay cache entry MAY be discarded.

5f. aip_version: aip_version MUST be present and MUST be "0.3" for tokens conforming to this specification. If aip_version is absent or unrecognised, reject with invalid_token and include the received aip_version in the error description to aid debugging.

9.6. Step 6: TTL Validation

Verify that the token's lifetime does not exceed the maximum permitted by its scopes. Compute lifetime = exp - iat (in seconds). Compare against the TTL limits defined in Section 8.2:

```
Standard scopes (email.*, calendar.*, web.*):     max 3600 seconds
Sensitive scopes (transactions.*, communicate.*,
                  filesystem.execute,
                  spawn_agents.create/manage):    max 300 seconds
```

When a token contains scopes from multiple categories, the most restrictive limit applies.

If lifetime exceeds the limit, reject with invalid_token.

REMINDER: A token's Tier is determined by its highest-risk scope (Section 3.1). A token containing one Tier 2 scope and nine Tier 1 scopes is a Tier 2 token in its entirety. Do not derive Tier from the majority or first scope.

9.7. Step 6a: Registry Trust Anchoring (Conditional)

This step is REQUIRED for Tier 2 operations and RECOMMENDED for Tier 1. It verifies that the Registry from which the Relying Party has been fetching data matches the Registry declared by the principal's DID Document.

1. Extract principal_id from aip_chain[0].iss (the root principal's DID). This is the authorising human or organisational principal.

2. Resolve principal_id using the DID method's own resolution mechanism (e.g., did:web resolution per [W3C-DID], did:key per [W3C-DID]). The resolution MUST be independent of any agent-provided data. Use the DID method's canonical resolver.

3. Examine the resolved DID Document's service array. Locate an entry with type: "AIPRegistry".

4. Extract the serviceEndpoint URI from that service entry. This is the authoritative Registry URI for this principal.

5. Verify that the Registry from which the Relying Party has been fetching revocation status, Capability Manifests, and step data matches the declared serviceEndpoint URI. Perform origin comparison per [RFC6454] (scheme + host + port must match).

6. If aip_registry is present in the Credential Token, verify it matches the DID-Document-declared Registry endpoint. If mismatch, reject with registry_untrusted.

7. If the DID Document does not contain an AIPRegistry service entry and the token contains Tier 2 scopes, reject with registry_untrusted.

For Tier 1 operations: this step is RECOMMENDED. Registry trust anchoring prevents MitM Registry substitution but adds latency (additional DID resolution). Tier 1 operators MAY skip this step if they accept the risk that the Registry might be MitM'd with a loss of up to 15-minute revocation staleness.

If DID resolution fails and the token contains Tier 2 scopes, reject with registry_unavailable.

9.8. Step 6b: Engagement Validation (Conditional)

This step applies only if aip_engagement_id is present in the Credential Token payload.

1. Fetch the Engagement Object from the Registry via GET /v1/engagements/{aip_engagement_id}.

2. Verify the Engagement's status field. If "active", continue. If "completed" or "terminated", reject with engagement_terminated. If "suspended", reject with engagement_suspended.

3. Verify that the token's sub (agent AID) appears in the Engagement's participants array as an active participant. If sub is not in the participants array or has been marked as removed, reject with engagement_participant_removed.

4. If the Engagement defines approval_gates with pending gates that guard the current operation, verify the token's operation is not gated (or that the gate has been approved). If a required gate is still pending, reject with engagement_gate_pending.

9.9. Step 7: Revocation Check

Query the Registry for revocation status of the agent identified by iss (extracted from the kid). The revocation check method depends on the token's Tier:

* *Tier 1:* Retrieve the CRL from the Registry cache. If the agent appears on the CRL with revocation type full_revoke or principal_revoke, reject with agent_revoked. If the agent appears with scope_revoke, the Relying Party MUST verify that none of the scopes in the token's aip_scope are present in the scopes_revoked list. If any match, reject with agent_revoked.

* *Tier 2:* Perform a live Registry lookup. Query GET /v1/agents/{iss}/revocation and verify no active revocation applies to the token's AID or its currently requested scopes. If revoked, reject with agent_revoked. If the Registry is unreachable, reject with registry_unavailable.

The Relying Party MUST verify the Registry trust state used for this revocation check using the Registry Trust Record procedure in Section 7.3.6 before treating Registry responses as authoritative.

9.10. Step 8: Delegation Chain Validation

Validate the Principal Token delegation chain in aip_chain. This array contains one or more compact-serialised JWTs, each conforming to the Principal Token schema.

For each Principal Token at index i (from 0 to n-1 where n is the length of aip_chain):

8a. Valid JWT: The token at index i MUST be a valid, well-formed JWT conforming to the Principal Token schema. If parsing or schema validation fails, reject with delegation_chain_invalid.

8b. delegation_depth Matches Index: The delegation_depth claim in

token i MUST equal exactly i. (The array is 0-indexed; aip_chain[0] has delegation_depth: 0, etc.) If mismatch, reject with invalid_delegation_depth.

8c. Delegation Depth Does Not Exceed Maximum: The delegation_depth of token i MUST NOT exceed the max_delegation_depth value declared in token 0 (the root token). If max_delegation_depth is absent from token 0, the default is 3. If i exceeds this limit, reject with invalid_delegation_depth.

8d. Signature Verification: For token at index i, extract the iss claim: For i = 0, iss MUST equal principal.id. Verify signature. For i > 0, iss MUST equal delegated_by. Verify signature. If verification fails, reject with delegation_chain_invalid.

8e. Delegation Chain Linkage: For token at index i > 0, verify that delegated_by[i] equals sub[i-1] (the sub of the previous token). This ensures the chain is continuous: each agent is delegated by the previous agent in the chain. If mismatch, reject with delegation_chain_invalid.

8f. Agent Revocation: For each sub AID in the chain (at all indices), verify it is not revoked using the same Tier-specific revocation check as Step 7. If any agent in the chain is revoked, reject with agent_revoked.

8g. No Duplicate AIDs: No AID may appear more than once in the chain (no cycles or duplicates). If an AID appears at indices i and j with i != j, reject with delegation_chain_invalid.

8h. Token Validity: For token at index i, verify expires_at > issued_at and expires_at is in the future. If either check fails, reject with chain_token_expired.

8i. Consistent Principal: The principal.id field MUST be byte-for-byte identical across ALL elements in the chain (index 0 through n-1). This ensures the chain always traces to the same root principal. If any element has a different principal.id, reject with delegation_chain_invalid.

8j. Principal is Not an Agent: The principal.id from aip_chain[0] MUST NOT begin with did:aip:. Principals are humans or organisations, never agents. If principal.id uses the did:aip method, reject with delegation_chain_invalid.

9.10.1. Step 8 Post-Check A

After validating all elements in the chain, verify that iss (the issuer of the Credential Token, from the JWT header kid) MUST equal aip_chain[n-1].sub (the sub of the last element in the chain, i.e., the acting agent's AID). This confirms that the agent issuing the token is the leaf of the delegation chain. If mismatch, reject with delegation_chain_invalid.

9.10.2. Step 8 Post-Check B

For single-element chains (n = 1), verify that iss MUST equal sub. This confirms a non-delegated token issued directly by the principal's agent. If mismatch, reject with delegation_chain_invalid.

9.11. Step 9: Capability Validation

Verify that the agent's requested scopes are permitted by its Capability Manifest.

1. Fetch the Capability Manifest for the agent identified by iss from the Registry. If unavailable, reject with manifest_invalid.

2. Verify the manifest signature. If verification fails, reject with manifest_invalid.

3. Verify expires_at is in the future. If expired, reject with manifest_expired.

4. For delegated agents: verify scope inheritance. For each scope, verify chain has permission.

9.12. Step 9a: Scope Verification

For all agents: verify each scope in aip_scope is present in the Capability Manifest.

9.13. Step 9b: Capability Overlay (Conditional)

If a Capability Overlay exists, verify scope constraints permit the requested operation.

9.14. Step 10: DPoP Validation (Conditional)

REMINDER: A token's Tier is determined by its highest-risk scope (Section 3.1). A token containing one Tier 2 scope is a Tier 2 token.

If aip_scope contains any of the following scopes, DPoP (Demonstration of Proof-of-Possession) MUST be verified:

* transactions or any scope with prefix transactions.

* communicate.whatsapp, communicate.telegram, communicate.sms, communicate.voice

* filesystem.execute

* spawn_agents.create, spawn_agents.manage

If any of these scopes is present, the HTTP request MUST include a DPoP header containing a Demonstration of Proof-of-Possession proof JWT per [RFC9449]. Verify:

1. The DPoP proof is a valid JWT.

2. The htm (HTTP method) claim matches the HTTP method of the request.

3. The htu (HTTP URI) claim matches the request URI.

4. The jti has not been seen before (DPoP-specific replay cache, separate from Credential Token jti cache, keyed by (kid, jti)).

5. The public key in the jwk claim matches the agent’s key material (used to issue the Credential Token).

If DPoP validation fails at any step, reject with dpop_proof_required (if proof is missing) or invalid_token (if proof is malformed or invalid).

9.15. Step 10a: Approval Envelope Step Verification (Conditional)

This step applies only if both aip_approval_id and aip_approval_step are present in the Credential Token (the Step Execution Token case, Section 13.8).

The Relying Party MUST call the Registry endpoint GET /v1/approvals/{aip_approval_id}/steps/{n} where {n} is the value of aip_approval_step verbatim (an integer >= 1, per Section 13.4; MUST NOT be decremented or adjusted). NOTE: aip_approval_step uses 1-based indexing (e.g., the first step in an envelope is step 1). This is a change from deprecated 0-indexed drafts; implementations MUST NOT subtract 1 from the value. Verify:

1. *Step Status:* The step’s status field MUST be "claimed" (not "pending", "completed", etc.). If status is "pending", the step has not been claimed and cannot execute. Reject with approval_step_invalid.

2. *Actor Match:* The step’s actor field MUST equal the Credential Token’s sub (the agent AID). Reject if mismatch with approval_step_invalid.

3. *Relying Party Match:* The step’s relying_party_uri MUST match the host of the current HTTP request (origin comparison per [RFC6454]: scheme + host + port). Reject if mismatch with approval_step_invalid.

4. *Action Hash:* Compute the expected action hash for this step per Section 13.7. Compare against the step’s action_hash field. If mismatch, reject with approval_step_invalid. This ensures the approving principal authorised the exact action being executed.

9.16. Step 11: Tier 3 Enterprise Checks (Conditional)

This step applies only to Tier 3 enterprise deployments, which are declared and documented in the Registry’s /.well-known/aip-registry endpoint.

For Tier 3 operations:

1. *mTLS Client Certificate:* The HTTP connection MUST use mutual TLS. Verify that the client certificate’s subject DN maps to the agent’s AID (iss). If mismatch or certificate is absent, reject with invalid_token.

2. *OCSP Revocation Check:* Perform an OCSP check per [RFC6960] on the client certificate to verify it has not been revoked at the transport layer. If the certificate is revoked, reject with agent_revoked.

If either check fails, reject with the appropriate error code (invalid_token or agent_revoked).

9.17. Step 12: Accept

If all preceding steps pass without rejection, the Relying Party MUST accept the token and grant the requested access.

