Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: 15 November 2026

P. Singla
Independent

14 May 2026

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

draft-singla-agent-identity-protocol-02

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 15 November 2026.

Copyright Notice

Table of Contents

1. Introduction 7
1.1. Motivation 7
1.2. Design Philosophy 7
1.3. Architecture Layers 9
2. Conventions 10
2.1. Canonical JSON Serialization 10
3. Definitions and Terminology 11
3.1. Abbreviations 15
3.2. Architecture Tiers 16
4. Core Identity (did:aip) 17
4.1. AID Syntax 17
4.2. AID Derivation 17
4.3. AIP Namespace Catalog 18
4.4. Agent Identity Object Schema 18
4.5. Uniqueness and Immutability 19
5. Resource Model and Data Structures 19
5.1. Resource Naming 19
5.2. Agent Identity Object 20
5.3. Capability Manifest 22
5.4. Credential Token 23
5.4.1. Step Execution Token 25
5.5. Principal Token 28
5.6. Registration Envelope 31
5.7. Revocation Object 33
5.8. Endorsement Object 36
5.9. Field Constraints and Catalog-Bound Scope Identifiers . . 37
5.10. Delegation Rules 39
5.11. Capability Overlays 39
5.12. Engagement Objects 43
6. Registration Protocol 46
6.1. Envelope Submission 46
6.2. Registration Validation 46
6.3. Error Responses 50
7. Agent Resolution 50
7.1. DID Resolution 50
7.2. did:aip Method Resolution 51
7.3. DID Document Structure 53

7.4. Registry Genesis 54
7.4.1. Key Generation 54
7.4.2. Registry ID Establishment 54
7.4.3. Self-Registration Exemption 55
7.4.4. Registry Metadata Publication 55
7.4.5. Registry Trust Record 58
7.4.6. Single-Instance Constraint 60
7.4.7. Registry Key Rotation 60
7.5. AIP Gateway Protected Resource Metadata 62
8. Credential Tokens 64
8.1. Credential Token Transport and Version Header 64
8.2. Token Issuance 65
8.2.1. Token Cache Requirements for Stateless Deployments . 66
8.3. Token Refresh and Long-Running Tasks 67
8.3.1. Agent Self-Refresh 67
8.3.2. Pre-emptive Refresh Requirements 68
8.3.3. Delegation Chain Expiry 69
8.3.4. Interaction with Approval Envelopes 69
8.4. Token Exchange for MCP 70
8.4.1. Overview 70
8.4.2. Exchange Request 70
8.4.3. Exchange Validation 71
8.4.4. Exchange Response 72
8.4.5. Enterprise IdP Federation Profile 72
9. Credential and Step Execution Token Validation 76
9.1. Step 1: Parse 78
9.2. Step 2: Header Validation 78
9.3. Step 2a: Expiration Preflight 79
9.4. Step 3: Identify Agent and Retrieve Public Key 79
9.5. Step 4: Verify Signature 79
9.6. Step 5: Validate Claims 80
9.6.1. Step 5a: iat (Issued-At Time) 80
9.6.2. Step 5b: exp (Expiration Time) 80
9.6.3. Step 5c: Token Not Expired 80
9.6.4. Step 5d: aud (Audience) 80
9.6.5. Step 5e: jti (JWT ID) Replay Check 80
9.6.6. Step 5f: aip_version 81
9.6.7. Step 5g: Issuer and Subject Binding 81
9.7. Step 6: TTL Validation 81
9.8. Step 6a: Registry Trust Anchoring (Conditional) 82
9.9. Step 6b: Engagement Validation (Conditional) 83
9.10. Step 7: Revocation Check 83
9.11. Step 8: Delegation Chain Validation 84
9.11.1. Step 8 Post-Check A 87
9.11.2. Step 8 Post-Check B 87
9.11.3. Step 8 Post-Check C: Identity Proofing 87
9.12. Step 9: Capability Validation 88
9.13. Step 9a: Scope Verification 88

9.14. Step 9b: Capability Overlay (Conditional) 89
9.15. Step 9c: Delegation Scope Inheritance 89
9.16. Step 9d: Grant Tier Conformance 90
9.17. Step 10: DPoP Validation (Conditional) 90
9.18. Step 10a: Approval Envelope Step Verification (Conditional) 92
9.19. Step 11: Tier 3 Enterprise Checks (Conditional) 93
9.20. Step 12: Accept 94
9.21. Step Execution Token Validation Profile 94
10. Delegation 96
10.1. Delegation Chain 96
10.2. Capability Scope Rules 97
10.3. Delegation Validation 98
10.4. Delegated Identity Chaining for A2A Workflows 99
11. Revocation Management 100
11.1. Revocation Object 100
11.2. Revocation Submission Validation 101
11.3. Certificate Revocation List (CRL) 102
11.4. Revocation Checking 104
11.5. Registry Push Notification Protocol (RPNP) 105
11.5.1. Overview 105
11.5.2. Subscription 105
11.5.3. Push Event Payload 108
11.5.4. Delivery Guarantees 110
12. Principal Grant Ceremony (AIP-GRANT) 111
12.1. Overview and Roles 111
12.2. GrantRequest Object 112
12.3. Principal Wallet Consent Requirements 114
12.4. GrantResponse Object 118
12.5. Transport Bindings 120
12.6. Sub-Agent Delegation Flow 120
12.7. AIP-GRANT Error Codes 121
12.8. G1: Registry-Mediated Grant Flow 121
12.9. G2: Direct Deployer Grant Flow 123
12.10. G3: Full Ceremony Grant Flow (OAuth 2.1) 124
13. Approval Envelopes 127
13.1. Motivation 127
13.2. The Token-Expiry-While-Pending Problem 127
13.3. Approval Envelope Schema 127
13.4. Step Schema 130
13.5. Compensation Step Schema 132
13.6. Approval Envelope Lifecycle 133
13.7. Action Hash Computation 135
13.8. Step Claim and Execution Protocol 136
13.9. SAGA Compensation Semantics 139
13.10. Approval Envelope Validation Rules 140
14. Reputation and Endorsements 142
14.1. Endorsement Object 142

14.1.1. Endorsement Acceptance Requirements 143
14.2. Reputation Scoring 143
15. Lifecycle States 144
16. Principal Chain 145
17. Registry Interface 145
17.1. Endpoint Inventory 146
17.2. AID URL Encoding 149
17.3. Response Format 149
17.4. Agent Registration Metadata 150
17.5. Agent Key Rotation Endpoint 150
17.6. Agent Public-Key Endpoints 152
17.7. Agent Capability Manifest Endpoint 153
17.8. Agent Revocation Status Endpoint 153
17.9. Revocation Submission Endpoint 154
17.10. CRL Response 155
17.11. Agent Heartbeat Endpoint 156
17.12. Approval Envelope Endpoints 156
17.13. Resource Registration 159
17.14. OAuth 2.1 Authorization Server 160
17.14.1. Registry Metadata Additions 161
17.15. AIP Catalog Sync, Scope Map, and Namespace Map 163
17.16. Capability Overlay Endpoints 167
17.17. Engagement Endpoints 168
17.18. RPNP Subscription Endpoints 169
17.19. Key Management and Rotation 170
17.20. Pagination and Large Responses 170
18. Error Handling 170
18.1. Error Response Format 171
18.2. Standard Error Codes 172
18.3. Error Detail Types 177
19. Rate Limiting and Abuse Prevention 178
19.1. Rate Limit Response Format 178
19.2. Per-Endpoint Rate Limit Categories 179
19.2.1. Category R1 - Registration writes 179
19.2.2. Category R2 - Key rotation writes 180
19.2.3. Category R3 - Revocation writes 180
19.2.4. Category R4 - Validation-driven key reads 180
19.2.5. Category R5 - CRL reads 181
19.2.6. Category R6 - Endorsement writes 181
19.2.7. Category R7 - Approval Envelope writes and step claims 181
19.3. Registration Abuse Prevention 181
19.3.1. AID uniqueness enforcement 182
19.3.2. Principal delegation chain verification at registration 182
19.3.3. Registration flood from shared principals 182
19.3.4. Public Registry challenge for unauthenticated deployers 182

19.4. Validation-Driven Lookup Limits 183
19.4.1. Key version caching 183
19.4.2. Historical key depth limit 183
19.4.3. kid validation at the Relying Party 183
19.5. Approval Envelope Rate Limits 183
19.5.1. Envelope submission rate 183
19.5.2. Pending envelope limit 183
19.5.3. Step claim timeout 184
19.5.4. Compensation cascade depth 184
19.6. Graduated Backoff Requirements 184
19.7. Historical Key Retention Requirements 185
20. Versioning and Compatibility 185
20.1. Tier Conformance 187
21. Security Considerations 188
21.1. Threat Model 188
21.2. Cryptographic Requirements 190
21.3. Proof-of-Possession (DPoP) 191
21.4. Key Management 194
21.5. Token Security 196
21.6. Tier 3 mTLS Certificate Profile 196
21.7. Delegation Chain Security 197
21.8. Registry Security 198
21.9. Revocation Security 198
21.10. Approval Envelope Security 199
21.11. Privacy Considerations 199
22. JSON Schema and Catalog Artifact Index 199
23. Conformance 202
23.1. Common Requirements 202
23.2. Conformance Classes 202
23.3. Optional Feature Profiles 205
23.4. Conformance Testing Methodology 207
23.5. Conformance Claim Content 208
24. IANA Considerations 208
24.1. Non-IANA DID Method Registry 208
24.2. Existing HTTP and Discovery Registrations 210
24.3. AIP Scope Identifiers 210
24.4. AID Namespace Catalog 210
24.5. AIP Grant Tier Values 210
24.6. AIP Error Codes 211
24.7. Media Types 217
24.7.1. application/aip+jwt 217
24.7.2. application/aip-set+jwt 218
24.7.3. application/aip-crl+json 219
25. Normative References 221
26. Informative References 223
Acknowledgements 224
Appendix A: Implementation Considerations (Non-Normative) 224
Author’s Address 225

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 the following design principles and relationships:

Neutral and open. This document is submitted under BCP 78, BCP 79, and the IETF Trust Legal Provisions. Code Components extracted from this document are licensed under the Revised BSD License as described in those Legal Provisions. This document defines the did:aip DID method and requests its registration with the W3C DID Method Registry.

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 is designed to support the zero trust principles defined in [SP-800-207] and to provide protocol controls that can be used by deployments seeking alignment with NIST SP 800-63-4 [SP-800-63-4] AAL2 for agent-to-service interactions. This specification does not by itself assert SP 800-63-4 conformance, certification, or audit sufficiency. Any implementation claiming SP 800-63-4 or AAL2 conformance MUST maintain a separate conformance mapping covering its concrete authenticators, identity proofing process, verifier behavior, key management, logging, and operational controls. No agent is implicitly trusted by virtue of its origin, network location, or prior successful interaction. Every interaction MUST execute the validation algorithm in Section 9; Registry-dependent checks are performed according to the token's Tier, revocation-checking mode, and cache rules. 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 as specified in Section 11.4. 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, supporting the anti-replay objectives of [SP-800-207] Section 3.

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 principal-approved capability set or, for delegated agents, the delegating parent's effective Capability Manifest. 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. An AIP-authenticated agent can obtain scoped access tokens for MCP servers via the token exchange mechanism defined in Section 8.4 and the OAuth 2.1 Authorization Server profile in Section 17.14.

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 revoked via full_revoke (Section 11.1). Tier 2 and Tier 3 revocation checks fail closed against live Registry state; Tier 1 rejection is bounded by the CRL freshness SLA, while child materialisation and replica convergence are bounded by the revocation propagation requirements in Section 11. AIP does not prevent the initial injection - it limits the attacker's persistence.

1.3. Architecture Layers

AIP is structured as six ordered layers:

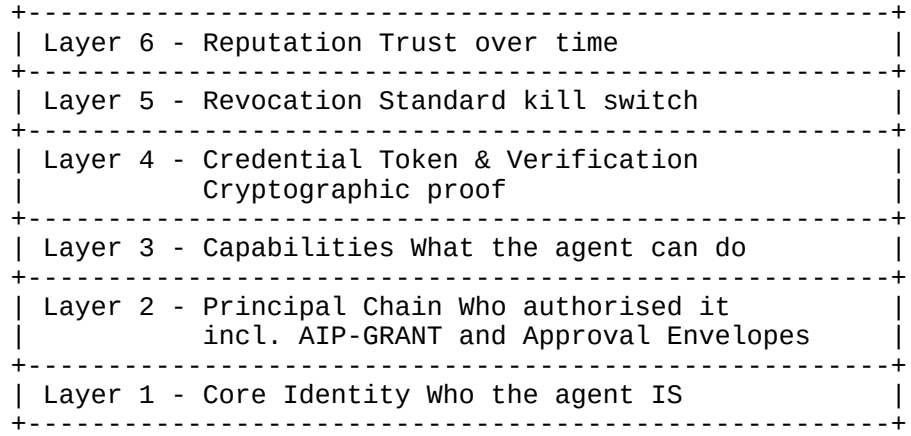


Figure 1: AIP Architecture Layers

2. Conventions

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, and the other rules 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, AIP first constructs the object-specific signing input described below, then serializes that signing input with the JSON Canonicalization Scheme (JCS) defined in [RFC8785]. The following 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 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 "".

3 Serialize the resulting signing-input JSON value using JCS [RFC8785]. Array element order MUST be preserved; arrays MUST NOT be sorted.

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

Every object type signed with this procedure MUST define a top-level signature member in its schema or in the object-specific signing profile. If an unsigned object instance is being prepared for signing and the top-level signature member is absent, the signer MUST inject that member with value "" before canonicalization. Verifiers MUST reconstruct the same signing input by replacing the received signature value with "". Object types that do not define a signature member MUST NOT be signed with this procedure unless a later profile explicitly defines the signature member name and signing input.

The serialization step is JCS. The preceding signature-field replacement or injection is AIP-specific signing-input normalization and is not part of RFC 8785 itself. Implementations SHOULD use an RFC8785-conformant library to ensure serialization 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}
```

This canonical serialization procedure applies to Capability Manifests, Agent Identity Objects, Revocation Objects, Endorsement Objects, GrantRequest Objects when signed, and Approval Envelopes. It MUST NOT be used for JWT-format objects, including Credential Tokens, Principal Tokens, Step Execution Tokens, and RPNP push notifications. JWT-format objects MUST use standard JWS compact serialization per [RFC7515].

3. Definitions and 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 cryptographically derived, persistent identifier for an agent conforming to the did:aip DID method defined in Section 4.1. An AID is a W3C DID. A standalone did:aip string does not encode its authoritative Registry; resolution requires Registry context as defined in Section 7.

Decentralized Identifier (DID): A W3C Decentralized Identifier. DID Documents, DID URLs, and DID resolution are used as defined by W3C DID Core.

Production Deployment: Any AIP Registry or Relying Party deployment where more than one process, container, or server instance may validate tokens for the same AID simultaneously, or where the deployment serves external principals not under the operator's direct control. Deployments that are explicitly single-instance and serve only internal development or testing purposes are not production deployments for the purposes of Section 8.2.1.

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, Principal Wallet references use this capitalized term unless the context explicitly describes a generic wallet ecosystem.

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 (SET): A short-lived Registry-issued JWT,

distinct from an agent-issued Credential Token, attesting that a specific Approval Envelope step has been claimed by its designated actor. Defined in Section 13.8 and validated using the Step Execution Token profile in Section 9.

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 or Step Execution Token. Defined in Section 5.5.

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

Authorization Server (AS): The OAuth 2.1 authorization server role. In G3 grant flows, the Registry commonly acts as the AS for authorization-code and token endpoints.

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: An operation security classification derived from the highest-risk requested scope in the synced AIP Scope Catalog. Tier determines the applicable revocation-checking mode, maximum Credential Token lifetime, and mandatory security mechanisms. Three Tiers are defined in Section 3.2. Tier selection on performance or availability grounds alone, without using the synced catalog metadata, 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. Grant Tier is a ceremony-assurance axis; it is distinct from the security Tier derived from an operation's requested scopes.

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): An AIP signed JSON revocation artifact published by the Registry for Tier 1 bounded-staleness validation. Tier 2 and Tier 3 validation use live Registry revocation status lookups; they do not satisfy their real-time revocation requirement by fetching an on-demand CRL. Defined in Section 11.3.

DPoP Proof: A Demonstration of Proof-of-Possession as defined in [RFC9449]. Mandatory for Tier 2 and Tier 3 operations, and also mandatory for any Tier 1 scope or endpoint that explicitly requires DPoP. Binds a Credential Token to a specific HTTP request and the agent's key material.

Authentication Assurance Level 2 (AAL2): A NIST SP 800-63 assurance level for authentication strength. When this specification refers to AAL2, it means AAL2 as defined by the cited NIST SP 800-63 document.

JWK, JWS, and JWT: JSON Web Key, JSON Web Signature, and JSON Web

Token, respectively. AIP uses JWK for public key representation, JWS compact serialization for signed JWTs, and JWT payloads for Credential Tokens, Principal Tokens, Step Execution Tokens, and RPNP push notifications.

Lowercase Hexadecimal Encoding (LCHEX): The lowercase hexadecimal representation of a byte string, using characters 0-9 and a-f only.

Registry Push Notification Protocol (RPNP): The optional Registry push protocol for revocation notifications, defined in Section 11.5.

Enterprise Identity Provider (Enterprise IdP): An OAuth 2.0/2.1 or OIDC-compliant authorization server operated by an enterprise organisation, such as Microsoft Entra ID, Okta, Ping Identity, or an equivalent system, that manages human principal identities, enforces enterprise access policies including Conditional Access and ABAC, and issues access tokens scoped to enterprise resources. In AIP, an Enterprise IdP is the authorization server that a Relying Party trusts for resource access tokens, distinct from the AIP Registry acting as an OAuth AS for AIP-scope token exchange. Registries identify an Enterprise IdP by the presence of the enterprise_idp_required flag on a registered resource record (see Section 8.4.5).

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. Abbreviations

The following common abbreviations are used with their ordinary industry meanings or with the referenced external specifications:

ABAC Attribute-Based Access Control.

BOM Byte Order Mark. AIP canonical JSON signing input uses UTF-8 without a leading BOM.

CDN Content Delivery Network.

DAG Directed Acyclic Graph.

FIDO2/WebAuthn FIDO2 and Web Authentication.

HSM Hardware Security Module.

HMAC Hash-based Message Authentication Code.

JCS JSON Canonicalization Scheme, defined by [RFC8785].

JWKS JSON Web Key Set, the JWK container format used by OAuth and OIDC providers to publish signing keys.

MITM Man-in-the-Middle.

mTLS Mutual TLS.

OCSP Online Certificate Status Protocol, defined by [RFC6960].

PII Personally Identifiable Information.

PKCE Proof Key for Code Exchange, defined by [RFC7636].

SAGA Distributed-workflow compensation pattern.

SLA Service-Level Agreement.

TTL Time To Live.

3.2. Architecture Tiers

AIP defines three security Tiers that map synced AIP Scope Catalog metadata to mandatory protocol behaviors. Tiers are NOT performance classes and are not locally chosen labels; they are derived from the highest-risk requested scope.

* *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 certificate revocation checking using OCSP or an equivalent deployment-profile mechanism defined in Section 21.6.

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 cryptographically derived, persistent identifier conforming to the W3C Decentralized Identifier (DID) standard [W3C-DID]. This document defines the AIP DID method did:aip and provides non-IANA W3C DID Method Registry registration information in Section 24.1.

4.1. AID Syntax

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

```
AID        = "did:aip:" namespace ":" agent-id
namespace  = LOALPHA *( LOALPHA / DIGIT )
             *( "-" 1*( 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 with optional single-hyphen-separated segments identifying the agent type. Each hyphen MUST be followed by one or more lowercase alphanumeric characters. Concrete namespace registrations are maintained in the AIP Namespace Catalog (Section 17.15).