10. Delegation

AIP enables hierarchical delegation where a principal authorizes a primary agent, which may in turn authorize sub-agents under narrowing scope constraints. Delegation is encoded in the aip_chain array of the Credential Token, with each Principal Token in the chain representing one delegation hop from principal to agent.

10.1. Delegation Chain

Every Credential Token MUST include a verifiable principal chain linking the acting agent to its root principal via the aip_chain array. The root principal MUST be a human or organizational entity identified by a W3C Decentralized Identifier (DID) that does NOT use the did:aip method. For example: did:web, did:key, or proprietary DID methods are acceptable.

The maximum delegation depth is a hard constraint of 10 levels. This means an agent may not delegate to a sub-agent if doing so would create a chain longer than 10 Principal Tokens (0-indexed from 0 to 9, or 1-indexed as depths 1 to 10).

The default value of max_delegation_depth MUST NOT exceed 3. When a Principal Token does not explicitly set max_delegation_depth, implementations MUST treat the default as 3. This conservative default prevents accidental authorization chains from becoming unmanageably deep; parties requiring deeper chains MUST explicitly opt in by setting max_delegation_depth to a value between 3 and 10.

10.2. Capability Scope Rules

Scope inheritance is the mechanism by which child agents are constrained to operate within the bounds of their parent’s authorization. Five core rules (D-1 through D-5) govern this relationship; they are defined in Section 5.10 of the AIP specification.

Scope Inheritance Rule: For each scope s granted to a child agent, s MUST be present in the parent’s Capability Manifest AND all constraint values for s in the child’s manifest MUST be no more permissive (<=) than the corresponding values in the parent’s manifest.

For numeric limits (e.g., max_single_transaction, max_daily_value), the child value MUST be <= the parent value. For boolean enables or enum values, the child constraint MUST NOT be more permissive than the parent (e.g., child cannot set a flag true if parent sets it false).

Implementations MUST enforce this rule at delegation time -- when a child agent is registered, the Registry MUST validate that all scopes in its Capability Manifest satisfy the inheritance rule relative to its immediate parent's manifest. Implementations MAY reject delegation requests that violate this rule before they are registered.

Relying Parties MUST independently verify this rule during validation at Step 9c of the Credential Token validation algorithm (see Section 9.11). This ensures that even if a Registry incorrectly permits a violating delegation, Relying Parties will catch it and reject the token.

10.3. Delegation Validation

When validating a delegated Credential Token, Relying Parties must fetch and verify the Capability Manifests of all agents in the delegation chain. This section specifies the performance and caching constraints for these operations.

Ancestor Manifest Fetch Limits: Implementations MUST NOT fetch more ancestor manifests than the max_delegation_depth value of the chain's root token (which defaults to 3 when absent). This limit prevents accidental O(n^2) fetch patterns in deep delegation chains and bounds the performance cost of validation.

Ancestor Manifest Caching for Tier 1: For Tier 1 operations (low-risk scopes with bounded-staleness threat model), ancestor manifests MAY be cached for a maximum of 60 seconds. This cache is per-agent and per-manifest, and MUST respect the 60-second TTL. After 60 seconds, a fresh fetch is required.

No-Cache Requirement for Tier 2: For Tier 2 operations (high-risk scopes with real-time threat model), the no-cache requirement in the Credential Token validation algorithm (see Section 9.11) applies to ALL manifests in the delegation chain -- including ancestor manifests -- not only the leaf agent's manifest. The 60-second ancestor cache MUST NOT be used for any manifest appearing in a Tier 2 validation. Every manifest must be fetched fresh from the Registry.

Unavailable Manifests: If an ancestor manifest is unavailable or cannot be fetched (due to network failure, Registry downtime, or the manifest having been deleted), the Relying Party MUST reject the token by returning the error code manifest_invalid. Partial delegation chains are not acceptable; either all manifests in the chain are available and valid, or the token is rejected.

11. Revocation Management

Revocation provides the kill switch for agent identity. When an agent is revoked, all its Credential Tokens become invalid and the agent can no longer act on behalf of any principal.

11.1. Revocation Object

Revocation is performed by submitting a signed Revocation Object to POST /v1/revocations.

Type	Description
full_revoke	Permanently revokes the AID.
scope_revoke	Invalidates specific scopes for the agent. For Tier 1 and 2, this MUST cause Relying Parties to reject Credential Tokens containing the revoked scopes (Step 7, Section 9).
delegation_revoke	Invalidates delegation chains rooted at target. Child agents lose authority.
principal_revoke	Issued by the root principal to revoke their entire authorisation of the target agent (via AID) or all agents under their authority (via Principal DID).

Table 18: Revocation Types

11.2. Certificate Revocation List (CRL)

The Registry MUST expose a CRL at GET /v1/crl. The CRL MUST be updated within 15 minutes of a new Revocation Object being accepted. The CRL endpoint MUST be served from a CDN or distributed infrastructure. CRL documents MUST be signed by a key referenced in active_verification_keys.crl of the Registry Trust Record version current at issuance time.

11.3. Revocation Checking

A token’s Tier is determined by its highest-risk scope. A token with one Tier 2 scope and any number of Tier 1 scopes is a Tier 2 token.

Tier 1 - Bounded-staleness: TTL is 3600 seconds. Validate against CRL at issuance time. CRL MUST be refreshed every 15 minutes.

Tier 2 - Real-time revocation: For scopes: transactions.*, communicate.*, filesystem.execute, spawn_agents.create, spawn_agents.manage. TTL is 300 seconds. Real-time Registry check on EVERY request. MUST NOT cache revocation status. DPoP MUST be verified. If Registry unreachable: MUST deny and return registry_unavailable.

Tier 3 - Enterprise: MUST use mTLS. MUST support OCSP per RFC 6960. Tier 3 supplements, not replaces, Tier 2.

Child Agent Propagation: When the Registry processes a Revocation Object with propagate_to_children: true, the Registry MUST recursively revoke descendants within 15 seconds.

Replica Registries MUST synchronise within 45 seconds. Combined end-to-end propagation MUST NOT exceed 60 seconds.

Approval Envelope interaction: When an agent AID is revoked, the Registry MUST transition all Approval Envelopes in pending_approval, approved, or executing status to failed. Unclaimed steps for that actor MUST be marked failed. In-progress claims MUST be treated as failed; the Registry MUST initiate compensation if applicable.

11.4. Registry Push Notification Protocol (RPNP)

RPNP is an OPTIONAL Registry capability for real-time revocation event delivery.

11.4.1. Overview

When RPNP is implemented:

* Registry MUST deliver push events within 5 seconds
* Push payloads MUST be signed by a key referenced in active_verification_keys.notifications of the Registry Trust Record version current at issuance time
* Subscriber authentication MUST be verified at subscription time

For subscribing Relying Parties, the effective revocation window is the RPNP delivery latency (at most 5 seconds) rather than the CRL refresh interval. RPNP does not replace CRL; it supplements it.

11.4.2. Subscription

A Relying Party subscribes by calling POST /v1/subscriptions:

Field	Required	Constraints
subscriber_did	REQUIRED	MUST be did:web or did:aip
event_types	REQUIRED	full_revoke, scope_revoke, delegation_revoke, etc.
scope_filter	REQUIRED	aid, principal, or all
targets	CONDITIONAL	REQUIRED when scope_filter is aid or principal
webhook_uri	REQUIRED	HTTPS URI
hmac_secret_hash	REQUIRED	SHA-256 hash of shared secret
subscription_expires_at	REQUIRED	ISO 8601 UTC; max 90 days

Table 19: Subscription Fields

The subscription request MUST be authenticated via DPoP proof. Unsigned requests MUST be rejected with subscription_auth_required.

11.4.3. Push Event Payload

Push events are delivered as HTTP POST to the subscriber's webhook_uri. The body is a compact-serialised JWT signed by a key referenced in active_verification_keys.notifications of the Registry Trust Record version current at issuance time.

Header Field	Value
typ	AIP-RPNP+JWT
alg	EdDSA
kid	Registry notification key ID

Table 20: RPNP JWT Fields

Payload Field	Description
iss	Registry ID
sub	Affected AID or engagement ID
iat	Unix timestamp
jti	UUID v4; unique event ID
event_type	One of subscribed types
event_data	Event-specific data

Table 21: RPNP Payload Fields

The request MUST include an X-AIP-Signature header containing HMAC-SHA256(shared_secret, body). The subscriber MUST verify both the JWT signature and the HMAC.

11.4.4. Delivery Guarantees

1. Registry MUST deliver push events within 5 seconds
2. On delivery failure: retry with exponential backoff (1s, 2s, 4s - 3 attempts minimum)
3. After 3 consecutive failures: mark subscription as degraded. Subscriber falls back to CRL.
4. Subscribers MUST reject duplicate jti within 60 seconds

12. Principal Grant Ceremony (AIP-GRANT)

The AIP-GRANT ceremony provides a standardised protocol for principals to authorise AI agents. AIP-GRANT is analogous to the OAuth 2.0 Authorization Code Flow, adapted for the agent identity use case. Two independent implementations following this section MUST produce interoperable grant interactions.

12.1. Overview and Roles

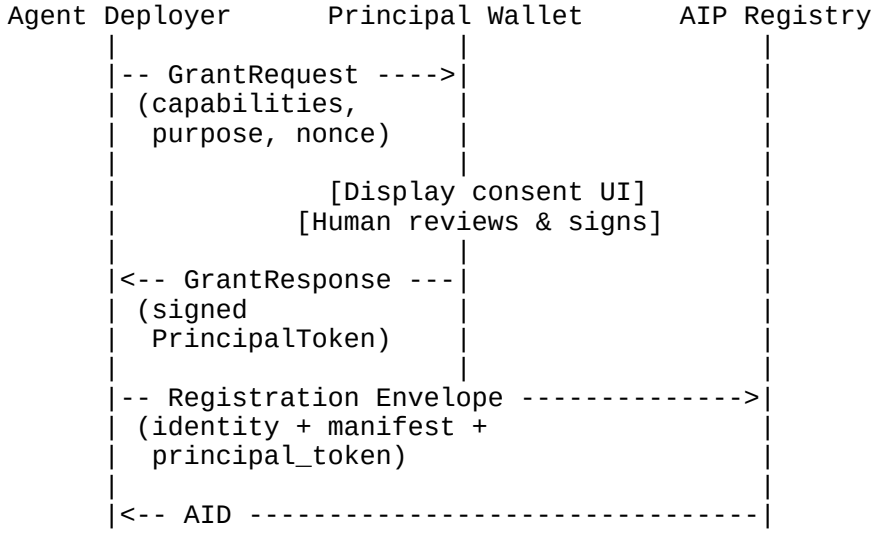
The AIP-GRANT ceremony involves three actors:

Agent Deployer The party that constructs the GrantRequest and submits the Registration Envelope to the Registry. The deployer generates the agent's Ed25519 keypair before initiating the grant and may be the principal themselves or a service acting on the principal's behalf.

Principal Wallet Software that holds the principal's DID private key and executes the signing ceremony. The wallet MUST verify the deployer's signature and MUST obtain explicit human approval before signing.

AIP Registry The service that accepts the Registration Envelope. The Registry's role is defined in Section 17; it is not modified by AIP-GRANT.

The basic flow:



12.2. GrantRequest Object

The GrantRequest is a JSON object constructed by the Agent Deployer. It MUST be signed by the deployer's Ed25519 key when transmitted over the Web Redirect or QR Code bindings. The nonce MUST be cryptographically random and MUST contain at least 128 bits of entropy.

Field	Type	Required	Constraints
grant_request_id	string	REQUIRED	pattern: gr:[0-9a-f]{8}-[0-9a-f]{4}-4[0-9a-f]{3}-[89ab][0-9a-f]{3}-[0-9a-f]{12}
aip_version	string	REQUIRED	MUST be "0.3"
agent_name	string	REQUIRED	maxLength: 64; displayed in consent UI
agent_type	string	REQUIRED	registered namespace value
model.provider	string	REQUIRED	displayed in consent UI
model.model_id	string	REQUIRED	displayed in consent UI
requested_capabilities	object	REQUIRED	Capability Manifest capabilities sub-schema
purpose	string	REQUIRED	minLength: 1; maxLength: 512
delegation_valid_for_seconds	integer	REQUIRED	min: 300; max: 31536000
nonce	string	REQUIRED	minLength: 22; cryptographically random

request_expires_at	string	REQUIRED	ISO 8601 UTC
callback_uri	string	CONDITIONAL	REQUIRED for Web Redirect; MUST use HTTPS

Table 22: GrantRequest Fields

The deployer MUST generate the agent’s AID before constructing the GrantRequest so that the resulting Principal Token correctly names the agent’s AID in its sub field.

12.3. Wallet Consent Requirements

The Principal Wallet MUST implement the following requirements before signing any grant.

Nonce and expiry checks: The Principal Wallet MUST check request_expires_at against the current clock. If expired, it MUST reject with grant_request_expired and MUST NOT show the consent UI. The Principal Wallet MUST maintain a record of seen grant_request_id values for at least 30 days and MUST reject replays with grant_request_replayed.

Mandatory display elements: The consent UI MUST display ALL of the following before presenting the sign/decline choice:

1. Agent name (agent_name) and type (agent_type)
2. AI model - provider and model_id
3. Purpose - the purpose field verbatim
4. Deployer identity - deployer_name and deployer_did
5. All requested capabilities in canonical human-readable strings
6. Delegation validity - expiry computed from delegation_valid_for_seconds
7. *Destructive Operations:* Any scope marked as destructive: true in the Registry’s Scope Map MUST be highlighted in the UI with a mandatory "confirm destructive action" checkbox or equivalent additional friction.

Canonical human-readable capability strings:

Scope	Display String
email.read	Read your email messages and metadata
email.write	Create and draft email messages
email.send	Send email on your behalf
email.delete	Permanently delete your email messages - this cannot be undone
calendar.read	Read your calendar events
calendar.write	Create and update calendar events
calendar.delete	Delete your calendar events
filesystem.read	Read files from your local storage
filesystem.write	Save and modify files on your local storage
filesystem.execute	Execute scripts and commands on your system - HIGH RISK
filesystem.delete	Delete files from your local storage
web.browse	Browse the web and read website content
web.forms_submit	Submit data to web forms
web.download	Download files from the web to your system
transactions	Make financial transactions up to specified limits
communicate.whatsapp	Send and receive messages via WhatsApp
communicate.telegram	Send and receive messages via Telegram
communicate.sms	Send and receive SMS messages
communicate.voice	Initiate and receive voice calls

spawn_agents.create	Create child AI agents on your behalf
spawn_agents.manage	Monitor and manage your existing child agents

Table 23: Canonical Capability Display Strings

12.4. GrantResponse Object

The GrantResponse is constructed and signed by the Principal Wallet after the principal completes the signing ceremony.

Field	Required	Constraints
grant_request_id	REQUIRED	MUST match original request
nonce	REQUIRED	MUST match original nonce
status	REQUIRED	enum: "approved", "rejected", "partial"
principal_id	REQUIRED	W3C DID of signing principal
principal_token	CONDITIONAL	REQUIRED when status is "approved" or "partial"

Table 24: GrantResponse Fields

Deployer validation of GrantResponse: Upon receipt the deployer MUST:

1. Verify grant_request_id matches the original request
2. Verify nonce matches - mismatch MUST be treated as forgery
3. If status is "rejected", halt - no agent is registered
4. Decode and verify the principal_token JWT signature
5. Verify the principal_token payload sub matches the agent’s AID

12.5. Transport Bindings

Implementations MUST support the Web Redirect Flow.

Web Redirect Flow:

```
https://wallet.example.com/aip-grant
  ?request=<base64url-signed-GrantRequest>
  &aip_version=0.3
```

After the principal signs, the Principal Wallet delivers the GrantResponse:

```
POST <callback_uri>
Content-Type: application/json

{GrantResponse JSON}
```

The callback_uri MUST be pre-registered by the deployer. Principal Wallets MUST NOT deliver GrantResponses to unregistered URIs.

12.6. Sub-Agent Delegation Flow

When a parent agent delegates to a child agent, the parent acts as the Principal Wallet. No human consent UI is required.

The parent agent MUST:

1. Verify requested child capabilities are a strict subset of its own Capability Manifest (Rule D-1)

2. Generate a fresh Ed25519 keypair for the child agent

3. Construct and sign the child’s Principal Token

4. Construct the Registration Envelope for the child

5. Submit the Registration Envelope to the Registry

6. Provision the child’s private key through a secure channel

The parent MUST NOT retain the child’s private key after successful provisioning. Retaining the child’s private key enables the parent to forge Credential Tokens in the child’s name and constitutes a violation of the principle of least privilege.

12.7. AIP-GRANT Error Codes

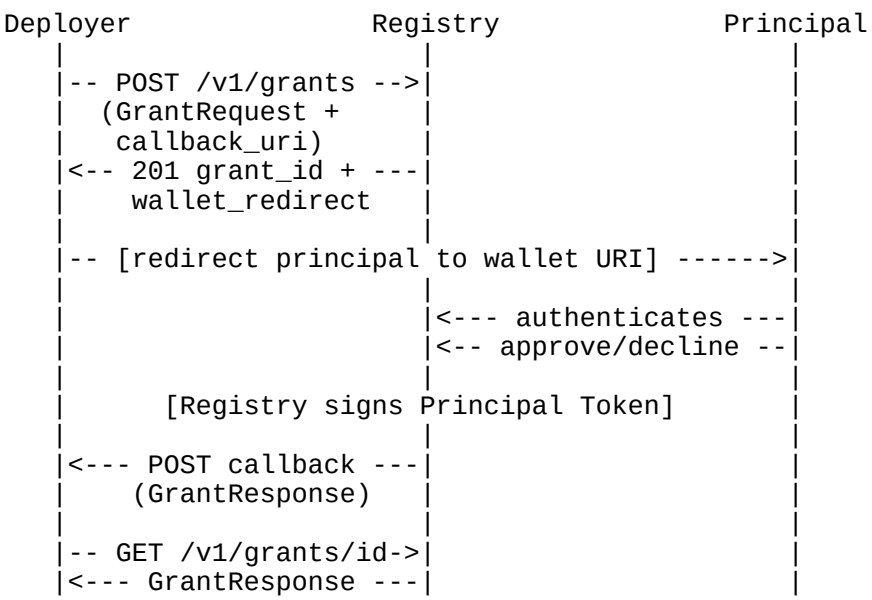
Code	Description
grant_request_expired	request_expires_at has passed
grant_request_replayed	grant_request_id seen before
grant_request_invalid	GrantRequest malformed or signature failed
grant_rejected_by_principal	Principal declined the grant
grant_nonce_mismatch	GrantResponse nonce does not match

Table 25: AIP-GRANT Error Codes

12.8. G1: Registry-Mediated Grant Flow

The G1 (Registry-Mediated) grant profile is designed for consumer deployments where the agent deployer does not operate its own Principal Wallet integration. The AIP Registry brokers the consent ceremony on the deployer’s behalf.

Actors: Agent Deployer, AIP Registry, Principal (via Registry-hosted or redirected wallet consent UI).



Step-by-step definition:

1. The deployer generates the agent's Ed25519 keypair and constructs a GrantRequest. For G1, callback_uri is REQUIRED.

2. The deployer calls POST /v1/grants. See example response.

3. The deployer redirects the principal to wallet_redirect_uri. The Registry presents a consent UI.

4. On approval, the Registry MUST construct the Principal Token and store the GrantResponse.

5. The deployer receives the GrantResponse and submits the Registration Envelope to POST /v1/agents with grant_tier: "G1".

GET /v1/grants/{grant_id} authorisation: Only the deployer whose deployer_did appears in the original GrantRequest MUST be permitted to retrieve the GrantResponse.

12.9. G2: Direct Deployer Grant Flow

The G2 (Direct Deployer) grant profile is the standard flow for applications that integrate directly with Principal Wallets. The deployer signs the GrantRequest directly and transmits it to the wallet via a front-channel binding (Web Redirect or QR Code).

Actors: Agent Deployer, Principal Wallet, Principal.

Step-by-step definition:

1. The deployer generates the agent’s Ed25519 keypair and constructs a GrantRequest.

2. The deployer signs the GrantRequest with its own Ed25519 key (deployer_did).

3. The deployer transmits the request to the Principal Wallet via a front-channel binding.

4. The Principal Wallet verifies the deployer’s signature and presents the consent UI to the principal.

5. On approval, the Principal Wallet signs the Principal Token and returns a GrantResponse to the deployer’s callback_uri.

6. The deployer submits the Registration Envelope to the Registry with grant_tier: "G2".

12.10. G3: Full Ceremony Grant Flow (OAuth 2.1)

The G3 (Full Ceremony) grant profile is required for high-assurance Tier 3 operations and environments requiring identity proofing. It uses an OAuth 2.1 Authorization Server (AS) typically operated by the Registry.

Actors: Agent Deployer, AIP Registry (as OAuth AS), Principal Wallet, Principal.

Step-by-step definition:

1. The deployer initiates an OAuth 2.1 Authorization Code Flow with PKCE at the Registry’s authorization endpoint.

2. The Registry redirects the principal to their configured Principal Wallet for authentication and consent.

3. The wallet performs high-assurance authentication (e.g., FIDO2/WebAuthn) and obtains consent.

4. The wallet returns an authorization code to the Registry.

5. The Registry issues a signed Principal Token to the deployer’s token endpoint. The token MUST include acr and amr claims.

6. The deployer submits the Registration Envelope to the Registry with grant_tier: "G3".

13. Approval Envelopes

Approval Envelopes enable a single human approval to authorise a pre-declared sequence of dependent agent actions. The principal approves the complete workflow graph upfront, and each Relying Party independently verifies its specific step against the Registry without requiring additional human interaction.

13.1. Motivation

The Cascading Approval Problem: Without Approval Envelopes, multi-step agent workflows require the human to approve each step independently, eliminating the benefit of autonomous agents. For example, an agent places an order with an e-commerce platform (step 1), which triggers a payment processor (step 2). Without Approval Envelopes, each step would require independent approval.

13.2. The Token-Expiry-While-Pending Problem

A Credential Token (TTL = 300s for Tier 2) may expire while approval is pending. Approval Envelopes decouple the approval phase from execution: the envelope waits in pending_approval state without requiring a valid token, and step-claim tokens are issued at execution time.