* agent-id: A 32-character hexadecimal string (lowercase) computed as the leftmost 128 bits (first 16 octets) of 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 digest.

4. Take the leftmost 16 octets of that digest and hex-encode them (lowercase) to form the 32-character 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 corresponding to the registered public key is both necessary and sufficient to prove control of the AID.

4.3. AIP Namespace Catalog

Standard and community AID namespace registrations are defined by the immutable AIP Catalog Snapshot identified in Section 17.15 and exposed by conformant Registries through the Namespace Catalog endpoint defined there. The Draft-02 Catalog Snapshot contains the standard namespace entries for personal, enterprise, service, orchestrator, ephemeral, and registry agents.

Namespace registration metadata is normative for namespace-level lifecycle rules. If a namespace entry sets requires_task_id: true, Principal Tokens for agents in that namespace MUST contain a non-empty task_id. If a namespace entry sets reserved: true, the namespace MUST NOT be registered through the normal POST /v1/agents flow.

Namespaces are immutable once registered. Custom or community namespaces MUST be registered in the AIP Namespace Catalog before they are treated as interoperable AIP namespaces.

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 unique within the authoritative Registry namespace and is cryptographically derived from the agent public key. Once an agent is registered in a Registry, its AID is immutable. An AID cannot be reused or transferred within that Registry. Two agents with identical public keys will derive identical AIDs, so the cryptographic relationship is deterministic. If two different public keys derive the same AID, the Registry MUST treat the later registration as an AID collision and reject it as a duplicate AID. If a registration attempt presents a public key that is already associated with a registered, non-revoked AID in that Registry, the Registry MUST reject the attempt with aid_already_registered.

5. Resource Model and Data Structures

This section defines the core data structures in AIP: JSON Schema representations of agents, principals, capabilities, tokens, and registration envelopes. Non-JWT signed JSON objects use the canonical JSON serialization rules of Section 2.1 unless a specific object defines a narrower detached-signature target.

5.1. Resource Naming

Agent Identities (AIDs) are W3C Decentralized Identifiers [W3C-DID] using the did:aip method. AID syntax is normatively defined by the AID, namespace, and agent-id ABNF productions in Section 4.1. This section does not define a second AID grammar. JSON Schemas and implementations MUST use grammar equivalent to Section 4.1.

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.

Concrete namespace values are maintained in the AIP Namespace Catalog (Section 17.15), not in this prose table. The namespace grammar from Section 4.1 remains the wire-format constraint; catalog membership supplies the interoperability and lifecycle metadata for each namespace.

Any namespace marked reserved: true in the AIP Namespace Catalog MUST NOT be registered via the standard POST /v1/agents endpoint. The standard registry namespace is reserved and 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-a716-446655440000
Revocation Object	rev:	rev:6ba7b810-9dad-41d1-80b4-00c04fd430c8
Endorsement Object	end:	end:6ba7b811-9dad-41d1-80b4-00c04fd430c8
Grant Request	gr:	gr:550e8401-e29b-41d4-a716-446655440000
Approval Envelope	apr:	apr:550e8402-e29b-41d4-a716-446655440000

Table 1: Typed Identifier Prefixes

5.2. Agent Identity Object

The Agent Identity Object is the canonical signed JSON document that establishes an agent’s identity. The key-continuity signing target for key rotation is defined by the previous_key_signature requirements below.

aid Type: string. Required: REQUIRED. Constraints: MUST match

did:aip ABNF; pattern is lowercase namespace plus a 32-character lowercase hexadecimal agent-id derived from the leftmost 128 bits of the SHA-256 digest of the agent public key.

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 for Tier 1, SHOULD be present for Tier 2, REQUIRED for Tier 3. Constraints: pattern sha256:<64 lowercase hex characters>.

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 computed by the retiring previous private key over the new full Agent Identity Object, serialized using the Section 2.1 canonical JSON procedure with previous_key_signature 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.

* The model.attestation_hash field is the protocol binding between the registered agent identity and a model binary or version manifest. It SHOULD be present for Tier 2 deployments and MUST be present for Tier 3 deployments. A missing hash means the AID is not cryptographically pinned to a specific model artifact.

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

* For key rotation, the previous_key_signature value MUST be generated with the retiring previous private key, not the new private key. The signing target is the complete new Agent Identity Object after applying the new version and public_key, serialized per Section 2.1 with previous_key_signature temporarily set to "". The computed signature then replaces that empty string in the stored Agent Identity Object. A Registry MUST verify this signature with the public key from the immediately previous Agent Identity Object version before accepting the rotation.

5.3. Capability Manifest

The Capability Manifest is a versioned, signed JSON document that declares the specific permissions granted to an agent. The signature field is computed using the Section 2.1 canonical JSON procedure with signature set to "". The signature_kid field identifies the Ed25519 verification method used for the signature.

Field	Type	Required	Constraints
manifest_id	string	REQUIRED	pattern: cm:<UUIDv4-lowerhex>
aid	string	REQUIRED	MUST match did:aip ABNF
granted_by	string	REQUIRED	MUST be a valid W3C DID matching did:<method>:<method-specific-id>
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_kid	string	REQUIRED	DID URL controlled by granted_by
signature	string	REQUIRED	base64url EdDSA signature; pattern: ^[A-Za-z0-9_-]+$

Table 2: 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 identified by signature_kid before trusting any capability declared within it. The DID portion of signature_kid MUST be byte-for-byte equal to granted_by.

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]. Credential Tokens conforming to this specification MUST be signed with EdDSA using the registered Ed25519 key identified by the JOSE kid. Credential Tokens signed with ES256, RS256, symmetric algorithms, none, or non-Ed25519 key material MUST be rejected.

JWT Header fields:

Field	Requirement	Value / Constraints
typ	MUST	"AIP+JWT"
alg	MUST	"EdDSA"
kid	MUST	DID URL identifying the signing key; MUST match the kid in the signing agent's Agent Identity

Table 3: JWT Header Fields

Credential Token Payload fields:

aip_version string, REQUIRED. AIP protocol compatibility version. MUST be "0.3" for tokens conforming to this specification. See Section 20 for version semantics and Section 9.6.6 for validation.

iss string, REQUIRED. MUST match the did:aip ABNF; MUST equal the AID identified by the JWT header kid; MUST equal sub; MUST equal sub of the last aip_chain element.

sub string, REQUIRED. MUST match the did:aip ABNF; MUST equal iss. For an agent-issued Credential Token, this is the acting leaf agent's AID. This specification does not define an on-behalf-of subject distinct from the signing agent.

aud string or array, REQUIRED. Single string or array; MUST include the Relying Party's identifier.

iat integer, REQUIRED. Unix timestamp; MUST NOT be in the future, with a 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; unique per issuer AID across all delegation chains.

aip_scope array, REQUIRED. minItems: 1; uniqueItems: true; each item matches <dot-separated-scope-name>.

aip_chain array, REQUIRED. minItems: 1; maxItems: 11; each element

is a compact-serialised signed Principal Token JWT. Elements are ordered root-to-leaf, and the last element's sub is the acting agent AID. The maximum length corresponds to delegation depths 0 through 10.

aip_principal_assertion string, OPTIONAL for Tier 1 and Tier 2; REQUIRED for Tier 3. A compact-serialised signed JWT issued by an enterprise Identity Provider, such as an OIDC ID Token or equivalent JWT-formatted assertion. The assertion MUST contain a sub claim identifying the human principal. When present, the Relying Party MUST verify the assertion's signature against the IdP's JWKS endpoint before treating the human identity context as valid. The sub claim in aip_principal_assertion MUST match the root Principal Token's principal.id value in aip_chain[0]. The issuer (iss) of this JWT MUST NOT be the AIP Registry. See Section 9.19 Steps 11a, 11b, and 11c for normative validation.

aip_originator_aid string, OPTIONAL; MUST be present when the agent is acting in a delegated agent-to-agent (A2A) workflow as specified in Section 10.4. The value is the AID of the originating agent that initiated the A2A workflow. When present, it MUST be a valid did:aip AID conforming to the ABNF in Section 4.1 and MUST NOT equal the sub claim of this Credential Token.

aip_registry string, OPTIONAL. Advisory URI of the AIP Registry; if present, Step 6a trust anchoring is REQUIRED.

aip_engagement_id string, OPTIONAL. Pattern: eng:<UUIDv4-lowerhex>; present when the token is scoped to an Engagement Object.

5.4.1. Step Execution Token

A Step Execution Token (SET) is a compact JWT issued by a Registry after a step actor successfully claims an Approval Envelope step. It is not an agent-issued Credential Token. Its issuer is the Registry ID, and its subject is the actor AID that claimed the step.

SET JWT Header fields:

Field	Requirement	Value / Constraints
typ	MUST	"AIP-SET+JWT"
alg	MUST	"EdDSA"
kid	MUST	HTTPS URI identifying a Registry step-execution verification key listed in the Registry Trust Record version current at issuance time
aip_trv	MUST	Integer Registry Trust Record version whose active_verification_keys.step_execution contains kid

Table 4: Step Execution Token Header Fields

The aip_trv header is a protected JOSE header value used only to select an already pinned Registry Trust Record. It is not a trust anchor. Relying Parties MUST NOT accept a Step Execution Token unless they already have pinned trust state for iss and for the indicated Registry Trust Record version.

SET Payload fields:

Field	Type	Required	Constraints
aip_version	string	REQUIRED	AIP protocol compatibility version; MUST be "0.3" for this spec
iss	string	REQUIRED	Registry ID HTTPS URI; MUST match the issuing Registry Trust Record
sub	string	REQUIRED	AID of the claimed step actor; MUST match the referenced Approval Envelope step actor
aud	string/	REQUIRED	Single string or

	array		array; MUST include the Relying Party's identifier
iat	integer	REQUIRED	Unix timestamp; MUST NOT be in the future
exp	integer	REQUIRED	Unix timestamp; MUST be strictly greater than iat; MUST NOT exceed the Step Execution Token TTL limit defined below
jti	string	REQUIRED	UUID v4 canonical lowercase; unique per Registry issuer across all SETs
aip_scope	array	REQUIRED	Scopes authorised for the claimed step
aip_chain	array	REQUIRED	Principal Token chain for the actor AID in sub
aip_registry	string	OPTIONAL	If present, MUST equal iss
aip_approval_id	string	REQUIRED	Approval Envelope identifier; pattern: apr:<UUIDv4-lowerhex>
aip_step_kind	string	REQUIRED	Either forward or compensation
aip_approval_step	integer	CONDITIONAL	1-based step_index; REQUIRED when aip_step_kind is forward; MUST be absent for compensation SETs
aip_compensation_step	integer	CONDITIONAL	1-based compensation_index; REQUIRED when aip_step_kind is compensation; MUST be

			absent for forward SETs
aip_engagement_id	string	OPTIONAL	Engagement Object identifier when the parent Approval Envelope is engagement-scoped

Table 5: Step Execution Token Payload Fields

The Registry MUST derive a Step Execution Token TTL limit as the minimum of: (1) the Registry's step_claim_timeout_seconds value published in /v1/registry-metadata; (2) the lowest ttl_max_seconds value for any requested aip_scope in the synced AIP Scope Catalog; and (3) the Tier ceiling for the highest-risk requested scope from Section 8.2. The Registry MUST set exp no later than iat + effective_set_ttl_limit. A Relying Party validating a SET MUST compute the same scope and Tier limit through Section 9.7 and reject a SET whose exp - iat exceeds that limit with invalid_token. A Registry completing or failing a step MUST also enforce the stored claim deadline, even if the SET has not yet expired.

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 or Step Execution Token.

A Principal Token MUST be a compact-serialised JWT signed using EdDSA. Its JOSE header is outside the payload schema but is normative: typ MUST be "JWT", alg MUST be "EdDSA", and kid MUST be a DID URL identifying an Ed25519 verification method controlled by the token issuer identified by iss. For the root Principal Token (delegation_depth 0), the issuer is the root principal.id. For delegated Principal Tokens, the issuer is the parent agent AID in delegated_by, and that parent AID MUST be registered in the authoritative AIP Registry. Relying Parties resolve delegated Principal Token signing keys through the Registry historical public-key endpoint and verify that the returned key was valid at the Principal Token's issued_at instant. Principal Tokens signed with ES256, RS256, symmetric algorithms, none, or non-Ed25519 key material MUST be rejected. X25519 verification methods MUST NOT be used for Principal Token signing.

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; MUST NOT equal sub
delegation_depth	integer	REQUIRED	minimum: 0, maximum: 10; MUST equal the array index of this token in aip_chain
max_delegation_depth	integer	OPTIONAL	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; signed consent/ audit metadata only; MUST NOT be used as an authorization constraint
task_id	string/ null	OPTIONAL; REQUIRED when the subject namespace catalog entry has requires_task_id: true	minLength: 1, maxLength: 256 when non-null; enforced at registration and during Step 8 chain validation
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:.

For a root Principal Token produced by AIP-GRANT, issued_at is the issuer's current UTC signing time and expires_at equals issued_at plus the GrantResponse approved_delegation_valid_for_seconds value. The detailed derivation rule is defined in Section 12.3.

The purpose field, when present, is a signed human-readable consent and audit claim. It is not a machine-enforced authorization constraint. Relying Parties MAY display or log purpose for operator context, but MUST NOT use it to grant capabilities, satisfy scope checks, or override Capability Manifest, Capability Overlay, Approval Envelope, or policy evaluation.

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); for direct registration delegation_depth MUST be 0 and delegated_by MUST be null; for sub-agent registration delegation_depth MUST be greater than 0 and delegated_by MUST identify the registered parent AID
grant_tier	string	REQUIRED	enum: ["G1", "G2", "G3"]; MUST meet the minimum Grant Tier required by the highest synced AIP Scope Catalog tier in capability_manifest

Table 7: Registration Envelope Fields

A Registration Envelope registers exactly one agent: the AID in identity.aid. The submitted principal_token payload sub MUST equal that AID. Direct principal-to-agent registrations use a root Principal Token with delegation_depth: 0 and delegated_by: null. Sub-agent registrations use a delegated Principal Token signed by the registered parent agent identified in delegated_by; the Registry reconstructs the complete principal chain from the parent’s stored registration chain and the submitted token during Registration Check 9.

5.7. Revocation Object

Revocation is performed by submitting a signed Revocation Object to POST /v1/revocations. The signature field is computed using the Section 2.1 canonical JSON procedure with signature set to "". The signing input order is revocation_id, target_id, type, issued_by, kid, reason, timestamp, propagate_to_children, scopes_revoked, and signature.

Field	Type	Required	Constraints
revocation_id	string	REQUIRED	pattern: rev:<UUIDv4-lowerhex>
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	Issuer DID authorised for the target, or Registry ID for Registry-generated revocations
kid	string	REQUIRED	Verification key identifier; DID URL controlled by issued_by for external submissions, or active Registry CRL key for Registry-generated revocations
reason	string	REQUIRED	enum defined in the Revocation Reason Codes table
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 authorisation of a target agent (via AID) or all agents under their authority (via Principal DID). Existing Credential Tokens and Step Execution Tokens depending on that authorisation become invalid immediately. This validity effect is independent of propagate_to_children; that flag controls only whether descendant Revocation Objects are materialised.

Revocation Reason Codes: The reason field MUST be one of the following values:

Reason	Semantics
device_compromised	The principal's device or agent host holding relevant key material was compromised.
key_compromised	The agent private key or signing key was directly compromised.
task_complete	The agent completed its assigned task and no longer requires the delegated authority.
policy_violation	The agent or deployment violated Registry, principal, or relying-party policy.
principal_request	The principal requested revocation without a more specific protocol reason.
account_closure	The principal or deployer account associated with the delegation is being closed.
heartbeat_timeout	Registry-generated reason for Dead Man's Switch expiry.
lifecycle_expired	Registry-generated reason for namespace lifecycle expiry.
parent_revoked	Registry-generated reason for recursive child revocation after a parent revocation.
other	Other auditable reason. Implementations SHOULD record additional detail in local audit metadata.

Table 9: Revocation Reason Codes

Externally submitted Revocation Objects MUST NOT use parent_revoked, heartbeat_timeout, or lifecycle_expired; those values are reserved for Registry-generated Revocation Objects.

5.8. Endorsement Object

Any Relying Party or agent MAY submit a signed Endorsement Object to the Registry after a completed interaction. The signature field is computed using the Section 2.1 canonical JSON procedure with signature set to "".

endorsement_id string, REQUIRED. Pattern: end:<UUIDv4-lowerhex>.

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. One of success, partial, or failure.

score number, REQUIRED. Endorsement score in the range 1 through 5 inclusive. Values outside this range MUST be rejected with endorsement_invalid.

notes string or null, OPTIONAL. Maximum length 512. Signed audit context only; MUST NOT affect reputation scoring or validation.

timestamp string, REQUIRED. ISO 8601 UTC timestamp.

signature_kid string, REQUIRED. DID URL identifying an Ed25519 verification method controlled by from_aid.

signature string, REQUIRED. Base64url EdDSA signature by from_aid.

The Registry MUST verify every submitted Endorsement Object signature. The DID portion of signature_kid MUST equal from_aid. Only success and partial outcomes increment endorsement_count. failure increments incident_count.

The notes field, when present, is signed human-readable audit context about the observed task outcome. The Registry MUST preserve it as part of the Endorsement Object, but MUST NOT parse, classify, or use notes to compute reputation scores, increment counters, validate capability authorization, or alter endorsement acceptance. Reputation effects are determined by the structured fields defined in this section and Section 14.

5.9. Field Constraints and Catalog-Bound Scope Identifiers

The Capability Manifest capabilities object is a closed object: top-level capability families not listed below MUST NOT appear unless a Registry accepts them through an explicit private extension policy. The immutable machine-readable schema for this draft is https://provai.dev/schemas/aip/capability-manifest/draft-02, also listed as capability-manifest.schema.json in the JSON Schema Index.

email Permitted sub-fields are boolean read, write, send, and delete, plus optional integer max_recipients_per_send in the range 1-100. Boolean fields default to false when absent. The recipient cap applies only when send is true.

calendar Permitted sub-fields are boolean read, write, and delete. Boolean fields default to false when absent.

filesystem Permitted sub-fields are read and write arrays of absolute path strings, plus boolean execute and delete. Each path string MUST be non-empty and no longer than 512 characters. Empty or absent path arrays mean deny-all for that operation. File deletion is permitted only for paths also permitted by write.

web Permitted sub-fields are boolean browse, forms_submit, and download, plus optional integer max_requests_per_hour in the range 1-10000. Boolean fields default to false when absent.

transactions The enabled boolean is REQUIRED when the transactions object is present. When enabled is true, max_single_transaction, max_daily_total, and currency are REQUIRED; transaction limits MUST be greater than zero; currency MUST be an ISO 4217 three-letter uppercase currency code. Optional require_confirmation_above MUST be less than or equal to max_single_transaction when both are present.

communicate The enabled boolean is REQUIRED when the communicate object is present. Permitted channel fields are boolean whatsapp, telegram, sms, and voice. When enabled is true, at least one channel field MUST be explicitly set to true; when enabled is false, channel fields have no granting effect.

spawn_agents The enabled boolean is REQUIRED when the spawn_agents object is present. When enabled is true, integer max_concurrent in the range 1-100 is REQUIRED. Optional types_allowed lists namespace labels the agent may spawn; each value MUST be an active, spawnable entry in the synced AIP Namespace Catalog unless a private namespace policy explicitly permits it.

registry Permitted sub-fields are Registry control-plane grants. Draft-02 defines boolean heartbeat, which grants the registry.heartbeat scope for authenticated liveness submissions.

approvals Permitted sub-fields are Approval Envelope control-plane grants. Draft-02 defines boolean create, which grants the approvals.create scope for submitting Approval Envelopes to POST /v1/approvals.