13.3. Approval Envelope Schema

An Approval Envelope is submitted by the orchestrating agent to POST /v1/approvals and approved by the principal through their Principal Wallet.

Field	Required	Constraints
approval_id	REQUIRED	pattern: apr:[uuid]
created_by	REQUIRED	AID of orchestrating agent
principal_id	REQUIRED	W3C DID; MUST NOT be did:aip
description	REQUIRED	minLength: 1; maxLength: 512
approval_window_expires_at	REQUIRED	ISO 8601 UTC; RECOMMENDED max: 72 hours
steps	REQUIRED	minItems: 1; maxItems: 20
compensation_steps	OPTIONAL	Compensation steps for SAGA rollback
total_value	OPTIONAL	Total financial value; MUST equal sum of step values
currency	OPTIONAL	ISO 4217; pattern: three uppercase ASCII letters
creator_signature	REQUIRED	base64url EdDSA signature
principal_signature	CONDITIONAL	REQUIRED once status is approved

Table 26: Approval Envelope Fields

total_value and currency MUST both be present or both absent. When present, total_value MUST equal the sum of value fields across ALL steps.

13.4. Step Schema

Each element in the steps array represents one atomic action at one Relying Party.

Field	Required	Constraints
step_index	REQUIRED	1-based; unique within envelope
actor	REQUIRED	AID executing this step
relying_party_uri	REQUIRED	URI of the Relying Party
action_type	REQUIRED	Application-defined; maxLength: 128
action_hash	REQUIRED	sha256:64-hex; see Section 13.7
description	REQUIRED	minLength: 1; maxLength: 256
required	REQUIRED	boolean; false = optional step
triggered_by	REQUIRED	null or step_index; 0 forbidden
value	OPTIONAL	Financial value; minimum: 0
status	READ-ONLY	pending | claimed | completed | failed | compensated | skipped

Table 27: Step Fields

The triggered_by field implements the SAGA DAG. A step with triggered_by: null is a root step. A step with triggered_by: N may only be claimed after step N reaches completed status. Circular dependencies MUST be rejected.

Multiple steps MAY share the same triggered_by value, enabling parallel execution paths.

13.5. Compensation Step Schema

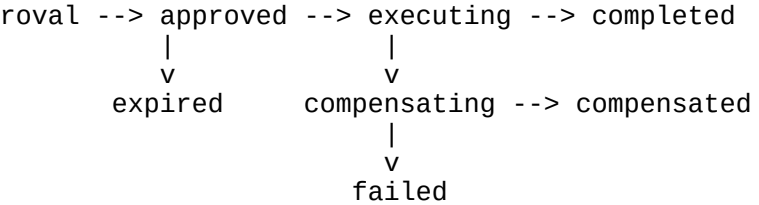
Compensation steps define SAGA rollback actions pre-authorised by the principal. If a forward step fails, compensation actions execute without requiring new human approval.

Field	Required	Constraints
compensation_index	REQUIRED	0-based index
actor	REQUIRED	AID executing compensation
relying_party_uri	REQUIRED	URI of the Relying Party
action_type	REQUIRED	Application-defined rollback
action_hash	REQUIRED	sha256:64-hex
description	REQUIRED	Plain-language rollback description

Table 28: Compensation Step Fields

13.6. Approval Envelope Lifecycle

The Registry maintains lifecycle state with valid transitions:



Transition rules:

pending_approval to approved
Principal signs via Principal Wallet. Registry stores principal_signature.

pending_approval to rejected
Principal declines.

pending_approval to expired
approval_window_expires_at passes without approval.

approved to executing
First step is claimed. Automatic transition.

executing to completed
All required: true steps reach completed status.

executing to compensating
Any required: true step fails with compensation_index present.

compensating to compensated
All compensation steps complete successfully.

compensating to failed
One or more compensation steps fail. Terminal state.

Terminal states (no further transitions): completed, compensated, failed, rejected, expired.

13.7. Action Hash Computation

The action_hash binds the principal's approval to the specific content of the action. An agent MUST NOT execute a different action than the principal approved.

The action_hash is computed as:

```
action_hash = "sha256:"
  + LCHEX( SHA-256( JCS( action_parameters ) ) )
```

Where action_parameters contains:

* approval_id - The approval_id of the envelope
* step_index - The step_index of this step
* actor - The actor AID
* relying_party_uri - The Relying Party URI
* action_type - The action type
* parameters - Application-defined JSON object

Implementations MUST use an RFC8785-conformant library. Financial amounts MUST be represented as JSON numbers (not strings).

13.8. Step Claim and Execution Protocol

To execute a step, an agent follows this protocol:

Step 1 - Claim: The agent calls POST /v1/approvals/{id}/steps/{n}/claim with a valid Credential Token and action_parameters.

The Registry MUST atomically verify all of the following:

1. Envelope status is approved or executing

2. approval_window_expires_at has not passed

3. Step status is pending

4. All triggered_by steps are completed

5. Token iss matches step actor

6. Credential Token passes validation

7. action_hash matches stored value

If all checks pass, the Registry returns a Step Execution Token signed by a key referenced in active_verification_keys.step_execution of the Registry Trust Record version current at issuance time:

```
{
  "iss": "<Registry ID>",
  "sub": "<actor AID>",
  "aud": "<relying_party_uri>",
  "iat": <now>,
  "exp": <now + 300>,
  "jti": "<UUID>",
  "aip_version": "0.3",
  "aip_scope": "&lt;scope array&gt;",
  "aip_chain": "&lt;principal chain array&gt;",
  "aip_approval_id": "<approval_id>",
  "aip_approval_step": <step_index>
}
```

Step 2 - Execute: The agent presents the Step Execution Token to the Relying Party.

Step 3 - Complete or Fail: After execution, the agent MUST call /complete or /fail within step_claim_timeout_seconds (RECOMMENDED: 600 seconds).

13.9. SAGA Compensation Semantics

When any required: true step fails after one or more earlier steps completed, the Registry MUST:

1. Transition envelope status to compensating

2. Trigger compensation steps in reverse step_index order

3. Notify the orchestrating agent

Because the principal pre-approved all compensation steps, no additional human interaction is required. This is the core SAGA property.

If a compensation step itself fails, the envelope transitions to failed (terminal). This requires manual intervention outside the AIP protocol.

13.10. Approval Envelope Validation Rules

The Registry MUST reject an Approval Envelope at submission time if any of the following are true:

1. principal_id begins with did:aip:

2. steps contains circular dependencies in triggered_by

3. steps contains duplicate step_index values

4. total_value != sum of step values

5. step currency != envelope currency

6. approval_window_expires_at is in the past

7. compensation_index references non-existent index

8. creator_signature does not verify

9. created_by agent is revoked

10. steps contains more than 20 elements

11. triggered_by is 0 or non-existent

The Registry MUST also reject envelopes from agents whose Capability Manifest does not include spawn_agents.create or minimum transactions capability.

Rule DESTRUCTIVE-1 (Mandatory Approval): Any operation involving a scope marked as destructive: true in the Registry's Scope Map MUST be authorised via a dedicated Approval Envelope step or a direct human confirmation ceremony. Registry-mediated G1 grants MUST NOT be used for destructive operations without an additional out-of-band confirmation.

14. Reputation and Endorsements

14.1. Endorsement Object

Any Relying Party or agent MAY submit a signed Endorsement Object to the Registry after a completed interaction. The schema is in Section 5.8.

The Registry MUST verify every submitted Endorsement Object signature. Only success and partial outcomes increment endorsement_count. failure increments incident_count.

Completed Approval Envelope steps SHOULD generate Endorsement Objects. When an Approval Envelope reaches completed status, the orchestrator SHOULD submit an Endorsement Object for each agent that executed a step successfully.

14.2. Reputation Scoring

The Registry MUST expose reputation data for every registered AID at GET /v1/agents/{aid}/reputation. Required fields: registration_date, task_count, successful_task_count, endorsement_count, incident_count, revocation_history, last_active.

The Registry MAY expose a reference advisory score labelled advisory_only: true. Relying Parties MUST NOT treat it as normative. AIP standardises reputation inputs, not the scoring formula.

Reputation Non-Transferability: Reputation is bound to a specific AID. Implementations MUST NOT transfer reputation from revoked to new AIDs. Endorsements from AIDs with current full_revoke or principal_revoke status MUST NOT be weighted.

15. Lifecycle States

An AID has exactly two lifecycle states: active and revoked. AIP does not define an inactive status; the Dead Man's Switch mechanism (Section 21.8) uses full_revoke.

An AID MUST remain valid until explicitly revoked. Revoked AIDs MUST NOT be reused. Key rotation preserves the AID but changes the active key. Outstanding tokens signed under the previous key remain valid until their exp.

Ephemeral agents MUST have a non-null task_id. They MUST be explicitly revoked on task completion. The Registry SHOULD auto-revoke ephemeral agents when their stored expires_at passes.

16. Principal Chain

The principal.id field MUST be byte-for-byte identical in every Principal Token in the aip_chain array. Relying Parties MUST verify this. An intermediate agent MUST NOT change it, substitute its own AID, or modify the original principal DID in any way.

Every element in aip_chain is a compact-serialised JWT whose payload conforms to the Principal Token schema. Elements are ordered root-to-leaf. The maximum aip_chain length is 11. Implementations MUST reject tokens with aip_chain length exceeding 11.

Cryptographic non-repudiation: Every Credential Token carries a delegation chain in which each link is signed by the delegating party’s private key. Because principal.id is byte-identical across all chain elements and is bound to the signing key, every agent action is cryptographically attributable to the human or organisational principal that authorised it. This satisfies the non-repudiation requirement of [SP-800-63-4] Section 11.

17. Registry Interface

A conformant AIP Registry MUST implement the HTTP endpoints defined in this section.

17.1. Required Endpoints

A conformant AIP Registry MUST implement the following HTTP endpoints:

Method	Path	Description
POST	/v1/agents	Register a new AID (Registration Envelope)
GET	/v1/agents/{aid}	Retrieve Agent Identity or DID

		Document
PUT	/v1/agents/{aid}	Key rotation only
GET	/v1/agents/{aid}/public-key	Current public key (JWK)
GET	/v1/agents/{aid}/public-key/{key-id}	Historical key version
GET	/v1/agents/{aid}/capabilities	Current Capability Manifest
PUT	/v1/agents/{aid}/capabilities	Replace Capability Manifest
GET	/v1/agents/{aid}/revocation	Revocation status
POST	/v1/revocations	Submit RevocationObject
GET	/v1/crl	Certificate Revocation List
GET	/v1/agents/{aid}/reputation	Reputation data
POST	/v1/endorsements	Submit Endorsement Object
POST	/v1/grants	Submit G1 grant
GET	/v1/grants/{grant_id}	Retrieve grant status
PUT	/v1/agents/{aid}/overlays	Submit Capability Overlay
GET	/v1/agents/{aid}/overlays	Retrieve current overlay
POST	/v1/engagements	Create Engagement Object
GET	/v1/engagements/{id}	Retrieve Engagement Object
PUT	/v1/engagements/{id}	Update Engagement (append change

		log)
POST	/v1/subscriptions	Create RPNP subscription (Section 11.4)
DELETE	/v1/subscriptions/{id}	Cancel RPNP subscription
GET	/v1/scopes	Scope Map registry
POST	/v1/oauth/authorize	G3 authorization endpoint
POST	/v1/oauth/token	G3 token endpoint / token exchange
GET	/.well-known/oauth-authorization-server	AS Metadata [RFC8414]
GET	/.well-known/aip-registry	Retrieve Registry discovery metadata
GET	/v1/registry-trust/current	Retrieve current Registry Trust Record
GET	/v1/registry-trust/{version}	Retrieve immutable Registry Trust Record by version

Table 29

17.2. AID URL Encoding

In all path parameters, the ‘did:aip:‘ prefix and colons MUST be percent-encoded per [RFC3986]:

did:aip:personal:9f3a1c82b4e6d7f0a2b5c8e1d4f7a0b3
→ /v1/agents/did%3Aaip%3Apersonal%3A9f3a1c82b4e6d7f0a2b5c8e1d4f7a0b3

17.3. Response Format

All Registry responses MUST use ‘Content-Type: application/json‘.

All timestamps MUST be in ISO 8601 UTC format.

17.4. Approval Envelope Endpoints

A conformant AIP Registry MUST implement the following additional endpoints for Approval Envelopes:

Method	Path	Description
POST	/v1/approvals	Submit Approval Envelope for approval
GET	/v1/approvals/{id}	Retrieve Approval Envelope with step statuses
POST	/v1/approvals/{id}/approve	Principal approves (wallet call; sets 'principal_signature')
POST	/v1/approvals/{id}/reject	Principal rejects
POST	/v1/approvals/{id}/steps/{n}/claim	Claim step for execution; returns Step Execution Token
POST	/v1/approvals/{id}/steps/{n}/complete	Mark step completed
POST	/v1/approvals/{id}/steps/{n}/fail	Mark step failed; triggers compensation
GET	/v1/approvals/{id}/steps/{n}	Get step status (used by Relying Parties for Step Execution Token verification)
POST	/v1/approvals/{id}/compensation-steps/{n}/claim	Claim compensation step
POST	/v1/approvals/{id}/compensation-steps/{n}/complete	Mark compensation step completed
POST	/v1/approvals/{id}/compensation-steps/{n}/fail	Mark compensation step failed

Table 30

‘POST /v1/approvals‘ validation: The Registry MUST perform all checks defined in Section 13.10 before accepting an Approval Envelope. The Registry MUST return HTTP 201 with the stored envelope (including the Registry-assigned ‘status: "pending_approval"‘) on success.

‘POST /v1/approvals/{id}/approve‘ flow: This endpoint is called by the Principal Wallet after the principal completes the signing ceremony. The request body MUST contain the ‘principal_signature‘ field: a base64url EdDSA signature computed by the principal’s wallet over the JCS-canonical serialisation of the full Approval Envelope with ‘principal_signature‘ set to the empty string ‘""‘ before serialisation — identical to the pattern used for ‘creator_signature‘. The signing key MUST be the private key of ‘principal_id‘. The Registry MUST verify this signature against the ‘principal_id‘’s resolved public key before transitioning the envelope to ‘approved‘.

Atomicity requirement for step claim: The Registry MUST implement step-claim operations atomically (e.g., using optimistic locking or a distributed lock) to prevent two actors from claiming the same step simultaneously. Only one claim MUST succeed; the other MUST receive ‘approval_step_already_claimed‘.

Step Execution Token format: The Step Execution Token returned by ‘POST .../steps/{n}/claim‘ is a JWT signed by a key referenced in ‘active_verification_keys.step_execution‘ of the Registry Trust Record version current at issuance time, with the claims described in Section 13.8. Its TTL MUST comply with normal TTL rules for the scopes involved.

17.5. OAuth 2.1 Authorization Server

A conformant AIP Registry supporting G3 grants MUST implement an OAuth 2.1 Authorization Server [RFC6749] with the following requirements:

1. The authorization endpoint MUST be published in the Registry’s well-known configuration at ‘/.well-known/oauth-authorization-server‘ per [RFC8414].

2. PKCE [RFC7636] with ‘code_challenge_method: "S256"‘ is REQUIRED for all authorization requests.

3. The ‘scope‘ parameter MUST use AIP scope URIs from the ‘urn:aip:scope:‘ namespace.

4. The token endpoint MUST return a signed Principal Token (not a standard OAuth access token) as the ‘access_token‘ value, with ‘token_type: "AIP+JWT"‘.

5. The token response MUST include ‘acr‘ and ‘amr‘ claims reflecting the authentication performed.

6. The authorization server MUST support the ‘acr_values‘ parameter to declare minimum identity-proofing requirements.

7. DPoP [RFC9449] MUST be supported on the token endpoint.

The token endpoint also serves RFC 8693 [RFC8693] token exchange.

17.5.1. Well-Known Metadata Additions

The ‘/.well-known/aip-registry‘ response MUST also include:

Field	Type	Description
‘registry_trust_uri‘	string	URI of the current Registry Trust Record
‘grant_tiers_supported‘	array	Supported grant tiers: ‘["G1", "G2", "G3"]‘
‘acr_values_supported‘	array	Supported ‘acr‘ values
‘amr_values_supported‘	array	Supported ‘amr‘ values
‘oauth_authorization_server‘	string	URI of the OAuth AS metadata document (present only if G3 supported)
‘identity_proofing_required_for_tier2‘	boolean	Whether this Registry requires G3 for Tier 2 operations

Table 31

17.6. Scope Map

All AIP capability scopes MUST be representable as URIs in the ‘urn:aip:scope:‘ namespace:

scope_string → urn:aip:scope:<scope_string>

Examples: ‘email.read‘ → ‘urn:aip:scope:email.read‘, ‘spawn_agents.create‘ → ‘urn:aip:scope:spawn_agents.create‘.

When used in OAuth ‘scope‘ parameters, the URI form MUST be used. Within AIP Credential Tokens (‘aip_scope‘ array), the short string form remains canonical.

A conformant AIP Registry MUST implement ‘GET /v1/scopes‘ returning a JSON object mapping scope strings to metadata:

Field	Type	Description
‘uri‘	string	Full ‘urn:aip:scope:‘ URI
‘description‘	string	Human-readable description
‘tier‘	integer	Tier classification (1, 2, or 3)
‘destructive‘	boolean	Whether the scope requires additional confirmation per Section 12.3
‘constraint_schema‘	object/ null	JSON Schema fragment for scope-specific constraints

Table 32

17.7. Engagement Endpoints

A conformant AIP Registry supporting Engagement Objects MUST implement:

Method	Path	Description
POST	/v1/engagements	Create Engagement Object
GET	/v1/engagements/{id}	Retrieve Engagement Object
PUT	/v1/engagements/{id}	Update Engagement (append change log entry)

Table 33

Create validation: The Registry MUST verify the initiator's DID signature, validate that all referenced participant AIDs exist and are not revoked, and assign 'status: "active"' with 'seq_number: 1'.

Update validation: The Registry MUST verify the submitter is an active participant, the change log entry's 'seq_number' is exactly 'current_max + 1', and the entry signature is valid. The Registry MUST reject modifications to existing entries with 'change_log_immutable'.

17.8. RPNP Subscription Endpoints

A conformant AIP Registry supporting RPNP (Section 11.4) MUST implement:

Method	Path	Description
POST	/v1/subscriptions	Create RPNP subscription
GET	/v1/subscriptions/{id}	Retrieve subscription status
DELETE	/v1/subscriptions/{id}	Cancel subscription

Table 34

Subscription creation MUST be authenticated via DPoP. The 'webhook_uri' MUST use HTTPS. The Registry MUST reject non-HTTPS URIs with 'invalid_webhook_uri'.

17.9. Key Management and Rotation

Registry trust evolution is defined in Section 7.3.6. Planned rotation preserves the same registry_id and uses versioned Registry Trust Records with overlapping signatures, expiry, and rollback checks. Emergency compromise recovery uses explicit re-bootstrap and MUST NOT be accepted automatically.

18. Error Handling

18.1. Error Response Format

EXAMPLE (informative):

```
{
  "error": "<error_code>",
  "error_description": "<human-readable description>",
  "error_uri": "https://provai.dev/errors/<error_code>"
}
```

Implementations MUST use exact string values for ‘error‘.

Implementations MUST NOT return HTTP 200 for error conditions.

18.2. Standard Error Codes

invalid_token HTTP 401. Token malformed, invalid signature, or invalid claims.

token_expired HTTP 401. Token exp is in the past.

token_replayed HTTP 401. Token jti seen before within validity window.

dpop_proof_required HTTP 401. DPoP proof absent or invalid.

delegation_chain_refresh_required HTTP 401. Principal Token in chain has expired; agent must re-establish delegation.

agent_revoked HTTP 403. The AID has been revoked.

insufficient_scope HTTP 403. Operation not within granted scopes.

invalid_delegation_depth HTTP 403. delegation_depth mismatch or exceeds max_delegation_depth.

chain_token_expired HTTP 403. Principal Token in aip_chain expired.

delegation_chain_invalid HTTP 403. Structural error in delegation chain.

manifest_invalid HTTP 403. Capability Manifest signature failed or unavailable.

manifest_expired HTTP 403. Capability Manifest expires_at passed.

approval_envelope_invalid HTTP 400. Approval Envelope malformed, circular dependencies, or hash mismatch.

approval_envelope_expired HTTP 403. approval_window_expires_at has passed.

approval_not_found HTTP 404. Approval Envelope ID not found.

approval_step_prerequisites_unmet HTTP 403. triggered_by step not yet completed.

approval_step_already_claimed HTTP 409. Step already claimed by another actor.

approval_step_action_mismatch HTTP 403. Presented action_parameters hash does not match stored action_hash.

approval_step_invalid HTTP 403. Step Execution Token verification against Registry failed.

grant_request_expired HTTP 400. AIP-GRANT request_expires_at passed.

grant_request_replayed HTTP 400. AIP-GRANT grant_request_id seen before.

grant_request_invalid HTTP 400. GrantRequest malformed or signature failed.

grant_rejected_by_principal HTTP 403. Principal declined the grant.

grant_nonce_mismatch HTTP 400. GrantResponse nonce does not match.

unknown_aid HTTP 404. AID not registered in any accessible Registry.

registry_unavailable HTTP 503. Registry could not be reached.

rate_limit_exceeded HTTP 429. Rate limit for this operation exceeded; see Section 19.

revocation_stale HTTP 403. Tier 2 operation with cached revocation status.

dpop_required HTTP 401. Tier 2 operation without DPoP proof.

mtls_required HTTP 403. Tier 3 operation without mTLS.

invalid_scope HTTP 400. Token contains retired bare spawn_agents scope.

principal_did_method_forbidden HTTP 403. Principal uses did:key for Tier 2 scope.

identity_proofing_insufficient HTTP 403. G3 identity proofing below requested acr_values.

grant_not_found HTTP 404. G1 grant_id not found or expired.

grant_deployer_mismatch HTTP 403. G1 grant_id does not match deployer.

pkce_required HTTP 400. G3 authorization request missing PKCE.

registry_untrusted HTTP 403. Registry does not match principal DID-Document-declared Registry.

overlay_exceeds_manifest HTTP 400. Overlay violates CO-1 attenuation rule.

overlay_issuer_invalid HTTP 400. Overlay issuer uses did:key.

overlay_version_conflict HTTP 409. Overlay version not strictly increasing.

overlay_signature_invalid HTTP 400. Overlay signature verification failure.

engagement_terminated HTTP 403. Engagement has been terminated or completed.

engagement_suspended HTTP 403. Engagement is currently suspended.

engagement_participant_removed HTTP 403. Agent removed from engagement.

engagement_gate_pending HTTP 403. Required approval gate not yet approved.

engagement_not_found HTTP 404. Engagement ID not found.

engagement_countersign_required HTTP 400. Missing required countersignature.

change_log_immutable HTTP 400. Attempt to modify change log entry.

change_log_sequence_invalid HTTP 400. Out-of-sequence change log append.

subscription_auth_required HTTP 401. RPNP subscription without DPoP.

subscription_scope_forbidden HTTP 403. scope_filter: "all" rejected by Registry policy.

invalid_webhook_uri HTTP 400. Webhook URI not HTTPS.

subscription_limit_exceeded HTTP 429. RPNP subscription limit reached.

invalid_target HTTP 400. Token exchange resource not registered.