An absent top-level capability family grants no capability in that family. An empty capabilities object grants no capabilities. Implementations MUST NOT treat absent fields as allow-all.

Scope Identifier Grammar and Catalog Binding:

Scope identifiers are dot-separated lowercase ASCII strings. Each segment MUST begin with a-z or _, and subsequent characters in the same segment MAY be a-z, 0-9, or _. The normative pattern is <dot-separated-scope-name>. This permits versioned or tiered scope names such as transactions.v2 or filesystem.read.tier1 while preserving a non-numeric first character in every segment.

Concrete scope identifiers and their security metadata are defined by the immutable AIP Catalog Snapshot identified in Section 17.15 and exposed by conformant Registries through the Scope Map endpoint defined in Section 17.15. A scope is interoperable only when it is present with status: "active" or status: "experimental" in the synced AIP Scope Catalog, or when a Registry explicitly documents a private local extension policy.

Tokens containing a scope that is absent from the synced AIP Scope Catalog, marked reserved, marked removed, or outside an explicitly documented local extension policy MUST be rejected with invalid_scope.

Where this document references transactions.* or communicate.*, the wildcard is category notation rather than a literal scope string. The matching semantics for scope-family entries are defined by the AIP Scope Catalog. In the Draft-02 catalog, transactions.* includes the bare transactions scope, any active scope beginning with transactions., and capabilities.transactions.enabled: true in the Capability Manifest; communicate.* includes active scopes beginning with communicate. and 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. The hard protocol maximum for max_delegation_depth is 10.

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. Because depths 0 through 10 are the complete allowed range, a valid aip_chain contains at most 11 elements.

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). For non-root delegation tokens, the signing AID MUST be a registered AIP agent whose public key can be resolved from the Registry as specified by Credential Token validation Step 8d-2.

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:<UUIDv4-lowerhex>
aid	string	REQUIRED	The target agent’s AID
engagement_id	string	OPTIONAL	Pattern: eng:<UUIDv4-lowerhex>; 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_kid	string	REQUIRED	DID URL controlled by issued_by
signature	string	REQUIRED	Base64url EdDSA signature by issued_by over the Section 2.1 signing input

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. Constraint comparison is recursive for nested objects. An omitted overlay field inherits the effective base value and MUST NOT be interpreted as removing or relaxing the base constraint.

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. The DID portion of signature_kid MUST be byte-for-byte equal to issued_by.

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. A submission whose version is less than or equal to the current active overlay version MUST be rejected with overlay_version_conflict.

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, the following type-specific rules are normative. Scope set to false removes that scope through the boolean rule below.

The intersect operation is defined recursively over JSON values:

Objects For each object member present in the base value, if the overlay omits that member, the effective value is the base member unchanged. If the overlay supplies that member, apply intersect recursively. If the overlay supplies a member that is absent from the base value, the overlay is valid only when this specification or the synced AIP Scope Catalog defines absence of that member as

an unrestricted allowance or default value that the overlay narrows. Otherwise, the overlay expands the manifest and MUST be rejected with overlay_exceeds_manifest.

Booleans Boolean fields use logical AND. false is more restrictive than true.

Numbers Numeric fields that represent maximum allowances, rate limits, budget caps, concurrency caps, or confirmation thresholds use min(base, overlay). In the Draft-02 standard Capability Manifest schema, this rule applies to email.max_recipients_per_send, web.max_requests_per_hour, transactions.max_single_transaction, transactions.max_daily_total, transactions.require_confirmation_above, and spawn_agents.max_concurrent. A lower require_confirmation_above value is more restrictive because it requires explicit confirmation for a larger set of transactions. If the base manifest omits an optional numeric cap whose absence is defined as "no cap" or "no confirmation threshold", an overlay MAY add that cap and the effective value is the overlay value. Numeric fields whose ordering semantics are not defined by the synced AIP Scope Catalog or this specification MUST NOT be narrowed by ordering; the overlay value MUST equal the base value or be rejected with overlay_exceeds_manifest.

Arrays Arrays of allowed values, including filesystem path arrays and spawn_agents.types_allowed, use set intersection. The result MAY be an empty array, which means deny-all for that allowed-value list. If the base manifest omits an optional allowed-value array whose absence is defined as all catalog-valid values being allowed, an overlay MAY add the array and the effective value is the overlay array.

Strings and enums String and enum fields are not ordered. If an overlay supplies a string or enum value, it MUST be byte-for-byte equal to the base value unless the synced AIP Scope Catalog defines explicit subset semantics for that field. Otherwise the overlay MUST be rejected with overlay_exceeds_manifest. For example, transactions.currency set to "GBP" cannot attenuate a base transactions.currency value of "USD". If the overlay omits transactions.currency, the base currency is inherited unchanged.

Null and unknown types null and any value type not covered above MUST NOT be used to relax or erase a base constraint. A non-equal overlay value of an unknown type MUST be rejected with overlay_exceeds_manifest.

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.

Engagement Object Fields:

engagement_id (string; REQUIRED)
Pattern: eng:<UUIDv4-lowerhex>

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

hiring_operator_kid (string; REQUIRED)
DID URL identifying the Ed25519 verification method for hiring_operator_signature

deploying_principal_signature (string; COND.)
Base64url EdDSA countersignature by deploying_principal over the same top-level engagement signing input defined below; REQUIRED for active, suspended, completed, and terminated Engagement Objects

deploying_principal_kid (string; COND.)
DID URL identifying the Ed25519 verification method for deploying_principal_signature; REQUIRED whenever deploying_principal_signature is present

version (integer; REQUIRED)
Positive integer; MUST equal max(change_log.seq)

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:<lowercase-token>), name (REQUIRED, maxLength: 128), required_approver (REQUIRED, DID), trigger (REQUIRED, scope:<scope-string> or action_type:<action-type>), status (REQUIRED, one of: "pending", "approved", "rejected"), approved_at (OPTIONAL, ISO 8601).

Top-level engagement signature input: Both hiring_operator_signature and, when present, deploying_principal_signature MUST be computed over the same JCS-canonical JSON serialization of the immutable engagement agreement object containing only engagement_id, title, hiring_operator, deploying_principal, created_at, and expires_at. The top-level signing input MUST NOT include mutable lifecycle or roster state (status, participants, approval_gates, change_log, or version) and MUST NOT include top-level signature or *_kid fields. Mutable state is authenticated by the signed append-only change_log entries. 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), actor_kid (REQUIRED DID URL), payload (OPTIONAL, object), signature (REQUIRED, Base64url EdDSA by actor over the Section 2.1 signing input with the entry signature excluded).

Engagement signature verification profile: Each top-level signature and each change-log entry signature MUST identify its verification key with the corresponding *_kid field. The kid value MUST be a DID URL whose DID portion is byte-for-byte equal to the signer DID: hiring_operator for hiring_operator_signature, deploying_principal for deploying_principal_signature, and the change-log entry actor for

an entry signature. For did:aip signer DIDs, the Registry MUST resolve the verification method through its registered public-key endpoint for that AID. For other DID methods, the Registry MUST use the signer's DID method resolution rules. The resolved verification method MUST be Ed25519 verification material controlled by the signer DID. Missing kid fields, malformed DID URLs, signer/key DID mismatches, unsupported key types, failed DID resolution, or failed EdDSA signature verification MUST cause rejection with engagement_signature_invalid.

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 version semantics: The version field is the current change-log sequence number, not an independent counter. On creation, the Registry MUST create exactly one engagement_created entry with seq: 1 and set version: 1. Each accepted mutation after creation MUST append exactly one new change-log entry whose seq is previous version + 1, and the Registry MUST set version to that new seq in the same atomic update. Therefore, for every stored Engagement Object, version MUST equal max(change_log[*].seq). A mutation that changes status, participants, approval gates, countersignature state, or termination state without appending a matching change-log entry MUST be rejected.

Engagement Lifecycle:

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

Engagement Lifecycle Transitions:

* proposed to active: Requires both hiring_operator_signature and deploying_principal_signature. Missing deploying_principal_signature during activation MUST be rejected with engagement_countersign_required.

* 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 using the Approval Envelope cancellation rules in Section 13.6.

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

This termination cascade applies only when an Engagement transitions to terminated. Transition to completed does not trigger any Capability Overlay invalidation or Approval Envelope cancellation cascade.

6. Registration Protocol

6.1. Envelope Submission

The Registration Envelope request body is defined only in Section 5.6. An agent is registered by submitting that object to POST /v1/agents. This section defines Registry processing and validation; it does not define a second Registration Envelope schema.

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.

If any required condition in Checks 1 through 15, including their lettered subchecks, fails and that check does not explicitly name a more specific error code, the Registry MUST reject the Registration Envelope with registration_invalid. Because the checks are ordered, the Registry SHOULD report the error code associated with the first failed check.

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. The namespace MUST have an active, non-reserved entry in the Registry's synced AIP Namespace Catalog, unless the Registry explicitly documents a private local namespace policy. A mismatch, unknown namespace, removed namespace, or reserved namespace MUST be rejected with registration_invalid.

4. Check 4. identity.aid MUST NOT already exist in the Registry, and identity.public_key MUST NOT already be associated with a registered, non-revoked AID in the Registry. If either condition fails, the Registry MUST reject with aid_already_registered.

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 MUST be present and MUST be valid JSON conforming to the Capability Manifest schema. It MUST include manifest_id, aid, granted_by, version, issued_at, expires_at, capabilities, and signature. The version field MUST equal 1 for initial registration; manifest_id MUST match the cm: UUID pattern; issued_at and expires_at MUST be valid date-time values; capabilities MUST be an object; and signature MUST be a non-empty base64url value. The capability_manifest.expires_at value 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, MUST conform to the Principal Token schema, and MUST carry a JOSE header with typ equal to "JWT", alg equal to "EdDSA", and kid identifying an Ed25519 verification method controlled by the decoded iss DID or AID. For a did:aip issuer, the Registry MUST resolve the key through its public-key endpoint for that registered AID. The Registry MUST verify the Principal Token signature using the resolved verification method. If any Check 8 condition fails, the Registry MUST reject with registration_invalid.

9. Check 9. The decoded principal_token payload sub MUST equal identity.aid. If delegation_depth is 0, delegated_by MUST be null and iss MUST equal principal.id. If delegation_depth is greater than 0, delegated_by MUST be a registered, non-revoked parent AID, iss MUST equal delegated_by, and the Registry MUST reconstruct the complete candidate chain by appending the

submitted Principal Token to the parent's stored registration chain. The reconstructed chain MUST satisfy the delegation-chain rules in Section 9.11, including linkage, consistent root principal, scope inheritance, task binding, and maximum delegation depth. The submitted capability_manifest scopes MUST be a subset of both the submitted Principal Token scope array and the parent's effective Capability Manifest. If any Check 9 condition fails, the Registry MUST reject with registration_invalid or invalid_delegation_depth when the depth limit is exceeded.

10. Check 9a. max_delegation_depth Bounds: If the root Principal Token for the reconstructed registration chain contains a max_delegation_depth claim, the Registry MUST verify that its value is an integer in the range 0 through 10 inclusive. If max_delegation_depth is absent from the root Principal Token, the Registry MUST treat it as 3 for all subsequent registration processing. If the value exceeds 10 or is a negative integer, the Registry MUST reject the registration with registration_invalid and MUST include the detail string max_delegation_depth exceeds the hard cap of 10 in the error response.

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

12. Check 11. If the synced AIP Namespace Catalog entry for identity.type has requires_task_id: true, the decoded principal_token payload task_id MUST be non-null and non-empty.

13. Check 12. capability_manifest.signature MUST be verifiable against the granted_by DID's public key.

14. Check 13. identity.version MUST equal 1 and identity.previous_key_signature MUST be absent. Registration Envelopes are initial-registration objects only; key rotation is processed through PUT /v1/agents/{aid}.

15. Check 14a. grant_tier MUST be present and MUST be one of "G1", "G2", or "G3". If grant_tier is absent or has any other value, the Registry MUST reject with registration_invalid.

16. Check 14b. The Registry MUST determine the requested security Tier as the highest AIP Scope Catalog tier among all active scopes in capability_manifest.capabilities.

17. Check 14c. The submitted grant_tier MUST satisfy the requested security Tier determined in Check 14b. Tier 1 permits "G1", "G2", or "G3". Tier 2 permits "G2" or "G3" unless the Registry's identity_proofing_required_for_tier2 metadata field is true, in which case Tier 2 permits only "G3". Tier 3 permits only "G3". If grant_tier is inconsistent with the requested security Tier or the Registry's Tier 2 identity-proofing policy, the Registry MUST reject with registration_invalid.

18. Check 14d. If the requested security Tier determined in Check 14b is 2 or 3, the decoded principal_token payload principal.id MUST use the did:web method. Otherwise, the Registry MUST reject with principal_did_method_forbidden.

19. Check 14e. If grant_tier is "G3", the decoded principal_token payload MUST include a non-empty acr string and a non-empty amr array. If either claim is absent or empty, the Registry MUST reject with identity_proofing_insufficient.

20. Check 15. If the requested security Tier determined in Check 14b is 2 and identity.model.attestation_hash is absent, the Registry SHOULD accept the Registration Envelope, but only if it creates a structured registration warning with code equal to "model_attestation_missing_tier2", severity equal to "warning", source_check equal to "registration_check_15", and issued_at set to the Registry's current time. The warning message MUST state that the Tier 2 agent is not cryptographically pinned to a model artifact. The Registry MUST persist this warning in Agent Registration Metadata, MUST include it in the successful POST /v1/agents response, and MUST return it in subsequent GET /v1/agents/{aid} metadata responses while the registered Agent Identity Object lacks model.attestation_hash. The persisted warning is the required audit artifact for this condition. If the requested security Tier is 3, identity.model.attestation_hash MUST be present and MUST match the sha256:<64 lowercase hex characters> pattern. If the requested security Tier is 3 and the field is absent or malformed, the Registry MUST reject with registration_invalid.

On acceptance, the Registry MUST store the reconstructed registration chain for the registered AID. For a direct registration, that chain contains the submitted root Principal Token. For a sub-agent registration, it contains the parent's stored registration chain followed by the submitted delegated Principal Token. 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 11.4).

If the synced AIP Namespace Catalog entry for identity.type defines an automatic lifecycle expiry rule, the Registry MUST persist the resulting lifecycle_expires_at value in its stored registration record before accepting the Registration Envelope. For the Draft-02 ephemeral namespace, lifecycle_expires_at MUST be no later than the earlier of the decoded root Principal Token expires_at value and capability_manifest.expires_at.

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 registered AID MUST have a corresponding W3C DID Document resolvable by a DID resolver that implements the did:aip method defined in this section. The DID Document is derived deterministically from the Agent Identity, active Capability Manifest, and lifecycle state stored in the authoritative 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. Without one of these DID-specific Accept values, the endpoint returns the default Agent Registration Metadata response defined in Section 17.4.

AIP implementations MUST support DID resolution for at minimum did:aip, did:key, and did:web DID methods. Resolved DID Documents MAY be cached for a maximum of 300 seconds.

DID resolution performed during token validation MUST use an explicit timeout. For any validation step that is REQUIRED because the token has security Tier 2 or Tier 3 or because the token contains aip_registry, each remote DID resolution attempt MUST time out no later than 2 seconds after it is started, and the aggregate wall-clock budget for DID resolution across a single token validation attempt MUST NOT exceed 5 seconds. Local non-network resolution, such as did:key, SHOULD complete without consuming this remote-resolution budget. Implementations MAY use shorter timeouts, but MUST NOT use a zero-duration timeout for a required remote DID resolution attempt.

If DID resolution fails or times out during a REQUIRED validation step, implementations MUST treat the result as registry_unavailable and MUST reject the operation. If a Tier 1-only operation performs

Registry Trust Anchoring as a RECOMMENDED check and DID resolution times out, the Relying Party MAY either reject with registry_unavailable or skip that RECOMMENDED check according to local policy.

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

7.2. did:aip Method Resolution

A did:aip resolver MUST NOT infer the authoritative Registry from the namespace or agent-id components of the DID. The resolver MUST be invoked with an authoritative Registry base URI, or MUST obtain one from protocol context: the aip_registry claim in a Credential Token, the iss or aip_registry value in a Step Execution Token, the root principal DID Document's AIPRegistry service entry, or a local trust policy that binds the AID namespace to a Registry. If no authoritative Registry can be determined, resolution fails and protocol validation that requires the result MUST reject with registry_unavailable.

The resolver's input DID MUST match the did:aip syntax in Section 4.1. A resolver receiving a malformed AID MUST return a DID Core resolution result with didResolutionMetadata.error set to invalidDid and MUST NOT contact a Registry.

To resolve a valid AID, the resolver MUST perform the following steps:

1. Resolve and, when required by the validation context, pin the authoritative Registry trust state using the Registry Genesis procedure in Section 7.4.

2. Percent-encode the complete AID for use as the {aid} path parameter according to Section 17.2.

3. Send GET /v1/agents/{aid} to the authoritative Registry with Accept: application/did+json or Accept: application/did+ld+json.

4. If the Registry returns HTTP 404, return a DID resolution result with didResolutionMetadata.error set to notFound.

5. If the Registry cannot be reached or times out under the limits in Section 7.1, return a DID resolution result with didResolutionMetadata.error set to registry_unavailable.

6. If the requested representation is unsupported, return a DID resolution result with didResolutionMetadata.error set to representationNotSupported.

7. On success, return a DID Core resolution result whose didDocument.id equals the input AID, whose didResolutionMetadata.contentType matches the returned representation, and whose didDocumentMetadata contains versionId, updated, deactivated, and aip_registry.

A Registry constructs the did:aip DID Document from its stored registration record as follows. The DID Document id is the AID. The verificationMethod array contains the current active Ed25519 public key from the current Agent Identity Object. The verification method id MUST equal <aid>#key-<identity.version>, its controller MUST equal the AID, and its publicKeyJwk MUST contain only public JWK members. The authentication array MUST contain that verification method identifier. The DID Document controller value MUST equal the active Capability Manifest granted_by value. The didDocumentMetadata.versionId value MUST equal the decimal string form of identity.version.

DID URL dereferencing for did:aip supports only the empty fragment and key fragments of the form #key-<positive-integer>. The empty fragment dereferences to the DID Document. A key fragment dereferences to the corresponding verification method if the Registry retains the requested historical key version at GET /v1/agents/{aid}/public-key/{key-id}; otherwise dereferencing returns didResolutionMetadata.error set to notFound. Other fragments MUST return notFound.

The CRUD operations for the did:aip method are:

Operation	Mechanism
Create	POST /v1/agents with a Registration Envelope that passes Section 6 validation
Read	GET /v1/agents/{aid}; default metadata response, or DID Document with DID Accept header
Update	PUT /v1/agents/{aid} for key rotation only; arbitrary DID Document mutation is not supported
Deactivate	POST /v1/revocations with a valid full_revoke Revocation Object

Table 11

Deactivation is permanent for the affected AID. After full revocation, the Registry MUST return didDocumentMetadata.deactivated as true. The Registry MAY still return the last DID Document and MUST continue to serve retained historical public keys for audit and signature verification, but Relying Parties MUST reject new protocol operations for a deactivated AID during revocation and lifecycle validation.

7.3. 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/did-document.json_. Method operations and DID URL dereferencing rules are defined in Section 7.2.

A principal that authorises agents for Tier 2 or Tier 3 operations MUST use the did:web DID method, and that principal’s DID Document MUST include an AIPRegistry service entry in its service array. 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). did:key principals MUST NOT authorise Tier 2 or Tier 3 operations.