18.3. Error Detail Types

For ‘registry_unavailable‘ (503): SHOULD include ‘Retry-After‘ per [RFC9110].

For ‘rate_limit_exceeded‘ (429): MUST include ‘Retry-After‘ per [RFC6585] Section 4 and rate limit headers per Section 19.1.

For ‘delegation_chain_refresh_required‘ (401): the response SHOULD include an ‘error_description‘ indicating which delegation depth’s Principal Token has expired, to help the agent identify which delegation level to renew.

19. Rate Limiting and Abuse Prevention

Rate limiting protects the Registry from denial-of-service attacks, registration floods, and validation-driven key lookup storms. A public Registry that permits unrestricted write operations or validation-driven lookups is exploitable in ways that undermine the security guarantees of the entire ecosystem.

19.1. Rate Limit Response Format

When rate limiting is applied, the Registry MUST return HTTP 429 with the following headers:

Header	Required	Description
‘Retry-After‘	MUST	Seconds until the client may retry, or a HTTP-date per [RFC9110]
‘X-RateLimit-Limit‘	SHOULD	The request limit for this window
‘X-RateLimit-Remaining‘	SHOULD	Remaining requests in this window
‘X-RateLimit-Reset‘	SHOULD	Unix timestamp when the window resets
‘X-RateLimit-Policy‘	MAY	Human-readable description of the applicable policy

Table 35

The response body MUST conform to the error response format (Section 18.1) with ‘error: "rate_limit_exceeded"‘ and a human-readable ‘error_description‘ identifying the rate-limited operation and the applicable window.

19.2. Per-Endpoint Rate Limit Categories

The Registry MUST implement separate rate limit buckets for each of the following operation categories. The limits below are RECOMMENDED minimums; Registry operators MAY enforce stricter limits based on observed traffic patterns and threat models.

19.2.1. Category R1 - Registration writes

(‘POST /v1/agents‘):

* Per-principal-DID: RECOMMENDED limit of 20 agent registrations per hour. This prevents a single compromised principal key from flooding the Registry with rogue agent registrations.

* Per-source-IP: RECOMMENDED limit of 50 registrations per hour across all principals. This prevents registration floods from a single network origin, regardless of the principal DID presented.

* Global: Registries SHOULD implement a global registration rate limit appropriate to their infrastructure capacity.

19.2.2. Category R2 - Key rotation writes

(‘PUT /v1/agents/{aid}‘):

* Per-AID: RECOMMENDED limit of 10 key rotations per 24-hour window. Legitimate key rotation is infrequent; high frequency suggests automated abuse or a compromised orchestrator.

19.2.3. Category R3 - Revocation writes

(‘POST /v1/revocations‘):

* Per-issuer-DID: RECOMMENDED limit of 100 revocations per hour. Higher limits are legitimate for enterprise orchestrators managing large ephemeral agent fleets. Registries MAY issue higher limits to verified principals.

* Registries MUST apply special throttling to ‘propagate_to_children: true‘ revocations that would cascade to more than 100 descendants, as these trigger recursive Registry writes. A revocation that would cascade to more than 100 descendants SHOULD be queued and processed asynchronously, with the Registry returning HTTP 202 (Accepted) and a status URI rather than blocking on the full cascade.

19.2.4. Category R4 - Validation-driven key reads

(‘GET /v1/agents/{aid}/public-key/{key-id}‘, ‘GET /v1/agents/{aid}/revocation‘):

* Per-requesting-IP: RECOMMENDED limit of 1,000 requests per minute across all AIDs. Validation-driven reads are triggered by token verification; legitimate Relying Parties have bounded lookup rates.

* Per-AID: RECOMMENDED limit of 200 reads per minute. A single AID being looked up 200 times per minute from varied IPs is likely the subject of a coordinated replay attack; rate limiting per AID allows the Registry to throttle targeted abuse.

* Registries SHOULD offer API key authentication for Relying Parties whose legitimate validation rates exceed these limits (e.g., high-traffic APIs that verify thousands of agent tokens per minute).

19.2.5. Category R5 - CRL reads

('GET /v1/crl'):

* The CRL endpoint MUST be served from a CDN or distributed infrastructure (Section 11.2). Direct-origin CRL reads SHOULD be rate limited per IP to 100 requests per minute to protect against CDN bypass attacks. CDN-served responses have no normative rate limit constraint.

19.2.6. Category R6 - Endorsement writes

('POST /v1/endorsements'):

* Per-from-AID: RECOMMENDED limit of 500 endorsements per hour. This prevents an AID from artificially inflating another AID's 'endorsement_count' through automated submission.

* Self-endorsement ('from_aid' == 'to_aid') MUST be rejected at the application layer before rate limits are checked.

19.2.7. Category R7 - Approval Envelope writes and step claims

('POST /v1/approvals', 'POST /v1/approvals/{id}/steps/{n}/claim'):

* Per-principal-DID: RECOMMENDED limit of 100 Approval Envelopes per hour. Approval Envelopes represent human-authorised workflows; high frequency is anomalous.

* Per-actor-AID per envelope: step claims are naturally rate-limited by the sequential structure of the workflow. No additional rate limit is required for step claims within a single envelope.

19.3. Registration Abuse Prevention

Beyond rate limiting, the Registry MUST implement the following structural checks to prevent registration abuse:

19.3.1. AID uniqueness enforcement

The Registry MUST check AID uniqueness under a distributed lock or equivalent atomic mechanism. Two simultaneous registration requests for the same AID MUST result in exactly one succeeding and one receiving an appropriate error (Section 6.2 Check 4).

19.3.2. Principal delegation chain verification at registration

The Registry MUST verify that the ‘principal_token‘ in the Registration Envelope was issued by a DID that is resolvable and has not been subjected to a ‘full_revoke‘ or ‘principal_revoke‘ RevocationObject in the Registry. A revoked principal MUST NOT be permitted to register new agents.

For the purposes of this check, a **revoked principal** is defined as a principal DID (‘principal_token.principal.id‘) against which a ‘principal_revoke‘ Revocation Object has been submitted to this Registry. The Registry MUST maintain an index of principal DIDs associated with ‘principal_revoke‘ revocations and MUST reject new Registration Envelopes where ‘principal_token.principal.id‘ matches a DID in this index, unless the ‘principal_revoke‘ object explicitly scopes the revocation to a specific agent AID other than the one being registered.

19.3.3. Registration flood from shared principals

If a single principal DID registers more than 1,000 agents (across all time), the Registry SHOULD require the deployer to present proof of legitimate use (out-of-band, implementation-specific). This is a SHOULD, not a MUST, because legitimate orchestrator-heavy deployments may reach this threshold.

19.3.4. Public Registry challenge for unauthenticated deployers

Registries that permit registration without deployer authentication (open Registries) SHOULD implement a lightweight proof-of-work or CAPTCHA mechanism for registrations where ‘deployer_did‘ is absent from the principal_token context.

19.4. Validation-Driven Lookup Limits

Key lookup amplification occurs when an adversary presents many tokens with distinct ‘kid‘ values, forcing the Registry to perform a lookup for each. Mitigations:

19.4.1. Key version caching

Relying Parties MUST cache resolved public keys for a given ‘kid‘ for up to 300 seconds. Repeated validation of tokens with the same ‘kid‘ SHOULD NOT trigger repeated Registry lookups within the cache window.

19.4.2. Historical key depth limit

Registries MAY reject requests for key versions older than a configurable retention window (RECOMMENDED: 90 days past the key rotation date, since all tokens issued with that key must have expired within 3600s of rotation). This prevents adversaries from constructing tokens with ancient, never-rotated keys to force deep history lookups.

19.4.3. ‘kid‘ validation at the Relying Party

Relying Parties MUST validate that the ‘kid‘ in the token header matches the pattern: did:aip:<lowercase-namespace>:<32-lowercase-hex>#key-<positive-integer> before performing any Registry lookup (Validation Step 3). Malformed ‘kid‘ values MUST be rejected with ‘invalid_token‘ without making a Registry call.

19.5. Approval Envelope Rate Limits

Approval Envelope operations require specific abuse prevention because they involve asynchronous principal interactions and potential cascade effects.

19.5.1. Envelope submission rate

Per Section 19.2, Category R7.

19.5.2. Pending envelope limit

A Registry SHOULD enforce a maximum of 1,000 ‘pending_approval‘ envelopes per principal DID at any time. Envelopes that expire transition to ‘expired‘ and free this quota.

19.5.3. Step claim timeout

Step claims that are not completed or failed within 600 seconds MUST be automatically failed by the Registry. This prevents a claimed step from blocking the workflow indefinitely due to a crashed or unresponsive agent.

19.5.4. Compensation cascade depth

Compensation step execution is not rate-limited separately - it is a recovery mechanism whose scope is bounded by the number of forward steps (maximum 20). No additional rate limit is required for compensation.

19.6. Graduated Backoff Requirements

Clients that receive HTTP 429 responses MUST implement exponential backoff with jitter. The minimum retry interval is the value in the ‘Retry-After‘ header. Clients MUST NOT retry before ‘Retry-After‘ expires.

Implementations SHOULD use the following backoff formula:

```
BACKOFF_BASE = 1         ; fixed exponential base, in seconds
MAX_DELAY    = 3600      ; hard ceiling, in seconds

computed_delay = min(BACKOFF_BASE * 2^attempt, MAX_DELAY)
jitter         = random(0, BACKOFF_BASE)
retry_delay    = max(computed_delay + jitter, Retry-After)
```

The ‘Retry-After‘ value from the 429 response acts as a mandatory minimum: clients MUST NOT retry before ‘Retry-After‘ seconds have elapsed, even if ‘computed_delay + jitter‘ is smaller. When no ‘Retry-After‘ header is present, clients MUST treat it as 0 and rely solely on the exponential formula.

Clients that continue to receive HTTP 429 after 5 exponential backoff attempts MUST cease retrying for a minimum of 1 hour and SHOULD alert an operator. Persistent rate limiting at this scale indicates either a misconfigured client or a sustained attack pattern.