For principals using any W3C DID method other than did:web or did:aip, authorisation is limited to Tier 1. Even if such a DID method supports service entries, this specification does not permit it for Tier 2 or Tier 3 principal authorisation. For did:web principals, the AIPRegistry service entry is REQUIRED when the principal authorises any agent with Tier 2 or Tier 3 scopes. It is OPTIONAL for principals authorising only Tier 1 agents.

7.4. 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.

7.4.1. 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 an equivalent at-rest key-protection control satisfying Section 21.4). The corresponding public key is published in the initial Registry Trust Record. Separate active verification keys for CRL documents (Section 11.3) and Step Execution Tokens are RECOMMENDED so that routine operational-key rotation does not require trust-root rotation.

7.4.2. 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 /v1/registry-metadata.

```
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.

7.4.3. 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 8* (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.

7.4.4. Registry Metadata Publication

The Registry MUST publish Registry Metadata at the following endpoint relative to its registry_id origin:

GET /v1/registry-metadata

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 protocol compatibility version this Registry conforms to; not the Internet-Draft

			revision
registry_trust_uri	string	REQUIRED	URI of the current Registry Trust Record
endpoints	object	REQUIRED	Map of service names to relative paths or absolute HTTPS URIs
enterprise_idp_federation_supported	boolean	OPTIONAL	True only when the Registry supports the Enterprise IdP Federation Profile in Section 8.4.5. SHOULD be present in all Registry Metadata documents. Absence is treated as equivalent to false. Gateways that surface this capability through /.well-known/ oauth-protected-resource SHOULD ensure the value is consistent with the Registry's declared support in

			this document. A mismatch between the two documents MUST be treated as a Registry configuration error.

Table 12

The _endpoints_ object MUST include at minimum:

Key	Description
agents	Base path for agent endpoints (e.g., /v1/agents)
crl	Preferred CRL retrieval URI. This value MAY be the origin path /v1/crl or an absolute HTTPS CDN/distribution URI.
revocations	Revocation submission path (e.g., /v1/revocations)

Table 13

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

* Fetch /v1/registry-metadata 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.

* Use the accepted Registry Trust Record's signed endpoints object as the authoritative endpoint map for Registry interactions. The Registry Metadata endpoints object is a bootstrap hint and MUST NOT override a valid signed Registry Trust Record.

* On subsequent contacts, use the versioned trust-update procedure in Section 7.4.7.1 before updating the local trust state.

Endpoint values that begin with / are resolved relative to registry_id. Absolute endpoint values MUST use HTTPS. The endpoints.crl value MAY use a different HTTPS origin from registry_id so that CRL documents can be served from a CDN or distributed object store. A different CRL origin does not identify a different Registry; verifiers bind CRL documents to the Registry by verifying the CRL signature against the accepted Registry Trust Record version current at CRL issuance time.

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 7.4.7.2.

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

7.4.5. 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 14

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.

Each entry in active_verification_keys.step_execution, active_verification_keys.crl, and active_verification_keys.notifications MUST be an Ed25519 public JWK containing kty, crv, x, and keyid. A verifier MUST NOT treat a bare key identifier as sufficient for runtime artifact verification; the accepted Registry Trust Record version must provide the public key material used to verify the artifact signature.

For CRL retrieval, verifiers MUST use signed.endpoints.crl from the accepted Registry Trust Record, resolving it according to the endpoint rules above. The unsigned Registry Metadata endpoints.crl value MAY be used only to locate bootstrap metadata before the Registry Trust Record is accepted.

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.

7.4.6. 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.

7.4.7. 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.

A Step Execution Token is not a trust-bootstrap artifact. Relying Parties MUST NOT establish first-contact Registry trust from a SET received in an application request. A SET whose iss Registry ID is unknown, unpinned, expired, or missing the locally retained Registry Trust Record version named by the SET protected header aip_trv MUST be rejected with registry_untrusted. An implementation MAY perform first-contact bootstrap or sequential trust update as a separate pre-validation operation, but it MUST complete and pin the accepted trust state before reprocessing the SET.

7.4.7.1. 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 Registry Metadata 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.

7.4.7.2. 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.

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.

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.

7.5. AIP Gateway Protected Resource Metadata

Any MCP gateway that enforces AIP authentication for tool listing or tool execution MUST publish OAuth Protected Resource Metadata [RFC9728] using the registered oauth-protected-resource well-known URI suffix:

https://{gateway-host}/.well-known/oauth-protected-resource

AIP does not define or require an AIP-specific well-known URI suffix for gateway discovery. The response MUST be a JSON object conforming to RFC 9728 and containing the following fields:

```
{
  "resource": "https://gateway.example.com/",
  "authorization_servers": ["https://registry.example.com"],
  "scopes_supported": ["email.read", "calendar.read"],
  "bearer_methods_supported": ["header"],
  "dpop_signing_alg_values_supported": ["EdDSA"],
  "dpop_bound_access_tokens_required": true,
  "resource_name": "Example AIP MCP Gateway",
  "resource_documentation": "https://gateway.example.com/docs"
}
```

resource REQUIRED. The gateway protected resource identifier. It MUST be an HTTPS URL with no fragment, as defined by [RFC9728].

authorization_servers REQUIRED for AIP gateways. The array MUST contain at least one AIP Registry OAuth authorization-server issuer identifier acceptable for this protected resource.

scopes_supported RECOMMENDED. When present, values MUST be AIP scope strings from the synced AIP Scope Catalog or accepted local extension policy.

bearer_methods_supported REQUIRED for AIP gateways that accept non-DPoP Bearer tokens and MUST include "header" when present. AIP uses the registered Bearer authentication scheme for requests that do not require proof-of-possession and the registered DPoP authentication scheme for DPoP-bound tokens.

dpop_signing_alg_values_supported REQUIRED when any advertised scope or endpoint requires DPoP, and MUST include "EdDSA".

dpop_bound_access_tokens_required REQUIRED when the gateway requires DPoP-bound access tokens for all requests to the protected resource. If omitted or false, clients MUST still apply AIP DPoP requirements derived from the requested scope Tier, Scope Catalog metadata, and endpoint policy.

tls_client_certificate_bound_access_tokens REQUIRED and set to true when the gateway enforces AIP Tier 3 mTLS-bound access tokens for the protected resource.

Endpoint Requirements:

1 The /.well-known/oauth-protected-resource endpoint MUST respond to unauthenticated HTTP GET requests. Authentication MUST NOT be required to retrieve the Protected Resource Metadata document.

2 The response MUST use HTTP status 200 and Content-Type: application/json.

3 The response SHOULD include a Cache-Control header with a max-age value between 60 and 300 seconds. Clients MUST NOT cache the Protected Resource Metadata document for longer than 300 seconds.

4 *Consistency Constraint:* If a gateway advertises Tier 2 or Tier 3 scopes in scopes_supported, or if local endpoint policy marks the protected resource as Tier 2 or Tier 3, the gateway MUST either set dpop_bound_access_tokens_required to true or document per-scope DPoP requirements through Registry policy. A client MUST treat contradictory metadata as gateway_config_invalid.

5 For stateless gateway deployments (Section 8.2.1), the gateway MUST NOT serve stale Protected Resource Metadata; the document MUST reflect the currently active resource configuration from the shared configuration store.

MCP clients that discover a gateway through OAuth Protected Resource Metadata SHOULD fetch this document before tool listing or tool execution. If the document is absent, malformed, or does not identify an acceptable AIP Registry authorization server, an AIP-aware MCP client MUST NOT assume that unauthenticated tool access is permitted.

8. Credential Tokens

8.1. Credential Token Transport and Version Header

The Credential Token JWT header and payload fields are defined in Section 5.4. This section defines only the HTTP transport profile, version-header requirements, issuance lifetime rules, and refresh behavior for those tokens.

An AIP Credential Token for a request that does not require proof-of-possession is transmitted as an HTTP Authorization header using the registered Bearer authentication scheme [RFC6750]:

EXAMPLE (informative):

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

For interactions requiring Proof-of-Possession, the Credential Token is transmitted using the registered DPoP authentication scheme [RFC9449] and the request also carries a DPoP proof JWT:

EXAMPLE (informative):

```
Authorization: DPoP <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". This value is the AIP protocol compatibility version defined in Section 20, not the Internet-Draft revision suffix. Implementations MUST reject requests with unsupported_version when the X-AIP-Version header is absent, carries an unsupported value, or does not match the token aip_version claim.

Every AIP-aware HTTP response to an AIP protocol request MUST include an X-AIP-Version response header containing the AIP protocol compatibility version used to interpret the request and generate the response. For responses conforming to this specification, the value MUST be "0.3". This requirement applies to successful responses and error responses, including responses that reject a request with unsupported_version. When a request is rejected because the requested version is unsupported, the responder MUST set X-AIP-Version to a protocol version it supports for that endpoint and SHOULD include X-AIP-Supported-Versions with a comma-separated list of supported AIP protocol versions.

8.2. Token Issuance

Implementations MUST enforce the following maximum Credential Token lifetimes. The effective TTL limit for a token is the lowest applicable limit from this table across all requested scopes.

Scope Category	Maximum TTL
Tier 1 scope	3600 seconds, unless the synced AIP Scope Catalog entry sets a lower ttl_max_seconds
Tier 2 scope	300 seconds, unless the synced AIP Scope Catalog entry sets a lower ttl_max_seconds
Tier 3 scope	300 seconds, unless the synced AIP Scope Catalog entry sets a lower ttl_max_seconds
Any requested scope	The ttl_max_seconds value in the synced AIP Scope Catalog entry for that scope, capped by the Tier ceiling above
Multiple scopes in one token	The lowest ttl_max_seconds value across all requested scopes

Table 15

When a token contains multiple scopes, the most restrictive TTL applies. Zero-duration and negative-duration tokens (where exp <= iat) MUST be rejected. A synced AIP Scope Catalog entry for a standard or extension scope MUST NOT advertise a ttl_max_seconds value greater than the ceiling for that scope's Tier. A token containing a scope whose active catalog entry is missing, inactive, or exceeds its Tier ceiling MUST be rejected with invalid_scope or invalid_token as applicable to the validation step.

A token's security Tier is determined by its highest-risk scope (Section 3.2). 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.

8.2.1. Token Cache Requirements for Stateless Deployments

AIP Registries and Relying Parties operating in stateless transport environments, including MCP stateless transport [MCP-STATELESS], or horizontally-scaled deployments MUST store validated Credential Token cache entries in a shared, externally-accessible store, such as Redis or a distributed key-value store, that is accessible to all instances that may validate tokens for the same AID. Local in-process or filesystem-backed caches are PERMITTED only in single-instance

development deployments and MUST NOT be used in a Production Deployment as defined in Section 3.

A validation cache entry MUST be keyed by SHA-256(exact-compact-token), not by jti alone. The cache entry MUST include:

* the token iss and jti for replay-cache correlation;

* the token exp as the maximum cache TTL;

* the cached Registry-dependent validation subresults and their freshness sources;

* the set of validation step identifiers whose results were cached; and

* the validated AIP Tier.

A cache entry is an optimization for previously completed validation, not a replacement for the validation algorithm. Implementations MAY reuse cached Registry-dependent validation results only when the token's Tier and the relevant validation step permit bounded staleness. For Tier 2 and Tier 3 tokens, a cached accepted result MUST NOT bypass the live revocation lookup required by Section 11.4 and Section 9.10. Per-request checks that are not cacheable, including proof-of-possession, mTLS, audience matching, expiration, and replay policy, remain in force for every interaction. A cached accepted result MUST NOT cause a token whose (iss, jti) pair has already been consumed to be accepted again. This cache prevents redundant Registry round-trips only for validation results whose freshness rules allow reuse.

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 across all tokens issued by that agent AID. This uniqueness requirement is independent of aip_chain, principal DID, registration grant, scope set, or audience. An agent delegated by multiple principals MUST still maintain a single jti uniqueness domain for its issuer AID.

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. The refresh threshold is derived from the token's effective TTL, not from a fixed Tier label:

```
effective_ttl_seconds = exp - iat

refresh_lead_seconds =
  max(1, min(
    300,
    max(30, ceiling(effective_ttl_seconds * 0.10)),
    floor(effective_ttl_seconds / 2)
  ))

begin refresh when (exp - now) <= refresh_lead_seconds
```

This formula preserves the former 300-second refresh lead for a 3600-second catalog TTL and the former 30-second refresh lead for a 300-second catalog TTL, while remaining valid for future catalog TTL values.

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 re-establish their delegation rather than merely refreshing their Credential Token. AIP does not define a separate on-wire error code for "delegation refresh required"; chain renewal is the required agent-side handling of chain_token_expired.

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 catalog-derived Credential Token TTLs 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 Section 8.2 catalog-derived rules as for any other interaction involving those scopes.

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=email.read 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. For OAuth-protected resources, the value MUST be an RFC 9728 [RFC9728] resource identifier.

5 scope MUST use AIP scope strings from the synced AIP Scope Catalog or accepted local extension policy. 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 identifies a registered MCP server or OAuth-protected resource. The Registry MUST perform this check using at least one of the following mechanisms:

* match the resource URI against a Registry-local registration record for an MCP server or protected resource; or

* obtain the resource's Protected Resource Metadata using RFC 9728 [RFC9728] and verify that the returned resource metadata value is the same resource identifier as the requested resource.

When RFC 9728 metadata is used, the metadata MUST identify an authorization server acceptable to the Registry, either by listing the Registry's OAuth authorization-server issuer identifier in authorization_servers or by satisfying an equivalent local trust policy. The Registry MUST reject with invalid_target if metadata retrieval fails, the returned resource value does not match the requested resource, the resource is not locally registered, or no acceptable authorization server relationship is established.

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": "email.read calendar.read",
"issued_token_type": "urn:ietf:params:oauth:token-type:access_token"
 }
```

8.4.5. Enterprise IdP Federation Profile

When the target resource in a token exchange request is registered in the Registry with enterprise_idp_required: true (see Section 17.13), the AIP Registry acting as intermediary MUST perform a secondary token exchange with the configured Enterprise IdP. The Registry MUST NOT perform the enterprise IdP leg for resources registered with enterprise_idp_required: false or with enterprise_idp_required absent. Enterprise integrations such as Microsoft Entra Agent ID [MS-ENTRAAGENTID] can use this profile when the resource registration record requires it.

1 The AIP Registry validates the inbound AIP Credential Token per Section 9.

2 The Registry constructs a signed JWT assertion using its registered client identity with the enterprise IdP. The assertion MUST include:

- iss: the AIP Registry’s client_id registered with the enterprise IdP;
- sub: the agent’s AID;
- aud: the enterprise IdP token endpoint URI;

* aip_sub_principal: the root Principal Token principal.id value from the validated aip_chain;

* aip_scopes: the validated aip_scope values from the Credential Token.

3 The Registry submits this assertion to the enterprise IdP token endpoint using grant_type=urn:ietf:params:oauth:grant-type:jwt-bearer per RFC 7523 [RFC7523] or grant_type=urn:ietf:params:oauth:grant-type:token-exchange per RFC 8693 [RFC8693].

4 The enterprise IdP evaluates the request against its Conditional Access, ABAC, and equivalent enterprise access policies using aip_sub_principal as the human or organizational principal context and the agent AID as the client identity.

5 On success, the enterprise IdP issues an access token scoped to the target resource.

6 The AIP Registry returns the enterprise-issued token in the token exchange response.

8.4.5.1. Enterprise Assertion JWT Schema

The JWT assertion constructed by the Registry for the enterprise IdP token endpoint MUST contain the following payload fields:

Field	Type	Required	Constraints
iss	string	REQUIRED	MUST be the AIP Registry's client_id as registered with the enterprise IdP. MUST NOT be the Registry's registry_id.
sub	string	REQUIRED	MUST be the acting agent's AID.
aud	string or array	REQUIRED	MUST be the enterprise IdP's token endpoint URI.
aip_sub_principal	string	REQUIRED	MUST be the root Principal Token principal.id value from

			aip_chain[0] of the validated Credential Token. MUST be a valid W3C DID.
aip_scopes	array of strings	REQUIRED	MUST be the validated aip_scope values from the Credential Token. MUST NOT include scopes not present in the validated Credential Token.
iat	integer	REQUIRED	Unix timestamp representing the current time at assertion construction.
exp	integer	REQUIRED	Unix timestamp. MUST be iat + 60 or earlier. Registries MUST NOT issue assertions with exp - iat > 60.
jti	string	REQUIRED	UUID v4. MUST be unique per assertion to prevent replay.

Table 16

8.4.5.2. Enterprise IdP Error Handling

When the enterprise IdP token endpoint returns an error or cannot be reached, the AIP Registry MUST map the result to the agent-facing AIP response as follows:

Enterprise IdP Response	AIP Registry Response to Agent
HTTP 400 (invalid_request)	HTTP 400, error: invalid_request
HTTP 400 (invalid_client)	HTTP 500, error: idp_client_misconfigured
HTTP 400 (invalid_grant)	HTTP 400, error: invalid_grant
HTTP 400 (unauthorized_client)	HTTP 403, error: insufficient_scope
HTTP 403 (access_denied or Conditional Access denial)	HTTP 403, error: enterprise_policy_denied
HTTP 4xx (other)	HTTP 400, error: invalid_request
HTTP 5xx	HTTP 503, error: registry_unavailable
Timeout greater than 5 seconds	HTTP 503, error: registry_unavailable
TLS or network error	HTTP 503, error: registry_unavailable

Table 17

The Registry MUST NOT expose raw enterprise IdP error messages to the calling agent. When enterprise_policy_denied is returned, the error_description SHOULD indicate that the request was denied by enterprise policy without revealing specific policy rules. The Registry MUST log the full enterprise IdP response for audit purposes per Section 21.4.

8.4.5.3. Federation Metadata and Grant Tier Guidance

AIP Registries that support this profile MUST advertise enterprise_idp_federation_supported: true in their /v1/registry-metadata metadata as described by Section 7.4.4 and Section 17.14.1. Relying Parties and agents MUST NOT infer federation support from absence of this field.

Enterprise deployments using this profile SHOULD require G3 grants from Section 12 for the root Principal Token so that human authentication context and IdP policy context are available during token exchange and Tier 3 validation.

9. Credential and Step Execution Token Validation

The Credential Token Validation Algorithm is the normative heart of AIP for agent-issued Credential Tokens. A Relying Party MUST execute the following steps in the order presented unless it is validating a Registry-issued Step Execution Token under the SET validation profile below. 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. Additional lettered steps marked 2a, 6a, 6b, 9a, 9b, 9c, 9d, and 10a are part of the numbered step at which they appear and do not change the base step numbering.

The labels in the following table are normative validation step identifiers for implementation traceability and conformance tests. Lettered labels, including Steps 5a through 5g, 8a through 8l, and 11a through 11c, are discrete sub-steps whether they appear as separate XML section headings or as labelled items within a validation step.

Label	Location	Validation subject
1	Section 9.1	JWT parse
2	Section 9.2	JWT header validation
2a	Section 9.3	Expiration preflight before Registry lookup
3	Section 9.4	Agent identification and public-key retrieval
4	Section 9.5	Credential Token signature verification
5a	Section 9.6.1	iat issued-at validation
5b	Section 9.6.2	exp ordering validation

5c	Section 9.6.3	Credential Token expiration validation
5d	Section 9.6.4	aud validation
5e	Section 9.6.5	jti replay validation
5f	Section 9.6.6	aip_version validation
5g	Section 9.6.7	Issuer and subject binding
6	Section 9.7	Credential Token TTL validation
6a	Section 9.8	Registry trust anchoring
6b	Section 9.9	Engagement validation
7	Section 9.10	Revocation validation
8a-8l	Section 9.11	Delegation chain sub-step validation
8 Post-Check A	Section 9.11	Credential Token issuer equals chain leaf
8 Post-Check B	Section 9.11	Credential Token subject equals issuer
8 Post-Check C	Section 9.11	Identity proofing claims
9	Section 9.12	Capability Manifest validation
9a	Section 9.13	Scope-to-manifest mapping
9b	Section 9.14	Capability Overlay application
9c	Section	Delegation inheritance

	9.15	
9d	Section 9.16	Grant Tier conformance
10	Section 9.17	DPoP validation
10a	Section 9.18	Approval step verification
11a	Section 9.19	aip_principal_assertion presence
11b	Section 9.19	aip_principal_assertion signature verification
11c	Section 9.19	aip_principal_assertion principal binding
12	Section 9.20	Accept

Table 18: Credential Token Validation Step Index

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 "EdDSA"

* 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 2a: Expiration Preflight

Before performing any Registry key lookup, the Relying Party MUST decode the JWT payload without using it for any acceptance decision. This preflight is only an error-ordering and lookup-amplification control. Signature verification in Step 4 and full claim validation in Step 5 remain mandatory before a token can be accepted.

The Relying Party MUST inspect only iat, exp, and aip_scope during this preflight. If iat or exp is absent, not an integer, or exp <= iat, reject with invalid_token. If exp is in the past, reject with token_expired. This rejection applies even if the token's kid references a historical key version that is no longer retained by the Registry.

If the unsigned payload's exp - iat exceeds the largest ttl_max_seconds value among active entries in the synced AIP Scope Catalog, the Relying Party MAY reject with invalid_token before key lookup. This check is conservative: it can reject tokens that no valid catalog entry could authorize, but it MUST NOT be used to accept a token.

9.4. 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 using GET /v1/ agents/{aid}/public-key/{key-id}, where {aid} is the DID URL without the fragment and {key-id} is the fragment without the leading #. The Registry response MUST conform to public-key-response.schema.json and return the public key material along with its validity period (valid_from timestamp and valid_until timestamp or null).

If the Registry cannot be reached or the public key lookup cannot be completed because the Registry is unavailable, reject with registry_unavailable. If the Registry lookup succeeds but the kid is not found or the key is not valid at the time of the JWT's iat claim, reject with unknown_aid. Because Step 2a runs before this lookup, an already-expired token MUST surface as token_expired, not unknown_aid, even when its historical signing key has aged out of Registry retention.

9.5. 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.

9.6. Step 5: Validate Claims

Validate the following claims from the decoded JWT payload:

9.6.1. Step 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.

9.6.2. Step 5b: exp (Expiration Time)

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

9.6.3. Step 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.

9.6.4. Step 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.

9.6.5. Step 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. This replay rule applies to authorisation decisions. Registry state-transition endpoints that define explicit idempotency behavior MAY return a previously stored response for an exact retry, but MUST NOT re-execute side effects and MUST NOT treat replay-cache acceptance as proof of fresh authority. The replay cache key MUST NOT include aip_chain, principal DID, registration grant, scope set, or audience. Credential Token issuers therefore MUST treat jti values as globally unique within the issuer AID across all delegation chains. After the token expires, the replay cache entry MAY be discarded.

9.6.6. Step 5f: aip_version

aip_version MUST be present and MUST be "0.3" for tokens conforming to this specification. This value is the AIP protocol compatibility version defined in Section 20, not the Internet-Draft revision suffix. If aip_version is absent, reject with invalid_token. If aip_version is present but unsupported, or if it does not match the X-AIP-Version request header, reject with unsupported_version and include the received aip_version and X-AIP-Version values in the error description to aid debugging.

9.6.7. Step 5g: Issuer and Subject Binding

The iss and sub claims MUST each match the did:aip ABNF. Let kid_aid be the DID URL in the JWT header kid with the fragment removed. The payload iss MUST equal kid_aid. The payload sub MUST equal iss. This specification does not define agent-issued Credential Tokens where a delegated token subject differs from the signing leaf agent. If any comparison fails, reject with invalid_token.

9.7. 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_max_seconds values in the synced AIP Scope Catalog and the Tier ceilings in Section 8.2:

```
effective_ttl_limit =
  min(
    scope.ttl_max_seconds,
    tier_ttl_ceiling(scope.tier)
    for scope in aip_scope
  )
```

When a token contains multiple scopes, the most restrictive catalog TTL applies.

If lifetime exceeds the limit, reject with invalid_token.

For validation, a token’s security Tier is determined by its highest-risk scope (Section 3.2). A token containing one Tier 2 scope and nine Tier 1 scopes is a Tier 2 token in its entirety. A Relying Party MUST NOT derive Tier from the majority of scopes or from the first scope in the array.

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

This step is REQUIRED for Tier 2 and Tier 3 operations and for any Credential Token that contains aip_registry. It is RECOMMENDED for Tier 1 operations when aip_registry is absent. 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. If the operation's security Tier is 2 or 3, principal_id MUST use the did:web method. If it uses did:key, did:aip, or any other DID method, reject with principal_did_method_forbidden.

3. 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.

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

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

6. 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).

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

8. If the DID Document does not contain an AIPRegistry service entry and the operation's security Tier is 2 or 3 or the token contains aip_registry, reject with registry_untrusted.

For Tier 1 operations where aip_registry is absent, 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 operation's security Tier is 2 or 3 or the token contains aip_registry, reject with registry_unavailable.

DID resolution for this step MUST use the timeout requirements in Section 7.1. For Tier 2 or Tier 3 operations and tokens containing aip_registry, a DID resolution timeout is a validation failure and MUST be reported as registry_unavailable; implementations MUST NOT wait indefinitely for the principal DID resolver.

A Relying Party MUST NOT use aip_registry as an authoritative Registry locator unless this step has verified it against the DID-Document-declared Registry endpoint.

9.9. 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, evaluate each gate trigger using the trigger grammar in Section 5.12. A scope:<scope-string> trigger guards any operation whose requested aip_scope set contains that scope. An action_type:<action-type> trigger guards Approval Envelope steps whose action_type exactly equals that value. If a matching required gate is still pending, reject with engagement_gate_pending. If a matching required gate has status "rejected", reject with engagement_terminated.

9.10. 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. Chain-wide and principal-wide revocation effects are completed in Step 8 after aip_chain has been validated.

* *Tier 1:* Retrieve a verifiable CRL from the Registry cache. The CRL's next_update MUST be later than the Relying Party's current UTC validation time. If no fresh verifiable CRL is available, reject with registry_unavailable. 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. The same CRL MUST also be used in Step 8 to evaluate principal_revoke entries whose target_id is the root Principal DID or any AID in the validated aip_chain.

* *Tier 2:* Perform a live Registry lookup. Query GET /v1/agents/{iss}/revocation. A successful response MUST be HTTP 200 with a body conforming to revocation-status.schema.json (Section 17.8). If revoked is true, reject with agent_revoked. If any currently requested scope appears in scopes_revoked, reject with agent_revoked. If the token relies on delegation authority rooted at an AID whose response has delegation_revoked equal to true, reject with agent_revoked. If the Registry returns unknown_aid, reject with unknown_aid. If the Registry is unreachable, reject with registry_unavailable. A previously cached accepted token validation result MUST NOT be used to skip this lookup.

The Relying Party MUST verify the Registry trust state used for this revocation check using the Registry trust bootstrapping and Registry Trust Record procedures in Section 7.4.4 and Section 7.4.5, and the sequential trust-update procedure in Section 7.4.7.1 when updating pinned trust state, before treating Registry responses as authoritative.

9.11. 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. Its JOSE header MUST contain typ equal to "JWT", alg equal to "EdDSA", and a non-empty kid DID URL. If parsing, schema validation, or header 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, and aip_chain[1], when present, has delegation_depth: 1. 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. Issuer and kid Binding: For token at index i, extract the iss claim. For i = 0, iss MUST equal principal.id. For i > 0, iss MUST equal delegated_by. The kid header MUST identify an Ed25519 verification method controlled by iss, and the DID portion of kid MUST equal iss. If any of these bindings fail, reject with delegation_chain_invalid.

8d-1. Root Principal Signature Verification: For i = 0, resolve kid using the DID method of the root principal DID in principal.id. The AIP Registry MUST NOT be used as the authority for root-principal keys. If DID resolution fails, the key is not Ed25519 verification material, the verification method is not controlled by principal.id, or signature verification fails, reject with delegation_chain_invalid.

8d-2. Delegated Agent Key Lookup: For i > 0, iss is the parent agent AID that signed this delegated Principal Token. The Relying Party MUST resolve the signing key through the authoritative AIP Registry using the percent-encoded issuer AID and the kid fragment: GET /v1/agents/{iss}/public-key/{key-id}. The key-id path component is the fragment portion of kid without the leading number-sign character. The Registry response MUST identify the same AID as iss and return Ed25519 public key material whose validity interval covers the Principal Token’s issued_at instant. If the Registry cannot be reached or the key lookup cannot be completed because the Registry is unavailable, reject with registry_unavailable. If the Registry lookup succeeds but iss is not a registered AID, kid is not found, or the key is not valid at issued_at, reject with unknown_aid.

8d-3. Delegated Principal Token Signature Verification: For i > 0, verify the Principal Token signature using the key resolved in Step 8d-2. If the resolved key is not Ed25519 verification material, identifies a key not controlled by iss, or signature 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. A delegated Principal Token MUST NOT be self-delegated: delegated_by[i] MUST NOT equal sub[i]. If linkage fails or the token is self-delegated, 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 the Registry returns unknown_aid for any chain sub AID, reject with unknown_aid. If any agent in the chain is revoked, reject with agent_revoked. A principal_revoke whose target_id equals any chain sub AID MUST be treated as revoked even when no descendant Revocation Object has been materialised for that AID.

8g. No Duplicate AIDs: No AID may appear more than once in the chain (no cycles or duplicates). This rule includes direct self-delegation, where a Principal Token attempts to delegate from an AID back to the same AID. 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, parse issued_at and expires_at as ISO 8601 UTC instants. Verify issued_at is not more than 30 seconds in the future relative to the Relying Party's current UTC clock. If it is, reject with delegation_chain_invalid. Verify expires_at > issued_at; if not, reject with delegation_chain_invalid. Verify expires_at is in the future; if expired, 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.

8k. Namespace Task Binding: For every Principal Token whose sub namespace has requires_task_id: true in the synced AIP Namespace Catalog, the token payload MUST include a non-null, non-empty task_id. If such a chain element omits task_id, sets it to null, or sets it to an empty string, reject with delegation_chain_invalid.

8l. Root Principal Revocation: Let root_principal_id be the principal.id value that Step 8i verified across the chain. Using the revocation source required for the token's Tier, verify that no active principal_revoke has target_id equal to root_principal_id. A match invalidates every Credential Token and Step Execution Token rooted at that Principal DID; reject with agent_revoked. This rejection is required regardless of the propagate_to_children value on the Revocation Object.

9.11.1. 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.

9.11.2. Step 8 Post-Check B

Verify that the Credential Token payload sub MUST equal iss. Together with Post-Check A, this confirms that the token subject, token issuer, signing key AID, and leaf Principal Token subject all identify the same acting agent. This check applies to both direct and delegated Credential Tokens. If mismatch, reject with delegation_chain_invalid.

9.11.3. Step 8 Post-Check C: Identity Proofing

This check applies when any scope in aip_scope has synced AIP Scope Catalog tier 3, when the Registry's identity_proofing_required_for_tier2 metadata field is true and the operation security Tier is 2, when Registry registration metadata for the agent identifies grant_tier: "G3", or when the Relying Party or operation policy declares one or more required acr_values.

When this check applies, the root Principal Token at aip_chain[0] MUST include a non-empty acr string and a non-empty amr array. If either claim is absent or empty, the Relying Party MUST reject the request with identity_proofing_insufficient.

The Relying Party MUST evaluate only the acr and amr values contained in the signed root Principal Token. It MUST NOT accept unsigned or out-of-band acr or amr values for this check.

If one or more acr_values are required, the acr value in aip_chain[0] MUST exactly match one of those required values unless the Registry's /v1/registry-metadata document contains an acr_equivalence_map object

(Section 17.14.1) that explicitly maps the received acr value to one of the required values. Equivalence MUST only map a received value to a declared required value where the received value represents equal or higher assurance than the required value under the Registry's published ACR equivalence policy. The acr_equivalence_map MUST NOT map a lower-assurance acr to a higher-assurance acr. The policy MUST identify the assurance framework or frameworks used for each mapping and MUST give a stable ordering or rationale sufficient for an auditor to reproduce the decision. Implementations MUST NOT accept equivalence mappings from any source other than the Registry's signed /v1/registry-metadata document and its referenced policy. This specification does not define a global ordering over acr values. If the acr claim does not satisfy the declared requirement, the Relying Party MUST reject the request with identity_proofing_insufficient.

9.12. 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 GET /v1/agents/{iss}/capabilities. The response MUST conform to capability-manifest.schema.json, and manifest.aid MUST equal iss. If unavailable, malformed, or not bound to iss, reject with manifest_invalid.

2. Verify the manifest signature using manifest.signature_kid. The DID portion of signature_kid MUST equal manifest.granted_by. If key resolution, key binding, or signature verification fails, reject with manifest_invalid.

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

4. For each requested scope whose synced AIP Scope Catalog entry has a non-null constraint_schema, validate the corresponding Capability Manifest value against that schema. If validation fails, reject with manifest_invalid.

9.13. Step 9a: Scope Verification

For all agents: verify each scope in aip_scope is present with status: "active" or, when local policy allows experimental behavior, status: "experimental" in the synced AIP Scope Catalog. If any requested scope is unknown, reserved, removed, or not permitted by local extension policy, reject with invalid_scope. Then verify each requested scope is present in the Capability Manifest. If any

requested scope is absent from the Capability Manifest, reject with insufficient_scope.

9.14. Step 9b: Capability Overlay (Conditional)

If a Capability Overlay exists, verify scope constraints permit the requested operation. For each overlay constraint whose synced AIP Scope Catalog entry has a non-null constraint_schema, validate the overlay constraint against that schema before applying attenuation. If schema validation fails, reject with overlay_exceeds_manifest.

9.15. Step 9c: Delegation Scope Inheritance

For delegated Credential Tokens, the Relying Party MUST verify that the requested scopes and the leaf agent's Capability Manifest are valid attenuations of every delegation in aip_chain.

Let chain[i] be the decoded Principal Token payload at aip_chain[i], and let manifest(chain[i].sub) be the current Capability Manifest for the agent identified by chain[i].sub. Let n be the length of aip_chain. The Relying Party MUST execute the inheritance validation in increasing index order, from the root delegated agent (i = 0) to the leaf acting agent (i = n-1).

1. For every i from 0 to n-1, obtain the current Capability Manifest M[i] for chain[i].sub. The leaf manifest M[n-1] MAY reuse the manifest already fetched in Step 9. All ancestor manifests MUST be fetched according to the caching rules in Section 10. If any required manifest is unavailable, reject with manifest_invalid.

2. For every obtained manifest M[i], verify M[i].aid == chain[i].sub, verify the manifest signature using M[i].signature_kid, verify the DID portion of M[i].signature_kid equals M[i].granted_by, and verify that M[i].expires_at is in the future. If the AID binding, key binding, or signature check fails, reject with manifest_invalid. If the manifest is expired, reject with manifest_expired.

3. For every requested scope in aip_scope, verify that the scope is present in the scope array of every Principal Token in aip_chain and is granted by the leaf manifest M[n-1]. If any requested scope is absent from any chain token or from the leaf manifest, reject with insufficient_scope.

4. For every adjacent pair (M[i-1], M[i]) where i ranges from 1 to n-1, verify that the child manifest M[i] is a valid attenuation of the parent manifest M[i-1]. Every scope granted by M[i] MUST be present in M[i-1], and every child constraint for that scope

MUST be equal to or more restrictive than the corresponding parent constraint. If a child scope is absent from the parent manifest, or if a child constraint is more permissive than the parent constraint, reject with insufficient_scope.

Constraint comparison MUST use the recursive attenuation semantics from Rule CO-1, treating the parent manifest constraint as the base value and the child manifest constraint as the overlay value. When the synced AIP Scope Catalog provides a non-null constraint_schema, both parent and child constraints MUST first validate against that schema. If schema validation fails, reject with manifest_invalid. Any violation at any adjacent pair is fatal to the chain. For example, a chain where M[0].transactions.max_single_transaction = 250, M[1].transactions.max_single_transaction = 300, and M[2].transactions.max_single_transaction = 200 is invalid at pair (M[0], M[1]), even though M[2] is more restrictive than M[1].

9.16. Step 9d: Grant Tier Conformance

Determine the operation's security Tier as the highest tier value among all requested aip_scope entries in the synced AIP Scope Catalog. For every Principal Token payload in aip_chain, retrieve the registered grant_tier from the Registry registration metadata for the AID identified by that payload's sub. The Relying Party MUST reject the request with grant_tier_insufficient if any participating AID's registered grant_tier is absent, unrecognised, or not permitted for the operation's security Tier.

The permitted runtime mappings are: Tier 1 permits G1, G2, or G3; Tier 2 permits G2 or G3; Tier 3 permits only G3. For Tier 2 and Tier 3, the root principal DID MUST use the did:web method. A higher-assurance Grant Tier MAY satisfy a lower security Tier, but a lower-assurance Grant Tier MUST NOT satisfy a higher security Tier.

If the operation's security Tier is 2 and the Registry's identity_proofing_required_for_tier2 metadata field is true, Tier 2 permits only G3 for this Registry. In that case, a participating AID registered with G2 MUST be rejected with grant_tier_insufficient.

9.17. Step 10: DPoP Validation (Conditional)

For this step, the operation's security Tier is the highest Tier among all requested scopes. A token containing one Tier 2 scope is a Tier 2 token for DPoP validation.

DPoP (Demonstration of Proof-of-Possession) MUST be verified when the operation's security Tier is 2 or 3, when any requested scope has requires_dpop: true in the synced AIP Scope Catalog, or when the

requested Registry endpoint requires DPoP independently of scope metadata. A Tier 2 or Tier 3 operation requires DPoP even if every requested scope has requires_dpop: false.

If DPoP is required, 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 absolute target URI of the current request after removing query and fragment components, with URI normalization performed as specified for DPoP in [RFC9449]. Because htu includes the target origin and path, a DPoP proof captured at one Relying Party or Registry MUST NOT validate at another Relying Party or Registry unless the normalized target URI is identical.

4. The ath claim is present and equals BASE64URL(SHA-256(token)), where token is the exact compact Credential Token or Step Execution Token value from the Authorization header after removing the authorization scheme and surrounding whitespace. When DPoP is required, the authorization scheme MUST be DPoP. The hash input MUST NOT include the authorization scheme, whitespace, or the DPoP proof itself.

5. The iat claim is within the verifier’s accepted DPoP proof window. Unless the verifier requires a server-provided DPoP nonce per [RFC9449], the verifier MUST reject proofs whose iat is more than 300 seconds before receipt time or more than 30 seconds after receipt time. When a server-provided nonce is required, the verifier MUST validate the nonce claim and the server-managed nonce lifetime instead of extending acceptance based on a client-chosen future iat.

6. The jti has not been seen before (DPoP-specific replay cache, separate from Credential Token jti cache, keyed by (kid, jti)). The replay cache is local to the verifier and MUST retain entries for at least the accepted DPoP proof window, or for the server-provided nonce lifetime when nonce validation is used, whichever is longer. AIP does not require cross-Relying Party or cross-Registry synchronization of DPoP replay caches; cross-target replay prevention relies on the mandatory htm, htu, and ath checks above.

7. The public key in the jwk claim matches the effective actor's key material. For an agent-issued Credential Token, this is the agent key that signed the Credential Token. For a Step Execution Token, this is public key material controlled by the actor AID in sub; it is not the Registry step-execution key that signed the SET.

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).

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

This step applies only to Step Execution Tokens that have already passed the SET validation profile defined in this section. Agent-issued Credential Tokens MUST NOT be accepted for Approval Envelope step execution merely because they contain approval-related claims.

The Relying Party MUST call the SET issuer Registry endpoint identified by aip_step_kind. For aip_step_kind: "forward", call GET /v1/approvals/{aip_approval_id}/steps/{n} where {n} is the value of aip_approval_step verbatim. For aip_step_kind: "compensation", call GET /v1/approvals/{aip_approval_id}/compensation-steps/{n} where {n} is the value of aip_compensation_step verbatim. In either case the index is an integer >= 1 and MUST NOT be decremented or adjusted. Both aip_approval_step and aip_compensation_step use 1-based indexing; the first forward step in an envelope is step 1. This is a change from deprecated 0-indexed drafts. Verify:

1. *Cancellation Status:* If the Registry response indicates the Approval Envelope or referenced step has status "cancelled", reject with engagement_cancelled.

2. *Step Status:* The step's status field MUST be "claimed". If status is "pending", the step has not been claimed and cannot execute; reject with approval_step_not_claimed. If status is "completed", "failed", "compensated", "skipped", or "cancelled", reject with approval_step_invalid. Prerequisite failures are reported at claim time by the Registry using approval_step_prerequisites_unmet; a Relying Party that sees an unclaimed step during Step Execution Token verification treats it as a state conflict rather than an authorization denial.

3. *Actor Match:* The returned forward or compensation step's actor field MUST equal the Step Execution Token's sub (the actor AID). Reject if mismatch with approval_step_invalid.

4. *Relying Party Match:* The returned 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.

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

6. *Scope Match:* The Step Execution Token aip_scope set MUST be identical to the returned step's scopes set. Order is not significant for this comparison, but duplicate scope strings are invalid under the SET schema. If the sets differ, reject with approval_step_invalid.

9.19. 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 /v1/registry-metadata endpoint.

For Tier 3 operations:

1. *mTLS Client Certificate:* The HTTP connection MUST use mutual TLS. Validate the client certificate using the Tier 3 mTLS Certificate Profile in Section 21.6. The effective actor AID is the Credential Token iss for agent-issued Credential Tokens, or the Step Execution Token sub for SETs. If the client certificate is absent, reject with mtls_required. If certificate path validation, key usage, EKU, SAN uniqueness, or SAN-to-actor matching fails, 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, or perform the equivalent deployment-profile revocation check allowed by Section 21.6. If the certificate is revoked, reject with agent_revoked. If revocation status is unavailable, stale, or indeterminate, reject with invalid_token.

Step 11a. Principal Assertion Presence Check: For Tier 3 agent-issued Credential Tokens, the Credential Token MUST contain an aip_principal_assertion claim. Absence MUST result in rejection with enterprise_assertion_missing.

Step 11b. Principal Assertion Signature Verification: The Relying

Party MUST resolve the iss claim of the aip_principal_assertion JWT to an OIDC provider configuration document and retrieve the provider's JWKS. The assertion signature MUST be verified against the matching kid in the JWKS. Failure MUST result in rejection with enterprise_assertion_invalid.

Step 11c. Principal Binding Check: The sub claim of the verified aip_principal_assertion MUST match the root Principal Token's principal.id value in aip_chain[0]. A mismatch indicates that the agent is presenting human context that does not match its registered delegation chain and MUST result in rejection with enterprise_assertion_principal_mismatch.

If any Tier 3 check fails, reject with the appropriate error code (mtls_required, invalid_token, agent_revoked, enterprise_assertion_missing, enterprise_assertion_invalid, or enterprise_assertion_principal_mismatch).

9.20. Step 12: Accept

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

9.21. Step Execution Token Validation Profile

A Step Execution Token (SET) is a Registry-issued JWT. It MUST NOT be validated by applying Credential Token Step 2 and Step 3 literally, because its iss is the Registry ID and its kid is a Registry step-execution key URI, not a did:aip agent key. A Relying Party that receives a token for Approval Envelope step execution MUST use this profile before applying Step 10a.

1. Parse the token as a JWT per Step 1.

2. Decode the JOSE header and payload without using either for acceptance. The payload MUST conform to step-execution-token.schema.json. The iss claim MUST be an HTTPS Registry ID URI, sub MUST be the actor AID, and aip_approval_id and aip_step_kind MUST both be present. When aip_step_kind is "forward", aip_approval_step MUST be present and aip_compensation_step MUST be absent. When aip_step_kind is "compensation", aip_compensation_step MUST be present and aip_approval_step MUST be absent. If aip_registry is present, it MUST equal iss. If any of these checks fail, reject with invalid_token.

3. Validate the SET header. typ MUST be "AIP-SET+JWT", alg MUST be "EdDSA", and kid MUST be an HTTPS URI whose origin matches the iss Registry ID origin. The protected header MUST also contain integer aip_trv. If any check fails, reject with invalid_token.

4. Apply the expiration preflight from Step 2a.

5. Resolve the issuing Registry trust state from local pinned trust records only. The Relying Party MUST have previously completed first-contact bootstrap or sequential trust update for the iss Registry ID under Section 7.4.4 and Section 7.4.7.1. It MUST select the locally retained Registry Trust Record whose signed.registry_id equals iss and whose signed.version equals the SET protected header aip_trv. The Relying Party MUST NOT fetch /v1/registry-metadata or a Registry Trust Record during SET validation to establish trust for the SET being validated. If no matching pinned trust record exists, or if the pinned trust record is expired and has not already been refreshed by a completed trust-update procedure, reject with registry_untrusted.

6. The SET header kid MUST exactly match the keyid of a JWK listed in active_verification_keys.step_execution of the accepted Registry Trust Record version. If no such key is listed, reject with invalid_token.

7. Verify the SET signature using the matched step-execution JWK. If signature verification fails, reject with invalid_token.

8. Apply Steps 5a through 5f and Step 6 to the SET common claims. Step 5g is replaced for SETs by the following SET-specific binding: the payload iss MUST be the HTTPS Registry ID whose pinned Registry Trust Record was selected by protected header aip_trv; the payload sub MUST match the did:aip ABNF and identify the actor AID; and the protected header kid MUST identify a Registry step-execution key listed in that selected Trust Record. The SET replay cache is keyed by (iss, jti), where iss is the Registry ID. A Registry MUST generate SET jti values that are unique within that Registry ID across all issued Step Execution Tokens. Registry complete/fail endpoints MAY return a stored idempotent response for an exact retry of a previously completed transition, but MUST NOT re-execute the transition or accept a replayed SET as fresh authority.