Registries MUST track clients that consistently exceed rate limits and MAY temporarily block their source IPs or API keys after sustained abuse. Blocking decisions are implementation-specific and are not normatively constrained by this specification.

20. Versioning and Compatibility

AIP follows Semantic Versioning. Before v1.0, MINOR versions MAY include breaking changes.

Breaking changes from v0.2:

* aip_version is now "0.3" for conforming implementations.
* X-AIP-Version: 0.3 replaces X-AIP-Version: 0.2.
* The bare spawn_agents scope is retired; use spawn_agents.create and spawn_agents.manage (Section 3.1).
* Registration Envelopes MUST include grant_tier.

* Principal DID Documents for Tier 2 agents MUST include an AIPRegistry service entry.

* The /.well-known/aip-registry response MUST include registry_id, registry_trust_uri, registry_name, and endpoints. Registry Trust Records are versioned separately and MUST support sequential updates.

* §19.6 backoff formula corrected (Section 19.6).

Implementations MUST NOT silently accept tokens from unsupported versions without logging a version warning.

20.1. Tier Conformance

Tier	Revocation	DPoP	mTLS	grant_tier	Principal DID Method
1	CRL (15 min)	NOT REQUIRED	NOT REQUIRED	G1 or G2	Any (note 1)
2	Real-time	REQUIRED	NOT REQUIRED	G2 or G3	did:web (note 2)
3	Real-time + OCSP	REQUIRED	REQUIRED	G3	did:web (note 2)

Table 36

Note 1 For Tier 1, the principal MAY use any W3C DID method. Principals using did:key are permitted for Tier 1 only; see Note 2.

Note 2 did:aip is NEVER a valid Principal DID method for Tier 2 or Tier 3. The principal.id field MUST NOT use the did:aip method. Principals using did:key for Tier 2 agents MUST be rejected with principal_did_method_forbidden; the did:key method does not support a DID Document with the AIPRegistry service entry required by Section 7.2.

21. Security Considerations

This section describes the security considerations for AIP implementations, including the threat model, cryptographic requirements, and recommended mitigations.

21.1. Threat Model

AIP identifies the following threat categories:

TS-1: Token Replay
: An adversary reuses a captured Credential Token. Mitigation: JTI replay cache.

TS-2: Key Compromise
: An adversary steals an agent's private key. Mitigation: Key rotation, HSM storage.

TS-3: Delegation Escalation
: A child agent exceeds granted scope. Mitigation: Rule D-1 (Scope Inheritance), Step 9c validation.

TS-4: Registry Impersonation
: A malicious Registry serves fake revocation status. Mitigation: Well-known configuration, key pinning.

TS-5: Principal Impersonation
: An adversary forges a Principal Token. Mitigation: DID resolution verification.

TS-6: Revocation Delay
: The gap between revocation issue and propagation. Mitigation: Real-time RPNP for Tier 2.

TS-7: Token Theft
: An adversary intercepts a token in transit. Mitigation: TLS 1.2+, DPoP.

TS-8: Child Agent Self-Replication
: An agent with spawn_agents.create creates children during TTL window. Mitigation: Tier 2 with propagate_to_children.

TS-9: Action Hash Manipulation
: An agent executes different action than approved. Mitigation: Action hash verification at claim time.

21.2. Cryptographic Requirements

Operation	Algorithm	Specification	Status
Signing / Verification	Ed25519 (EdDSA)	[RFC8037]	MUST

Hashing	SHA-256	[FIPS-180-4]	MUST
Key representation	JWK	[RFC7517]	MUST
Key exchange (future)	X25519	[RFC7748]	MUST

Table 37: Mandatory-to-Implement Cryptography

Optional suites: ES256 (ECDSA P-256) per [RFC7518] for WebAuthn compatibility; RS256 (RSA-PKCS1) per [RFC7518] for legacy enterprise only - MUST NOT be the sole supported algorithm; RSA keys MUST be at least 2048 bits.

Prohibited: none, HS256/384/512, RS512, MD5, SHA-1. The alg header MUST be explicitly specified.

21.3. Proof-of-Possession (DPoP)

DPoP is REQUIRED for transactions.*, communicate.*, and filesystem.execute scopes. DPoP proofs MUST use EdDSA.

DPoP Proof Header:

```
{
  "typ": "dpop+jwt",
  "alg": "EdDSA",
  "jwk": {
    "kty": "OKP",
    "crv": "Ed25519",
    "x": "<base64url public key>",
    "kid": "<DID URL>"
  }
}
```

DPoP Proof Payload:

```
{
  "jti": "<UUID v4>",
  "htm": "<HTTP method, uppercase>",
  "htu": "<scheme + host + path>",
  "iat": "<Unix timestamp>",
  "ath": "<BASE64URL(SHA-256(token)))>"
}
```

Relying Party validation: Verify alg is EdDSA, verify jwk.kid matches token kid, verify htm/htu/iat, check jti replay, verify ath, verify signature.

21.4. Key Management

Private keys MUST NOT be stored in plaintext on disk, transmitted in protocol messages, or included in logs. Private keys SHOULD be stored in HSM, secure enclave, or OS-level keychain.

Key rotation: Generate new keypair, increment version, include previous_key_signature signed by retiring key, submit to Registry, retain retiring key until outstanding tokens expire.

Key compromise response: Immediately revoke with reason: key_compromised, register new AID with new keypair, re-establish delegations.

21.5. Token Security

All tokens MUST be transmitted over TLS 1.2 or higher (TLS 1.3 RECOMMENDED). MUST NOT transmit over unencrypted HTTP.

JTI replay cache: Keyed by (iss, jti); window at least max TTL for served scopes; shared cache for distributed deployments.

Audience validation: MUST validate aud claim. Mismatch returns invalid_token.

21.6. Delegation Chain Security

Default max_delegation_depth MUST NOT exceed 3. Hard cap is 10. Circular delegation MUST be detected and rejected.

Ephemeral agents MUST have non-null task_id. Registry SHOULD auto-revoke when expires_at passes.

21.7. Registry Security

All write operations MUST be authenticated. Revocation issued_by MUST be verified as in the target’s delegation chain.

Registry read endpoints MUST target 99.9% uptime. CRL MUST be served from CDN.

Relying Parties MUST NOT trust the aip_registry claim in a Credential Token as sole indicator. For Tier 2, the authoritative Registry MUST be verified via the root principal’s DID Document.

21.8. Revocation Security

Every Revocation Object MUST be signed. Unsigned or invalidly signed objects MUST be rejected.

Dead Man's Switch (Optional): Registry MAY issue full_revoke for agents that fail to submit a signed heartbeat within configured window (RECOMMENDED: 24 hours).

Timing attack mitigation: Short sensitive-scope TTLs (max 300s) limit exploitation window.

21.9. Approval Envelope Security

Action hash integrity: The action_hash binds principal approval to specific action parameters. Registry MUST reject any claim where recomputed hash does not match stored action_hash.

Double-spend prevention: Atomic step-claim ensures each step claimed by exactly one actor.

Envelope replay prevention: approval_id is UUID v4 unique per envelope. Registries MUST reject duplicate approval_id values.

21.10. Privacy Considerations

AIP is designed for zero-trust environments. No personally identifiable information (PII) is transmitted in Credential Tokens beyond the AID and delegation chain.

Registries MUST NOT log or retain Credential Token payloads beyond validation. Relying Parties MUST NOT store tokens after validation.

The principal_id in Principal Tokens enables attribution for compliance but does not inherently reveal principal identity to Relying Parties.

22. IANA Considerations

This section describes the IANA considerations for the AIP specification.

22.1. DID Method Registration

The did:aip DID method is requested for registration in the W3C DID Method Registry per W3C DID Specification.

Field	Value
Method Name	did:aip
Status	Draft
Canonical ID	did:aip:namespace:32-hex-characters

Table 38: did:aip Method Registration

22.2. Scope URI Namespace

AIP defines a URN namespace for scope identifiers: urn:aip:scope:*. This namespace is used in OAuth 2.0 token exchange to identify capability scopes.

urn:aip:scope:email.read
urn:aip:scope:email.write
urn:aip:scope:transactions
urn:aip:scope:spawn_agents.create

22.3. Grant Tier Registry

Value	Description
G1	Registry-Mediated Grant Flow
G2	Direct Deployer Grant Flow
G3	Full Ceremony Grant Flow

Table 39: Grant Tier Values

22.4. Error Code Registry

AIP defines a registry of error codes.

Code	HTTP	Description
invalid_token	401	Token malformed, invalid signature, or invalid claims.

token_expired	401	Token exp is in the past.
token_replayed	401	Token jti seen before within validity window.
dpop_proof_required	401	DPoP proof absent or invalid.
delegation_chain_refresh_required	401	Principal Token in chain has expired.
agent_revoked	403	The AID has been revoked.
insufficient_scope	403	Operation not within granted scopes.
invalid_delegation_depth	403	delegation_depth mismatch or exceeds limit.
chain_token_expired	403	Principal Token in aip_chain expired.
delegation_chain_invalid	403	Structural error in delegation chain.
manifest_invalid	403	Capability Manifest signature failed or unavailable.
manifest_expired	403	Capability Manifest expires_at passed.
approval_envelope_invalid	400	Approval Envelope malformed or hash mismatch.
approval_envelope_expired	403	approval_window_expires_at has passed.
approval_not_found	404	Approval Envelope ID not found.
approval_step_prerequisites_unmet	403	triggered_by step not yet completed.
approval_step_already_claimed	409	Step already claimed by another actor.

approval_step_action_mismatch	403	Action hash mismatch.
grant_request_expired	400	AIP-GRANT request_expires_at passed.
grant_rejected_by_principal	403	Principal declined the grant.
unknown_aid	404	AID not registered in any accessible Registry.
registry_unavailable	503	Registry could not be reached.
rate_limit_exceeded	429	Rate limit for this operation exceeded.
revocation_stale	403	Tier 2 operation with cached revocation status.
principal_did_method_forbidden	403	Principal uses did:key for Tier 2 scope.
registry_untrusted	403	Registry does not match principal DID-Document.
overlay_exceeds_manifest	400	Overlay violates CO-1 attenuation rule.
engagement_terminated	403	Engagement has been terminated or completed.

Table 40: AIP Error Codes

22.5. Media Types

Type	Description
application/aip+jwt	AIP Credential Token (JWT format)

Table 41: AIP Media Types

The typ header value for AIP Credential Tokens is AIP+JWT per RFC 7515.

23. Normative References

[RFC2119] Bradner, S., "Key words for use in RFCs", BCP 14, March 1997.

[RFC3986] Berners-Lee et al., "URI Generic Syntax", STD 66, January 2005.

[RFC5234] Crocker & Overell, "ABNF", STD 68, January 2008.

[RFC5280] Cooper et al., "X.509 PKI Certificate Profile", May 2008.

[RFC6585] Nottingham & Fielding, "Additional HTTP Status Codes", April 2012.

[RFC7515] Jones, M., Bradley, J., and N. Sakimura, "JSON Web Signature (JWS)", RFC 7515, DOI 10.17487/RFC7515, May 2015, <https://www.rfc-editor.org/rfc/rfc7515>.

[RFC7517] Jones, M., "JSON Web Key (JWK)", May 2015.

[RFC7518] Jones, M., "JSON Web Algorithms (JWA)", May 2015.