9. Apply Step 6a when required, treating the SET issuer Registry iss as the Registry whose trust anchor is being verified. For Tier 2 or Tier 3 operations, or whenever aip_registry is present, the root principal DID Document's AIPRegistry service endpoint MUST match the SET issuer Registry by origin comparison.

10. Apply Step 6b to any aip_engagement_id, using sub as the actor AID whose engagement participation is being checked.

11. Apply Step 7 to the actor AID in sub, not to the Registry ID in iss. Apply Step 8 to aip_chain, except that Step 8 Post-Check A and B are replaced by the rule that aip_chain[n-1].sub MUST equal the SET sub.

12. Apply Steps 9 through 9d using the actor AID in sub as the effective agent AID for manifest lookup, scope verification, delegation inheritance, and Grant Tier Conformance.

13. Apply Step 10 when DPoP is required. For SETs, the DPoP proof key MUST match public key material controlled by the actor AID in sub, not the Registry step-execution key that signed the SET.

14. Apply Step 10a, Step 11 using sub as the effective actor AID, and Step 12.

A token with typ: "AIP+JWT" MUST NOT be accepted as a Step Execution Token. A token with typ: "AIP-SET+JWT" MUST NOT be accepted as an agent-issued Credential Token. This type separation prevents substitution between agent-issued and Registry-issued authorisation artifacts.

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. Subject to the Tier-specific Principal DID Method requirements in Section 20, did:web, did:key, or proprietary

DID methods can be acceptable for Tier 1.

The maximum delegation depth is a hard constraint of 10. Delegation depth is the zero-based delegation_depth value carried in each Principal Token and MUST equal that token's index in the aip_chain array. A valid chain therefore contains at most 11 Principal Tokens: indexes 0 through 10. An agent MUST NOT delegate to a sub-agent if doing so would create a token with delegation_depth greater than 10 or an aip_chain longer than 11 elements.

An agent MUST NOT delegate to itself. For every delegated Principal Token with delegation_depth greater than 0, the delegated_by AID MUST identify the immediate parent agent and MUST NOT equal the token's sub AID. Relying Parties enforce this prohibition during Step 8e and Step 8g of Credential Token validation.

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 greater than 3 and no greater than 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 equal to or more restrictive than the corresponding values in the parent's manifest.

Constraint comparison uses the recursive attenuation semantics from Rule CO-1. Numeric caps and thresholds use the ordering defined for that field; booleans, arrays, strings, enums, omitted fields, and unknown types follow the type-specific CO-1 rules.

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 using the ordered Step 9c algorithm in the Credential Token validation algorithm (see Section 9.15). 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, the Relying Party MUST fetch a fresh manifest.

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 9.15) 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.

10.4. Delegated Identity Chaining for A2A Workflows

Agent-to-agent asynchronous workflows require Relying Parties to distinguish the acting agent from the originating agent that delegated the work. When Agent B makes a tool call on behalf of Agent A, Agent B's Credential Token MUST carry both identities using the profile in this section.

1 Agent B's Credential Token aip_chain array MUST include Agent A's Principal Token as an additional chain element after Agent B's own root Principal Token.

2 The aip_scope in Agent B's Credential Token MUST be a strict subset of the scopes granted in Agent A's Capability Manifest. Rule D-1 enforces this at delegation time; this rule applies the same attenuation requirement to the presented Credential Token.

3 When Agent B is acting in a delegated A2A context, Agent B's Credential Token MUST include the aip_originator_aid claim. The value of aip_originator_aid MUST be Agent A's AID: the AID of the agent that initiated the workflow and delegated the task to Agent B. The aip_originator_aid value MUST conform to the did:aip ABNF defined in Section 4.1. The aip_originator_aid value MUST NOT equal the sub claim of Agent B's Credential Token. When aip_originator_aid is present but does not meet these constraints, the Relying Party MUST reject the request with a2a_originator_invalid.

4 Relying Parties MAY use aip_originator_aid to apply originator-level access policies and to index audit events by the originating agent. Full chain validation under Section 9.11 remains REQUIRED for every accepted Credential Token.

5 Relying Parties receiving a Credential Token that contains aip_originator_aid MUST perform the following validation:

   a. Verify aip_originator_aid conforms to the did:aip ABNF in Section 4.1. If not, reject with a2a_originator_invalid.

   b. Verify aip_originator_aid does not equal the Credential Token sub claim. If equal, reject with a2a_originator_invalid.

   c. Verify aip_originator_aid appears as the sub of aip_chain[0] or as a registered AID in the validated delegation chain, so that Agent A is demonstrably the originator of the chain. If no such binding exists, reject with a2a_originator_invalid.

d. Relying Parties MAY use aip_originator_aid to apply originator-level access policies and audit classification for the already validated request context. Every new Credential Token or Step Execution Token presented on a downstream call MUST still pass the full validation algorithm, including Section 9.11.

This profile aligns with RFC 8693 actor-token semantics: the signing Credential Token identifies the current actor, while aip_originator_aid and the additional Principal Token context allow enterprise audit systems to attribute tool calls to the originating agent chain.

Relying Parties that support enterprise audit logging SHOULD record aip_originator_aid in their audit trail as the chain originator for all A2A tool calls.

11. Revocation Management

Revocation provides the kill switch for agent identity. When an agent is revoked, its Credential Tokens and Step Execution Tokens MUST be rejected as soon as the applicable revocation check observes the active revocation. Tier 2 and Tier 3 checks are live and fail closed on Registry unavailability. Tier 1 uses bounded-staleness CRLs, so enforcement can lag until the next fresh CRL within the SLA defined in this section.

11.1. Revocation Object

Revocation is performed by submitting a signed Revocation Object to POST /v1/revocations. The Revocation Object fields, required members, type enum, and signing input are defined only in Section 5.7. This section defines processing effects for accepted Revocation Objects; it does not define a second Revocation Object schema.

A principal_revoke with a Principal DID target invalidates every Credential Token and Step Execution Token whose aip_chain[0].principal.id equals that Principal DID. A principal_revoke with an AID target invalidates every token whose aip_chain depends on that AID’s principal authorisation. This effect is independent of propagate_to_children. The propagate_to_children flag controls whether the Registry also materialises descendant Revocation Objects; it does not limit the validity effect of principal_revoke.

11.2. Revocation Submission Validation

A conformant Registry MUST implement POST /v1/revocations for externally submitted Revocation Objects. The request body MUST be a JSON object conforming to revocation-object.schema.json. Unless a check below specifies a more precise error code, failed validation MUST be rejected with revocation_invalid.

The Registry MUST perform the following checks in order:

1. The request MUST use Content-Type: application/json. The object MUST contain all required Revocation Object members, including kid. If propagate_to_children is absent, the Registry MUST process it as false.

2. If revocation_id was already accepted, the Registry MUST compare the Section 2.1 canonical JSON serialization of the submitted object to the stored object. If they are identical, the Registry MUST return HTTP 200 with the stored Revocation Object and MUST NOT apply side effects again. If they differ, the Registry MUST reject with revocation_conflict.

3. The timestamp value MUST be a valid UTC date-time and MUST NOT be more than 300 seconds in the future relative to the Registry's current clock. Registries MAY accept older timestamps, but the revocation_id uniqueness check remains mandatory.

4. The reason value MUST be one of the reason codes in the Revocation Object schema. Externally submitted Revocation Objects MUST NOT use parent_revoked, heartbeat_timeout, or lifecycle_expired; those reason codes are reserved for Registry-generated Revocation Objects.

5. For full_revoke, scope_revoke, and delegation_revoke, target_id MUST be a registered did:aip AID in the authoritative Registry. Unknown targets MUST be rejected with unknown_aid. For principal_revoke, target_id MUST be either a registered AID or a Principal DID associated with at least one stored registration in that Registry.

6. For scope_revoke, scopes_revoked MUST be present and non-empty, and every entry MUST be an active synced AIP Scope Catalog scope or an accepted local/private extension scope that is currently granted by the target's effective Capability Manifest. Invalid scope entries MUST be rejected with invalid_scope. For all other revocation types, scopes_revoked MUST be absent.

7. The Registry MUST construct the target authorization chain from its registration records. That chain consists of the target's root Principal DID and every ancestor AID recorded through the parent-child delegation index. For full_revoke, scope_revoke, and delegation_revoke, issued_by MUST equal either the target's root Principal DID or an ancestor AID in that chain. For principal_revoke targeting a Principal DID, issued_by MUST equal that Principal DID. For principal_revoke targeting an AID, issued_by MUST equal the target AID's root Principal DID. Unauthorized issuers MUST be rejected with revocation_unauthorized.

8. For a DID issuer, kid MUST be a DID URL whose DID portion equals issued_by. The Registry MUST resolve the DID, locate the Ed25519 verification method named by kid, and verify the Revocation Object signature over the Section 5.7 signing input. For a did:aip issuer, the key lookup MUST use the authoritative Registry public-key endpoint for the referenced AID and key version. If the key cannot be resolved, is not Ed25519, is not controlled by issued_by, or the signature fails, the Registry MUST reject with revocation_invalid.

Registry-generated Revocation Objects are not externally submitted through the public POST /v1/revocations endpoint. For parent_revoked, heartbeat_timeout, and lifecycle_expired, the Registry MUST set issued_by to its registry_id, set kid to a JWK listed in active_verification_keys.crl of the Registry Trust Record current at the Revocation Object timestamp, and sign the object with the corresponding private key.

Acceptance MUST be atomic. On success, the Registry MUST store the Revocation Object immutably, update live revocation status, schedule CRL publication within the 15-minute freshness window defined in Section 11.3, and return HTTP 201 with Content-Type: application/json and the stored Revocation Object. If propagate_to_children is true, the Registry MUST recursively materialise child Revocation Objects with reason parent_revoked within 15 seconds. For principal_revoke, token invalidation does not depend on materialised child objects; propagation only adds audit and index records.

11.3. Certificate Revocation List (CRL)

The Registry MUST expose a canonical origin CRL endpoint at GET /v1/crl and MUST publish the preferred CRL retrieval URI as endpoints.crl in both /v1/registry-metadata and the signed Registry Trust Record. The published endpoints.crl value MAY be /v1/crl or an absolute HTTPS URI for a CDN or distributed object store. Relying Parties MUST retrieve CRL documents from the accepted Registry Trust Record's

signed.endpoints.crl value rather than assuming registry_id + "/v1/crl".

The CRL MUST be updated within 15 minutes of a new Revocation Object being accepted. The published CRL retrieval URI MUST be served from a CDN or distributed infrastructure. CDN and object-store distribution is not a trust anchor: CRL documents MUST be signed by a JWK listed in active_verification_keys.crl of the Registry Trust Record version current at issuance time.

The CRL is an AIP signed JSON document, not an X.509 CRL. A successful CRL response MUST use Content-Type: application/aip-crl+json and MUST conform to crl.schema.json. The top-level object MUST contain signed and signatures. The signed object MUST include registry_id, trust_record_version, crl_id, issued_at, next_update, sequence, publication_mode, revocation_count, and revocations. The publication_mode value MUST be "complete", "index", or "segment". In "complete" mode, the revocations array contains the full signed Revocation Objects active for bounded-staleness validation at issued_at. In "index" mode, the revocations array MUST be empty and the signed object MUST contain segments as defined in Section 17.10. In "segment" mode, the revocations array contains only the Revocation Objects assigned to that segment, and the signed object MUST contain segment_id.

The CRL signing input is the RFC 8785 JSON Canonicalization Scheme (JCS) serialization of the signed object only. Each signature entry MUST contain keyid and sig. The keyid MUST match a JWK keyid in active_verification_keys.crl for the Registry Trust Record whose signed.version equals the CRL signed.trust_record_version. The signature algorithm is EdDSA over Ed25519. A Relying Party MUST reject an unsigned, malformed, or unverifiable CRL and MUST NOT treat CDN integrity, TLS termination, or object-store metadata as a substitute for this signature verification.

CRL consumers MUST verify that signed.registry_id equals the accepted Registry Trust Record’s signed.registry_id, that signed.trust_record_version identifies an accepted Registry Trust Record, and that at least one signature verifies under that trust record’s active_verification_keys.crl. The next_update timestamp MUST be no later than 15 minutes after issued_at. Relying Parties MUST NOT use a CRL for new token validations after next_update; if no fresh verifiable CRL is available when one is required, validation MUST fail with registry_unavailable.

CRL documents MUST include active principal_revoke entries by target_id. For Tier 1 validation, Relying Parties MUST check principal_revoke entries whose target_id equals the token’s root

Principal DID and entries whose target_id equals any AID in the token's validated aip_chain. A matching principal_revoke is equivalent to agent_revoked for that token.

11.4. 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: The Credential Token TTL is the effective catalog-derived TTL defined in Section 8.2. Relying Parties validate revocation status at request validation time using a verifiable CRL whose next_update has not passed. A Relying Party MUST refresh its cached CRL before using it for new validations after next_update.

Tier 2 - Real-time revocation: Applies when the highest synced AIP Scope Catalog tier among requested scopes is 2. Real-time Registry check on EVERY request. MUST NOT cache revocation status. DPoP MUST be verified for the request regardless of per-scope requires_dpop metadata. If Registry unreachable: MUST deny and return registry_unavailable.

The live Tier 2 lookup uses GET /v1/agents/{aid}/revocation. A registered AID returns HTTP 200 whether or not it is revoked. The response body is the Revocation Status response defined in Section 17.8. Unknown AIDs return unknown_aid (HTTP 404).

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. A principal_revoke invalidates dependent tokens regardless of this flag; propagation only controls whether descendant Revocation Objects are written for indexing, CRL compactness, and auditability.

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.

11.5. Registry Push Notification Protocol (RPNP)

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

11.5.1. Overview

When RPNP is implemented:

* Registry MUST start the first push delivery attempt within 5 seconds of the event commit time defined in Section 11.5.4

* Push payloads MUST be signed by a JWK listed 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 notification window is the RPNP first-attempt latency (at most 5 seconds) rather than the CRL refresh interval. RPNP does not replace CRL; it supplements it. Confirmed delivery depends on subscriber availability and the retry rules in Section 11.5.4.

11.5.2. Subscription

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

The request body MUST conform to the subscription_request definition in rpnp.schema.json. On success, the Registry assigns a subscription_id with the pattern sub:<uuid-v4> and returns HTTP 201 with a body conforming to the subscription_response definition in rpnp.schema.json.

Field	Required	Constraints
subscriber_did	REQUIRED	MUST be did:web or did:aip
event_types	REQUIRED	Array containing one or more of full_revoke, scope_revoke, delegation_revoke, principal_revoke, and engagement_cancelled
scope_filter	REQUIRED	aid, principal, or all
targets	CONDITIONAL	REQUIRED when scope_filter is aid or principal
webhook_uri	REQUIRED	HTTPS URI
hmac_secret	REQUIRED	Base64url-encoded shared secret; minimum 32 bytes of entropy
subscription_expires_at	REQUIRED	ISO 8601 UTC; max 90 days

Table 19: Subscription Fields

The subscription request MUST be authenticated via DPoP proof bound to subscriber_did. A DPoP proof that only proves possession of an arbitrary key is not sufficient subscription authentication. If the DPoP proof is absent, malformed, unverifiable, replayed, not bound to subscriber_did, or not bound to the current request target, the Registry MUST reject with subscription_auth_required.

The Registry MUST accept exactly the following RPNP subscription authentication profiles:

1 *DID-bound DPoP:* The DPoP proof protected header MUST contain a public verification key and a DID URL key identifier in jwk.kid. The DID portion of jwk.kid MUST equal subscriber_did. The Registry MUST resolve subscriber_did and verify that jwk.kid identifies an Ed25519 verification method controlled by that DID. For did:web, resolution uses the DID method's normal DID Document resolution. For did:aip, resolution uses the authoritative AIP

Registry public-key endpoint for the referenced AID and key identifier. The public key in the DPoP header MUST be byte-for-byte equivalent to the resolved verification method, and the DPoP proof signature MUST verify under that key.

2 *Authorization-bound DPoP:* The request MAY include Authorization: DPoP <token>. In that case, the Registry MUST validate the token using Section 9 with the subscription endpoint as the audience, MUST validate the DPoP proof using Section 9 Step 10 including the ath check, and MUST verify that the token issuer, token subject, and leaf aip_chain subject all equal subscriber_did. This profile is available only when subscriber_did is a did:aip AID.

For either profile, the Registry MUST verify htm, htu, iat, and jti replay state using the DPoP validation rules in Section 9 Step 10. In DID-bound DPoP without an Authorization header, the ath claim MUST be absent. If an Authorization header is present, ath MUST be present and valid for that exact token.

The Registry MUST authorize the requested scope_filter against the authenticated subscriber_did. If scope_filter is "all" and the subscriber is not explicitly authorized by Registry policy for global revocation notifications, the Registry MUST reject the request with subscription_scope_forbidden. If scope_filter is "aid" or "principal", every value in targets MUST be within the subscriber’s authorized observation set; otherwise, the Registry MUST reject with subscription_scope_forbidden.

The Registry MUST enforce configured per-subscriber and global active subscription quotas before accepting a new subscription. If accepting the request would exceed any applicable quota, the Registry MUST reject with subscription_limit_exceeded. The Registry SHOULD include Retry-After when quota exhaustion is temporary.

The hmac_secret value is shared secret material used by the Registry to authenticate push event delivery. The value MUST be transmitted only over TLS. The Registry MUST retain the secret, or a reversibly encrypted form of the secret, for the lifetime of the subscription so it can compute delivery HMACs. A Registry MUST NOT store only a one-way hash of this value while the subscription remains active. Registries MAY additionally store a hash of the secret for audit or lookup purposes, but that hash is not sufficient to sign delivery requests.

GET /v1/subscriptions/{id} MUST return the current subscription_response object. The status field MUST be one of active, degraded, cancelled, or expired. Status responses MUST include consecutive_failures, last_delivery_attempt_at,

last_success_at, next_retry_at, and last_delivery_status, using null for timestamp fields that have not yet occurred. DELETE /v1/subscriptions/{id} MUST atomically transition an active or degraded subscription to cancelled and return the updated subscription_response.

11.5.3. 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 JWK listed 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
aip_trv	Integer Registry Trust Record version whose active_verification_keys.notifications contains kid

Table 20: RPNP JWT Fields

The protected aip_trv header is used to select an already pinned Registry Trust Record for the event issuer. Subscribers MUST NOT fetch new trust state during push-event validation to establish trust for that event.

Payload Field	Description
iss	Registry ID
sub	Affected AID or engagement ID
iat	Unix timestamp
exp	Unix timestamp; MUST be no more than 300 seconds after iat
jti	UUID v4; unique event ID
subscription_id	Subscription identifier receiving this push event
event_type	One of subscribed types
event_data	Event-specific data

Table 21: RPNP Payload Fields

The JWT payload MUST conform to the push_payload definition in rpnp.schema.json. The request MUST include an X-AIP-Signature header using this exact syntax:

X-AIP-Signature: <sig-header>

sig-header = "v1;t=" timestamp ";kid=" subscription-id ";h=" hmac

The HMAC input is the ASCII decimal t value, followed by a period (.), followed by the exact request body octets. The h value is BASE64URL(HMAC-SHA256(hmac_secret, input)) without padding. The kid value MUST equal the JWT subscription_id. The subscriber MUST verify the Registry JWT signature, the HMAC, and the subscription binding before accepting the event.

Subscribers MUST reject a push event if the header timestamp t is more than 300 seconds from the subscriber’s current clock, if the JWT exp is in the past, if exp - iat exceeds 300 seconds, or if the jti has already been accepted for the same subscription_id. Subscribers MUST retain replay state keyed by (subscription_id, jti) at least until the later of the JWT exp time and 300 seconds after receipt.

11.5.4. Delivery Guarantees

RPNP provides at-least-once best-effort push delivery. It does not provide exactly-once delivery, and subscribers MUST use the jti replay rules in this section to make event processing idempotent.

1. The Registry MUST record an event_committed_at instant using its reliable UTC clock when the state transition that creates the push event is durably committed. For revocation events, this is the instant at which the Revocation Object is accepted and live revocation status is committed. For engagement cancellation events, this is the instant at which the cancellation state transition is committed.

2. The Registry MUST start the first HTTPS POST delivery attempt to each active matching subscription no later than 5 seconds after event_committed_at. The interval ends when the Registry begins sending the request to the subscriber's webhook_uri. This 5-second requirement applies to the first delivery attempt, not to confirmed successful receipt.

3. The Registry MUST set last_delivery_attempt_at to the Registry UTC time at which each delivery attempt begins. If the first attempt starts more than 5 seconds after event_committed_at, the Registry MUST record last_delivery_status as first_attempt_sla_missed until a later delivery attempt overwrites it with a more recent status.

4. On delivery failure, meaning timeout or any non-2xx HTTP response, the Registry MUST perform at least three retry attempts after the initial delivery attempt, using exponential backoff intervals of 1s, 2s, and 4s as the minimum retry schedule. The Registry MAY add jitter or continue retrying after those attempts, but MUST NOT schedule the first three retries later than those intervals unless the subscription has expired or been cancelled.

5. A delivery attempt MUST be treated as timed out if the subscriber does not return a final HTTP response within 5 seconds after the Registry finishes sending the request body, unless the Registry applies a shorter deployment-profile timeout.

6. After 3 consecutive failures, mark the subscription degraded, update consecutive_failures, last_delivery_attempt_at, last_delivery_status, and next_retry_at, and continue to make the event available through CRL or live revocation status. Subscribers MUST fall back to CRL or live Registry checks while the subscription is degraded.

7. On any successful 2xx delivery, reset consecutive_failures to 0, set last_success_at, clear next_retry_at, and restore status to active unless the subscription has expired or been cancelled.

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 [RFC6749], adapted for the agent identity use case. Two independent implementations following this section MUST produce interoperable grant interactions.

12.1. Overview and Roles

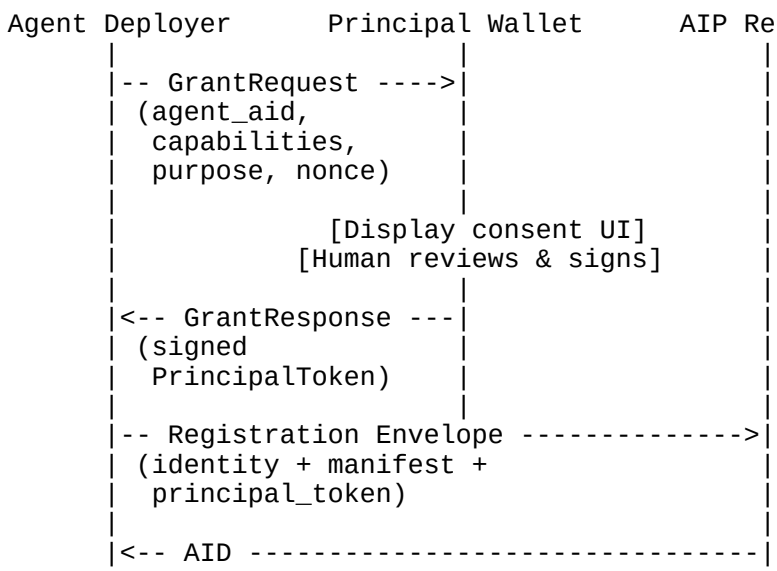
The AIP-GRANT ceremony involves three actors:

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 Principal 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. For G2, the signed request MUST identify the deployer through the deployer_did field and the signature key MUST be bound to that DID. The nonce MUST be cryptographically random and MUST contain at least 128 bits of entropy.

The request MUST also include deployer_name for consent UI display. The deployer_name value is human-readable metadata only; it is not a trust anchor and MUST NOT replace verification of deployer_did and the signed GrantRequest envelope.

GrantRequest Fields:

grant_request_id (string; REQUIRED)
: pattern: gr:<UUIDv4-lowerhex>

aip_version (string; REQUIRED)
: AIP protocol compatibility version; MUST be "0.3"; not the Internet-Draft revision

agent_aid (string; REQUIRED)
: MUST match did:aip ABNF; pre-derived AID of the agent being authorised; Principal Token sub MUST equal this value

```
   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)
      requested maximum delegation lifetime; min: 300; max: 31536000;
      approved value MUST NOT exceed this value

   nonce (string; REQUIRED)
      minLength: 22; cryptographically random

   request_expires_at (string; REQUIRED)
      ISO 8601 UTC

   callback_uri (string; CONDITIONAL)
      REQUIRED for Web Redirect and G3 OAuth authorization requests;
      MUST use HTTPS; for G3, MUST equal the OAuth redirect_uri

   state (string; CONDITIONAL)
      REQUIRED for G3 OAuth authorization requests; echoed unchanged and
      MUST equal the OAuth state parameter

   deployer_did (string; REQUIRED)
      W3C DID of the deployer; used to verify signed GrantRequests and
      displayed in consent UI

   deployer_name (string; REQUIRED)
      minLength: 1; maxLength: 128; displayed alongside deployer_did;
      display metadata only, not a trust anchor

   deployer_public_key (object; OPTIONAL)
      JWK key hint only; MUST match the verification method resolved
      from deployer_did and MUST NOT replace DID resolution
```

The deployer MUST generate the agent's AID before constructing the GrantRequest and MUST include that value as agent_aid. The Principal Token produced from this grant MUST use the original GrantRequest agent_aid as its sub claim.

Signed GrantRequest envelope: A GrantRequest sent directly to a Principal Wallet using the Web Redirect or QR Code binding MUST be carried as a compact JWS whose payload is the UTF-8 JSON serialization of the GrantRequest object. The protected JWS header MUST contain typ equal to "aip-grant+jws", alg equal to "EdDSA", and kid as a DID URL. The DID portion of kid MUST equal the GrantRequest deployer_did. The referenced verification method MUST resolve from the deployer_did DID Document, MUST be controlled by deployer_did, and MUST be an Ed25519 signing key. If deployer_public_key is present, it is a cache hint only; the Principal Wallet MUST verify that it is byte-for-byte equivalent to the resolved verification method before using it and MUST NOT trust a self-supplied key that is not present in the resolved DID Document.

The same signed GrantRequest envelope format is used by the G3 OAuth binding. In that case, the compact JWS is carried in the aip_grant_request authorization request parameter and is verified by the Registry acting as OAuth Authorization Server before any authentication or consent ceremony is started.

12.3. Principal 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.

The Principal Wallet MUST use its own reliable UTC clock for request_expires_at checks and MAY allow a 30-second clock skew tolerance. If the Principal Wallet's current time is more than 30 seconds after request_expires_at, the request is expired. If the Principal Wallet cannot determine reliable current UTC time, it MUST reject with grant_request_invalid rather than issuing a back-dated or future-dated Principal Token.

Signed request checks: For a G2 GrantRequest, the Principal Wallet MUST verify the signed GrantRequest envelope before displaying the consent UI. The Principal Wallet MUST reject with

grant_request_invalid and MUST NOT show the consent UI if the request is unsigned, if deployer_did is absent, if the JWS kid DID does not equal deployer_did, if the deployer DID cannot be resolved, if the referenced key is not an Ed25519 signing key controlled by deployer_did, or if the JWS signature verification fails.

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 using trusted AIP Scope Catalog display metadata

6. Delegation validity - requested duration and the approximate expiry computed from delegation_valid_for_seconds using the Principal Wallet’s current UTC clock

7. *Destructive Operations:* Any scope whose active AIP Scope Catalog entry has destructive: true MUST be highlighted in the UI using the mandatory two-step confirmation flow required for Principal Wallet conformance in Section 23.2.

Before displaying the consent UI, the Principal Wallet MUST resolve requested_capabilities to the concrete requested scope identifiers using the scope-to-capability rules in Section 5.9 and any explicit private extension policy accepted by the Principal Wallet. The Principal Wallet MUST use a trusted immutable AIP Scope Catalog Snapshot for this resolution. For each requested scope, the Principal Wallet MUST find an active or experimental scope entry and MUST find complete trusted display metadata for the Principal Wallet’s selected locale or for the entry’s display.default_locale. If no trusted catalog snapshot is available, if a requested capability cannot be resolved to a concrete scope identifier, if the scope entry is unknown, removed, reserved, or outside an accepted private extension policy, or if the required display metadata is absent or incomplete, the Principal Wallet MUST reject with grant_request_invalid and MUST NOT show the consent UI.

The Principal Wallet MUST NOT synthesize consent text from raw scope identifiers, capability field names, deployer-supplied display strings, generic family names, or untrusted catalog data. The title

value in display.strings[locale].title is the canonical human-readable capability string for that locale. The Principal Wallet MAY also display display.strings[locale].summary, but it MUST NOT use the summary to replace or weaken required destructive-operation highlighting.

If an issued Principal Token includes a purpose claim, the claim value MUST be either the exact GrantRequest purpose value or a shorter summary that was displayed to and approved by the principal before signing. A Principal Token issuer MUST NOT include a deployer-supplied or issuer-generated purpose claim that was not presented during consent.

Principal Token signing profile: For a root Principal Token, the token issuer is always the root principal identified by principal.id. A Principal Wallet or other principal-controlled signing component that constructs a root Principal Token after consent MUST sign the JWT with EdDSA, MUST set iss to principal.id, and MUST use a JOSE header kid identifying an Ed25519 verification method controlled by principal.id. A Registry or authorization server MAY prepare the payload, mediate consent, collect high-assurance authentication context, store the GrantResponse, or deliver the signed token, but it MUST NOT sign a root Principal Token with a Registry-controlled or authorization-server-controlled key and MUST NOT set iss to the Registry or authorization server. The Principal Token payload sub claim MUST equal the agent_aid value from the original GrantRequest. A service that cannot obtain a principal-controlled Ed25519 signature MUST NOT issue a root Principal Token with that principal as iss.

Delegation expiry derivation: The Principal Token issuer MUST compute a single issuance timestamp from its current reliable UTC clock at the moment of signing. It MUST set the Principal Token issued_at claim and the GrantResponse signed_at field to that same instant. The approved delegation lifetime MUST be represented as approved_delegation_valid_for_seconds, MUST be at least 300 seconds, and MUST NOT exceed the original GrantRequest delegation_valid_for_seconds. The issuer MAY approve a shorter lifetime than requested.

The Principal Token expires_at claim MUST equal issued_at plus approved_delegation_valid_for_seconds seconds. No clock skew tolerance is applied to this calculation; clock skew tolerance applies only when validators compare timestamps to their local current time. The issuer MUST NOT use the deployer’s clock, callback receipt time, or request_expires_at as the Principal Token issued_at value.

Draft-02 Catalog display metadata:

The AIP Scope Catalog is the normative source of capability display metadata. Each scope entry's display member is defined in Section 17.15. For standard Draft-02 scopes, display.strings["en-US"].title MUST equal the display string listed below. The corresponding summary value MUST be non-empty and MUST describe the same capability without broadening the authorization represented by the scope. Extension and private scopes MUST provide equivalent catalog display metadata through their catalog entry or accepted private extension policy before a Principal Wallet can present consent for them.

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 22: Draft-02 Standard Scope Display Titles

12.4. GrantResponse Object

The GrantResponse is constructed and signed by the Principal Wallet after the principal approves, partially approves, or declines 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"
signed_at	REQUIRED	ISO 8601 UTC; same instant as Principal Token issued_at when principal_token is present

Table 23: GrantResponse Fields

A GrantResponse with status "approved" or "partial" MUST also include approved_delegation_valid_for_seconds. The value MUST be an integer from 300 through the original GrantRequest delegation_valid_for_seconds, inclusive, and is used to derive the Principal Token expires_at claim.

Deployer validation of GrantResponse: Upon receipt the deployer MUST:

1. Verify grant_request_id matches the original request

2. Verify nonce matches. If mismatch, the deployer MUST reject the GrantResponse with grant_nonce_mismatch; the mismatch MUST be treated as a forgery attempt and no agent is registered.

3. If status is "rejected", the deployer MUST treat the grant as a terminal failure with grant_rejected_by_principal and no agent is registered.

4. Verify approved_delegation_valid_for_seconds is present for "approved" or "partial" responses, is at least 300 seconds, and does not exceed the original GrantRequest delegation_valid_for_seconds

5. Decode and verify the principal_token JWT signature

6. Verify the principal_token payload sub exactly matches agent_aid from the original GrantRequest

7. Verify the principal_token payload issued_at represents the same instant as the GrantResponse signed_at

8. Verify signed_at and issued_at are not more than 30 seconds in the future relative to the deployer’s current UTC clock, and that signed_at is not more than 24 hours in the past

9. Verify the principal_token payload expires_at equals issued_at plus approved_delegation_valid_for_seconds seconds

If any GrantResponse validation check fails, the deployer MUST treat the GrantResponse as invalid and MUST NOT submit a Registration Envelope based on it.

When a deployer, Registry, Principal Wallet, or authorization server returns an AIP error response for a rejected grant, the error code MUST be grant_rejected_by_principal. When it returns an AIP error response for a GrantResponse whose nonce does not match the original GrantRequest, the error code MUST be grant_nonce_mismatch.

12.5. Transport Bindings

Implementations MUST support the Web Redirect Flow.

Web Redirect Flow:

```
https://wallet.example.com/aip-grant
  ?request=<compact-JWS-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 delegation issuer for the child. No new human consent UI is required, because the parent can only delegate scopes and constraints that are already within its own accepted authority.