[RFC7519] Jones, M., Bradley, J., and N. Sakimura, "JSON Web Token (JWT)", RFC 7519, DOI 10.17487/RFC7519, May 2015, <https://www.rfc-editor.org/rfc/rfc7519>.

[RFC7748] Langley et al., "Elliptic Curves for Security", January 2016.

[RFC8032] Josefsson, S. and I. Liusvaara, "Edwards-Curve Digital Signature Algorithm (EdDSA)", RFC 8032, DOI 10.17487/RFC8032, January 2017, <https://www.rfc-editor.org/rfc/rfc8032>.

[RFC8037] Liusvaara, I., "CFRG Elliptic Curves for JOSE", January 2017.

[RFC8174] Leiba, B., "Ambiguity of Uppercase in RFC 2119", BCP 14, May 2017.

[RFC8259] Bray, T., "JSON Data Interchange Format", STD 90, December 2017.

[RFC8785] Rundgren, A., Jordan, B., and S. Erdtman, "JSON Canonicalization Scheme (JCS)", RFC 8785, DOI 10.17487/RFC8785, June 2020, <https://www.rfc-editor.org/rfc/rfc8785>.

[RFC9110] Fielding, R., Ed., Nottingham, M., Ed., and J. Reschke, Ed., "HTTP Semantics", STD 97, RFC 9110, DOI 10.17487/RFC9110, June 2022, <https://www.rfc-editor.org/rfc/rfc9110>.

[RFC9449] Fett, D., Campbell, B., Bradley, J., Lodderstedt, T., Jones, M., and D. Waite, "OAuth 2.0 Demonstrating Proof of Possession (DPoP)", RFC 9449, DOI 10.17487/RFC9449, September 2023, <https://www.rfc-editor.org/rfc/rfc9449>.

[W3C-DID] Sporny, M., Longley, D., Sabadello, M., Reed, D., Steele, O., and C. Allen, "Decentralized Identifiers (DIDs) v1.0", July 2022.

[RFC6454] Barth, A., "The Web Origin Concept", RFC 6454, December 2011, <https://www.rfc-editor.org/rfc/rfc6454>.

[RFC6960] Santesson, S., Myers, M., Ankney, R., Malpani, A., Galperin, S., and C. Adams, "X.509 Internet Public Key Infrastructure Online Certificate Status Protocol - OCSP", RFC 6960, June 2013, <https://www.rfc-editor.org/rfc/rfc6960>.

[RFC7636] Sakimura, N., Ed., Bradley, J., and N. Agarwal, "Proof Key for Code Exchange by OAuth Public Clients", RFC 7636, September 2015, <https://www.rfc-editor.org/rfc/rfc7636>.

[RFC8176] Jones, M., Hunt, P., and A. Nadalin, "Authentication Method Reference Values", RFC 8176, June 2017, <https://www.rfc-editor.org/rfc/rfc8176>.

[RFC8414] Jones, M., Sakimura, N., and J. Bradley, "OAuth 2.0 Authorization Server Metadata", RFC 8414, June 2018, <https://www.rfc-editor.org/rfc/rfc8414>.

[RFC8693] Jones, M., Nadalin, A., Campbell, B., Bradley, J., and C. Mortimore, "OAuth 2.0 Token Exchange", RFC 8693, January 2020, <https://www.rfc-editor.org/rfc/rfc8693>.

[RFC8707] Campbell, B., Bradley, J., and H. Tschofenig, "Resource Indicators for OAuth 2.0", RFC 8707, February 2020, <https://www.rfc-editor.org/rfc/rfc8707>.

[RFC9068] Bertocci, V., "JSON Web Token (JWT) Profile for OAuth 2.0 Access Tokens", RFC 9068, October 2021, <https://www.rfc-editor.org/rfc/rfc9068>.

[FIPS-180-4]
National Institute of Standards and Technology, "Secure Hash Standard (SHS)", FIPS PUB 180-4, DOI 10.6028/NIST.FIPS.180-4, August 2015, <https://doi.org/10.6028/NIST.FIPS.180-4>.

24. Informative References

[RFC6749] Hardt, D., Ed., "The OAuth 2.0 Authorization Framework", RFC 6749, DOI 10.17487/RFC6749, October 2012, <https://www.rfc-editor.org/rfc/rfc6749>.

[MCP] Anthropic, "Model Context Protocol Specification", 2024.

[SP-800-207]
Rose, S., Borchert, O., Mitchell, S., and S. Connelly, "Zero Trust Architecture", NIST Special Publication 800-207, DOI 10.6028/NIST.SP.800-207, August 2020, <https://doi.org/10.6028/NIST.SP.800-207>.

[SP-800-63-4]
Temoshok, D., Fenton, J., Choong, Y., Lefkovitz, N., Regenscheid, A., and J. Richer, "Digital Identity Guidelines", NIST Special Publication 800-63-4, July 2025.

[RFC5011] StJohns, M., "Automated Updates of DNS Security (DNSSEC) Trust Anchors", RFC 5011, DOI 10.17487/RFC5011, September 2007, <https://www.rfc-editor.org/rfc/rfc5011>.

[TUF] The Update Framework Authors, "The Update Framework Specification", Version 1.0.26, URI https://theupdateframework.github.io/specification/latest/, 2024.

JSON Schema Index

The following JSON Schema definitions are normatively referenced throughout this specification. They are available in JSON Schema Draft 2020-12 format in the specification repository at aip-spec (https://github.com/provai-dev/aip-spec) under spec/v0.3/schemas/.

agent-identity.schema.json Defines the Agent Identity Object structure (Section 5.2).

capability-manifest.schema.json Defines the Capability Manifest structure (Section 5.3).

credential-token.schema.json Defines the Credential Token JWT

payload structure (Section 5.4).

principal-token.schema.json Defines the Principal Token JWT payload structure (Section 5.5).

registration-envelope.schema.json Defines the Registration Envelope request body structure (Section 5.6).

revocation-object.schema.json Defines the Revocation Object structure (Section 5.7).

endorsement.schema.json Defines the Endorsement Object structure (Section 5.8).

approval-envelope.schema.json Defines the Approval Envelope, Step, and Compensation Step structures (Section 13).

capability-overlay.schema.json Defines the Capability Overlay structure (Section 5.11).

engagement-object.schema.json Defines the Engagement Object structure including Participant, Approval Gate, and Change Log Entry (Section 5.12).

grant-request.schema.json Defines the Grant Request structure (Section 12.2).

grant-response.schema.json Defines the Grant Response structure (Section 12.4).

All schema files conform to JSON Schema Draft 2020-12. Implementations MUST validate objects against these schemas. Validation failures MUST result in rejection.

Acknowledgements

AIP builds directly on the work of the W3C DID Working Group, IETF OAuth Working Group, and NIST NCCoE AI Agent Identity and Authorization Concept Paper (2026).

Appendix A: Changes from Version 0.1

This appendix is informative.

This version addresses four operational and security additions relative to v0.1:

* *Section 4.5 - AIP-GRANT Principal Authorization Protocol.* Defines the standardised ceremony by which a human principal reviews, consents to, and cryptographically authorises an agent delegation. Analogous to the OAuth 2.0 Authorization Code Flow.

* *Section 4.6 - Chained Approval Envelopes.* Defines a workflow-level authorisation primitive. A single human approval covers a pre-declared sequence of dependent agent actions, including SAGA compensation steps. Addresses the cascading-approval problem and the token-expiry-while-pending problem for Tier 2 workflows.

* *Section 7.4 - Token Refresh and Long-Running Tasks.* Specifies when and how agents re-issue Credential Tokens, pre-emptive refresh requirements, and handling of delegation chain expiry.

* *Section 14 (revised) - Rate Limiting and Abuse Prevention.* Defines per-endpoint rate limit categories, required response headers, anti-abuse rules for registration and validation-driven lookups, and graduated backoff requirements.

Appendix B: Changes from Version 0.2

This appendix is informative.

v0.3 introduces the following changes relative to v0.2. Items marked *(breaking)* are not backward compatible.

Breaking changes:

* *(breaking) §3 - Tiers reframed as threat-model declarations.* Tier 1 (bounded-staleness), Tier 2 (real-time revocation), Tier 3 (enterprise/regulated). Each Tier now enumerates its security property declarations rather than just listing scopes.

* *(breaking) §3, §4.5.3, §7.2 - spawn_agents scope retired.* Replaced by spawn_agents.create and spawn_agents.manage, both classified as Tier 2 with 300s max TTL and DPoP requirement.

* *(breaking) §4.5 - Three-tier grant model.* AIP-GRANT restructured into G1 (Registry-Mediated), G2 (Direct Deployer), and G3 (Full Ceremony with OAuth 2.1). Registration Envelopes MUST include grant_tier. did:key principals forbidden for Tier 2.

* *(breaking) §6.2 - AIPRegistry service entry.* Principal DID Documents authorising Tier 2 agents MUST include an AIPRegistry service entry.

* *(breaking) §6.3.4 - Well-known schema extended.* The /.well-known/aip-registry response MUST include signature, registry_name, and endpoints fields.

* *(breaking) §7.1 - Version header.* X-AIP-Version updated from "0.2" to "0.3".

Additive changes:

* *§1.2 - MCP integration note.* Informative note clarifying AIP's relationship with Model Context Protocol.

* *§4.2.3 - New claims.* aip_approval_step (integer) and aip_engagement_id (string) added to Credential Token schema.

* *§4.2.4 - Principal Token claims.* acr and amr claims added for G3 identity proofing. iss field added.

* *§4.7 - Capability Overlays.* New section defining context-specific capability attenuation with Rules CO-1 through CO-6.

* *§4.8 - Engagement Objects.* New section defining multi-party engagement lifecycle, append-only change log, and approval gates.

* *§7.3 - Validation steps.* Step 6a (Registry Trust Anchoring), Step 6b (Engagement validation), and Step 9e (Overlay application) added.

* *§7.5 - Token Exchange.* New section for RFC 8693 token exchange with DPoP binding, enabling AIP-to-OAuth bridging for MCP.

* *§9.4 - RPNP.* Registry Push Notification Protocol for near-real-time event delivery (5s SLA).

* *§13.2 - Error codes.* 26 new error codes for all v0.3 features.

* *§14.6 - Backoff formula corrected.* BACKOFF_BASE separated from Retry-After; jitter bounded by BACKOFF_BASE.

* *§15.5 - OAuth 2.1 AS.* Authorization server interface for G3.

* *§15.6 - Scope Map.* urn:aip:scope: URI namespace and GET /v1/scopes endpoint.

* *§15.7 - Engagement Endpoints.* CRUD for Engagement Objects.

* *§15.8 - RPNP Endpoints.* Subscription management.

* *§ 16.1 - Tier Conformance table.* Maps Tiers to required security properties.

* *§ 17.1 - Threat scenarios.* TS-12 (orchestrator cascade), TS-13 (overlay injection), TS-14 (engagement tampering).

* *§ 19 - References.* 10 new normative references (RFC 6454, 6960, 7636, 8176, 8414, 8693, 8707, 9068, 9728; RFC 8785 already present). 1 new informative reference (OWASP LLM Top 10).

Author’s Address

Paras Singla
Independent
Email: paras.singla@inviscel.com
URI: https://provai.dev