The child is registered with a delegated Principal Token, not a second root Principal Token. The delegated token’s sub is the child AID, delegated_by is the parent AID, and delegation_depth is the parent’s depth plus one. The Registry validates the submitted Registration Envelope using Registration Check 9 and stores the reconstructed parent-plus-child chain for later runtime validation.

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. This is a local implementation conformance and key-management requirement. The Registry and Relying Parties cannot determine from protocol messages whether the parent retained a copy of the child’s private key, and MUST NOT treat private-key deletion as a Registry acceptance check or Credential Token validation precondition. Deployment profiles that audit this requirement MUST collect the sub-agent provisioning and deletion evidence defined in Section 21.4.

For sub-agent delegation, a parent agent MAY include a purpose claim in the child’s Principal Token to describe the delegated work. That claim is signed audit metadata only. It MUST NOT expand the child agent’s scopes or constraints and MUST NOT be treated by Relying Parties as a substitute for Capability Manifest or Approval Envelope validation.

12.7. AIP-GRANT Error Codes

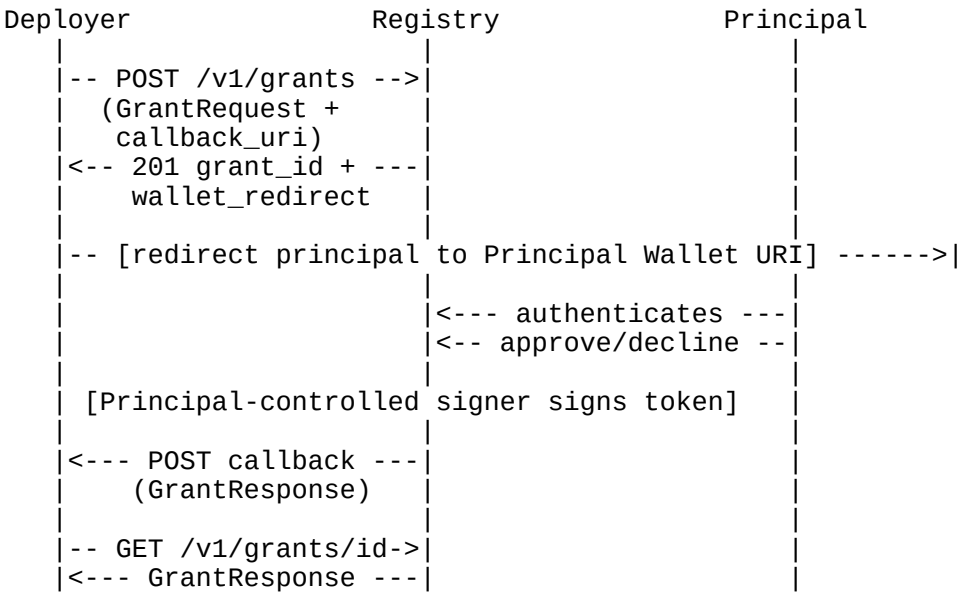
Code	Description
grant_request_expired	request_expires_at has passed
grant_request_replayed	grant_request_id seen before
grant_request_invalid	GrantRequest malformed or signature failed
grant_rejected_by_principal	Principal declined the grant
grant_nonce_mismatch	GrantResponse nonce does not match

Table 24: 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 Principal 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 obtain a root Principal Token signed using the Principal Token signing profile above, with iss equal to the approving principal’s DID and sub equal to the stored GrantRequest agent_aid. The Registry MAY construct the unsigned payload and present it to a principal-controlled Principal Wallet or signing component, but the final compact JWT MUST be signed by an Ed25519 verification method controlled by principal.id. The Registry MUST store the resulting GrantResponse without altering or re-signing the Principal Token.

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.

If grant_id does not identify a stored G1 grant, or if the referenced grant is expired and no GrantResponse is retrievable, the Registry MUST reject the request with grant_not_found.

If grant_id identifies a stored, retrievable G1 grant but the authenticated requester does not match the deployer_did in the original GrantRequest, the Registry MUST reject the request with grant_deployer_mismatch.

If grant_id identifies a stored G1 grant whose principal outcome is "rejected", the Registry MUST reject the retrieval request with grant_rejected_by_principal. If the stored GrantResponse nonce does not match the original GrantRequest nonce, the Registry MUST reject the retrieval request with grant_nonce_mismatch and MUST NOT return the stored 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 Principal 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, derives the agent AID, and constructs a GrantRequest that includes agent_aid and deployer_did.

2. The deployer signs the GrantRequest as a compact JWS using the Ed25519 key identified by a kid under deployer_did.

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

4. The Principal Wallet resolves deployer_did, verifies the signed GrantRequest envelope, and presents the consent UI to the principal only after successful verification.

5. On approval, the Principal Wallet signs the Principal Token using the Principal Token signing profile above 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.

OAuth binding of the GrantRequest: Before initiating the OAuth authorization request, the deployer MUST construct and sign a GrantRequest using the signed GrantRequest envelope defined in Section 12.2. The G3 OAuth authorization request MUST carry that compact JWS in an aip_grant_request form parameter. The authorization request MUST also include response_type=code, client_id, redirect_uri, scope, state, code_challenge, code_challenge_method=S256, acr_values, and aip_version=0.3.

```
POST /v1/oauth/authorize
Content-Type: application/x-www-form-urlencoded

response_type=code
&client_id=did:web:assistant.example.com
&redirect_uri=https://assistant.example.com/aip/grant-callback
&scope=email.read%20calendar.read
&state=sess_abc123_csrf_xyz789
&code_challenge=<S256-code-challenge>
&code_challenge_method=S256
&acr_values=<requested-acr>
&aip_version=0.3
&aip_grant_request=<compact-JWS-signed-GrantRequest>
```

The Registry Authorization Server MUST verify the signed GrantRequest envelope before redirecting the principal to a Principal Wallet or displaying any consent UI. Verification MUST include the signed GrantRequest checks in Section 12.2 and the following G3 binding checks:

1 The aip_grant_request parameter is present and contains a compact JWS whose payload is a GrantRequest object.

2 The OAuth client_id either equals the GrantRequest deployer_did or identifies a pre-registered OAuth client whose Registry metadata maps to that same deployer_did.

3 The OAuth redirect_uri exactly matches the GrantRequest callback_uri.

4 The OAuth state exactly matches the GrantRequest state.

5 The GrantRequest request_expires_at has not passed, allowing at most 30 seconds of clock skew.

6 The OAuth scope parameter does not request authority outside the GrantRequest requested_capabilities. If the Authorization Server cannot map a requested scope string to the GrantRequest capabilities under the synced AIP Scope Catalog, it MUST treat the authorization request as invalid.

If any binding check fails, the Authorization Server MUST reject the authorization request with grant_request_invalid and MUST NOT issue an authorization code. After successful verification, the Authorization Server MUST store an immutable authorization-code binding that includes the JCS hash of the GrantRequest payload, the compact JWS signature value, grant_request_id, agent_aid, deployer_did, nonce, state, callback_uri, requested_capabilities, and delegation_valid_for_seconds. The token endpoint MUST use this stored binding when redeeming the authorization code and MUST reject any code that lacks such a binding with grant_request_invalid.

G3 token endpoint response mapping: A successful POST /v1/oauth/token response for a G3 grant ceremony is an OAuth token response that transports, but does not replace, an AIP GrantResponse. The response body MUST be a JSON object with the following fields:

* token_type: REQUIRED. MUST be "AIP+JWT".

* access_token: REQUIRED. The compact Principal Token JWT. This value is an OAuth transport alias only.

* expires_in: REQUIRED. Integer seconds until the Principal Token expires; MUST equal grant_response.approved_delegation_valid_for_seconds.

* state: REQUIRED. MUST equal the OAuth state value stored in the authorization-code binding.

* grant_response: REQUIRED. A JSON object conforming to the GrantResponse schema in Section 12.4 with status equal to "approved" or "partial".

The access_token value MUST be byte-for-byte identical to grant_response.principal_token. The deployer MUST validate the nested grant_response using the GrantResponse validation rules in Section 12.4 and MUST use grant_response.principal_token when constructing the Registration Envelope. The access_token field exists only for OAuth client interoperability and MUST NOT be interpreted as replacing the principal_token field. The Principal Token payload MUST include the acr and amr claims from the completed G3 ceremony. If the principal rejects the grant or the ceremony fails before approval, the Authorization Server MUST return an OAuth error response and MUST NOT return a successful token response.

Step-by-step definition:

1. The deployer constructs and signs a GrantRequest, then initiates an OAuth 2.1 Authorization Code Flow with PKCE at the Registry’s authorization endpoint using the G3 OAuth binding above.

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

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

4. The Principal Wallet returns an authorization code or equivalent signed ceremony result to the Registry.

5. The Registry MUST obtain a root Principal Token signed using the Principal Token signing profile above by a principal-controlled Ed25519 verification method. The Registry MAY prepare the token payload and include the high-assurance acr and amr values from the OAuth ceremony, but it MUST NOT sign the root Principal Token with the Registry’s authorization-server key and MUST NOT change iss away from principal.id. The token sub MUST equal the agent_aid value from the stored authorization-code binding, and the token MUST include acr and amr claims from the completed ceremony.

6. The token endpoint returns the G3 token endpoint response defined above. Its grant_response member is the AIP GrantResponse used by the deployer for Registration Envelope construction.

7. 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 with a catalog-derived TTL 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

Approval Envelopes have two wire profiles. An ApprovalEnvelopeSubmission is submitted by the orchestrating agent to POST /v1/approvals. A StoredApprovalEnvelope is returned by Registry read and mutation endpoints and includes Registry-managed lifecycle fields. A Registry MUST validate POST /v1/approvals request bodies against the submission profile and MUST NOT accept read-only stored fields in the submission body.

Field	Required	Constraints
approval_id	REQUIRED	pattern: apr:<UUIDv4-lowerhex>
created_by	REQUIRED	AID of orchestrating agent
creator_kid	REQUIRED	DID URL identifying the Ed25519 verification method controlled by created_by

principal_id	REQUIRED	W3C DID; MUST NOT be did:aip
engagement_id	OPTIONAL	pattern: eng:<UUIDv4-lowerhex>; links to parent Engagement Object
description	REQUIRED	minLength: 1; maxLength: 512
created_at	REQUIRED	ISO 8601 UTC; set by orchestrating agent; MUST NOT be in the future beyond 30-second clock skew
approval_window_expires_at	REQUIRED	ISO 8601 UTC; MUST be strictly after created_at; MUST NOT be more than 72 hours after created_at
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, treating omitted step values as 0
currency	OPTIONAL	ISO 4217; pattern: ^[A-Z]{3}$
notification_uri	OPTIONAL	HTTPS URI for principal notification callbacks; included in creator and principal signing inputs when present
creator_signature	REQUIRED	base64url EdDSA signature

Table 25: ApprovalEnvelopeSubmission Fields

An ApprovalEnvelopeSubmission MUST NOT contain top-level status, principal_signature, or approved_at. Its forward steps MUST NOT contain read-only status, claimed_at, or completed_at fields. Its compensation steps MUST NOT contain read-only status fields. If any read-only field is present in the submission body, the Registry MUST reject with approval_envelope_invalid.

Field	Required	Constraints
status	REQUIRED	Registry-managed lifecycle state; initially pending_approval
principal_signature	CONDITIONAL	Set only by successful principal approval; required for approved, executing, completed, compensating, compensated, and failed envelopes
principal_kid	CONDITIONAL	DID URL identifying the principal verification method; required whenever principal_signature is present
approved_at	CONDITIONAL	Registry timestamp set with the winning approval transition; required whenever principal_signature is present
steps[].status	REQUIRED	Registry-managed step lifecycle state; initially pending

steps[].claimed_at	CONDITIONAL	Registry timestamp set when a Step Execution Token is issued
steps[].completed_at	CONDITIONAL	Registry timestamp set when a forward step completes
compensation_steps[].status	REQUIRED when compensation_steps is present	Registry-managed compensation lifecycle state; initially pending

Table 26: StoredApprovalEnvelope Registry 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. A step that omits value contributes 0 to this sum. Required and optional steps are both included in the sum when they carry a value field. This value is the maximum approved financial exposure of the envelope, not a prediction that every optional step will execute.

When present, engagement_id scopes the Approval Envelope to the parent Engagement Object. The field is part of the Approval Envelope body and MUST NOT be supplied only as external Registry metadata. Both creator_signature and principal_signature signing inputs MUST include engagement_id whenever it is present.

The creator_signature signing input is the Section 2.1 canonical JSON serialization of the ApprovalEnvelopeSubmission object with creator_signature set to "". The principal_signature signing input is the same immutable submission object after successful submission, including creator_signature, principal_kid, and a principal_signature member set to "". Both signing inputs include creator_kid, notification_uri, and engagement_id when present. The DID portion of creator_kid MUST equal created_by; the DID portion of principal_kid MUST equal principal_id. Neither signing input includes Registry-managed top-level lifecycle fields (status or approved_at) or Registry-managed step fields (status, claimed_at, or completed_at).

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
scopes	REQUIRED	minItems: 1; uniqueItems: true; each item is an AIP scope string required for this step
description	REQUIRED	minLength: 1; maxLength: 256
required	REQUIRED	boolean; false = optional step
triggered_by	REQUIRED	null or step_index; 0 forbidden
compensation_index	OPTIONAL	1-based pointer to a compensation step; null or absent means no compensation action is defined
value	OPTIONAL	Financial value; minimum: 0; omitted means 0 for total_value validation
currency	OPTIONAL	ISO 4217 currency code for this step’s value; when present, MUST match envelope currency
status	READ-ONLY	pending | claimed | completed | failed | compensated | skipped | cancelled

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 MUST NOT be claimed before step N reaches completed status. Circular dependencies MUST be rejected.

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

The compensation_index field, when present on a forward step, identifies the single compensation step that rolls back that forward step. The value is a 1-based identifier that MUST equal the compensation_index field of exactly one entry in compensation_steps; it is not a zero-based JSON array offset. The Registry MUST reject envelopes where a non-null compensation_index does not reference an existing compensation step, or where more than one forward step references the same compensation step.

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.

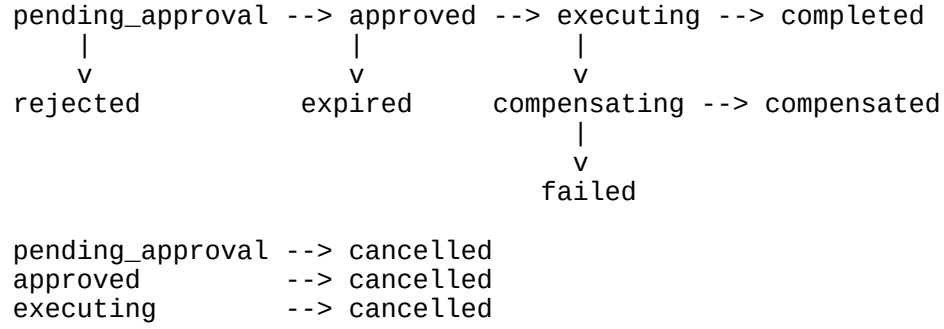
A compensation step is selected only through a forward step’s compensation_index field. The Registry MUST NOT infer a compensation relationship from matching action_type values, array position, actor identity, or relying party URI.

Field	Required	Constraints
compensation_index	REQUIRED	1-based; unique within compensation_steps
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
scopes	REQUIRED	minItems: 1; uniqueItems: true; each item is an AIP scope string required for this compensation step
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 atomically stores principal_signature, sets approved_at, and transitions the envelope to approved. Only envelopes currently in pending_approval can make this transition.

pending_approval to rejected
: Principal declines.

pending_approval to expired
: approval_window_expires_at passes before a successful approval transition is committed.

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.

pending_approval, approved, or executing to cancelled
: Parent Engagement Object transitions to terminated. The Registry MUST NOT transition Approval Envelopes to cancelled solely because the parent Engagement Object transitions to completed.

The approval_window_expires_at field is an approval-deadline only. It controls whether an envelope is eligible to transition from pending_approval to approved. Once an envelope has transitioned to approved before that deadline, later passage of approval_window_expires_at MUST NOT transition the envelope to expired and MUST NOT be used to reject step claims. Execution time limits, if any, MUST be expressed by a separate field or by endpoint policy; they MUST NOT be inferred from approval_window_expires_at.

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

The Registry MUST treat approval, rejection, expiry, cancellation, and first step claim as mutually exclusive state transitions over the same Approval Envelope lifecycle record. Implementations MUST use

optimistic locking, compare-and-set, a database transaction, or an equivalent distributed lock so that exactly one concurrent transition out of pending_approval can succeed. If multiple Principal Wallets submit valid approval signatures concurrently for the same principal_id, the first committed approval wins. The Registry MUST NOT replace principal_signature, approved_at, or the winning state with a later approval request. A later approval or rejection request that races with or follows a completed lifecycle transition MUST fail with approval_state_conflict, except that an exact byte-for-byte retry of the already stored approval response MAY be treated as idempotent and return the stored envelope.

When an envelope transitions to cancelled, pending forward steps under that envelope MUST transition directly to cancelled. Claimed forward steps MUST be treated as failed for compensation-trigger evaluation, then MUST transition to cancelled after any required compensation resolution completes or fails. Completed, failed, compensated, and skipped steps retain their existing status for audit purposes. The Registry MUST NOT issue new Step Execution Tokens for cancelled envelopes or cancelled steps.

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. Step status is pending
3. All triggered_by steps are completed
4. Token iss matches step actor
5. Credential Token passes validation
6. Credential Token aip_scope contains every scope listed in the step’s scopes array
7. action_hash matches stored value

The Registry MUST NOT evaluate approval_window_expires_at as a step-claim condition for an envelope already in approved or executing status.

If the triggered_by check fails because one or more prerequisite steps have not reached "completed", the Registry MUST reject the claim with approval_step_prerequisites_unmet and MUST NOT issue a Step Execution Token.

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

```
header = {
  "typ": "AIP-SET+JWT",
  "alg": "EdDSA",
  "kid": "<Registry step-execution keyid>",
  "aip_trv": <Registry Trust Record version>
}

payload = {
  "iss": "<Registry ID>",
  "sub": "<actor AID>",
  "aud": "<relying_party_uri>",
  "iat": <now>,
  "exp": <now + 300>,
  "jti": "<UUID>",
  "aip_version": "0.3",
  "aip_scope": ["transactions"],
  "aip_chain": ["<compact Principal Token JWT>"],
  "aip_approval_id": "<approval_id>",
  "aip_step_kind": "forward",
  "aip_approval_step": <step_index>
}
```

A Step Execution Token is not an agent-issued Credential Token. Its iss is the Registry ID HTTPS URI, its sub is the claimed step actor AID, and its JOSE kid identifies a step-execution verification key from the Registry Trust Record. Relying Parties MUST validate Step Execution Tokens using the SET validation profile in Section 9 before applying Step 10a. The Registry MUST derive the SET aip_scope value from the claimed step's scopes array and MUST derive the SET exp value using the TTL rule in Section 5.4.1.

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 before the claim deadline. The claim deadline is claimed_at + step_claim_timeout_seconds, where step_claim_timeout_seconds is the Registry policy value published in /v1/registry-metadata. The value MUST be an integer from 1 to 600 seconds inclusive.

If a claimed step is not completed or failed before that deadline, the Registry MUST treat the claim as timed out, transition the step to failed, and apply the SAGA compensation rules as if the actor had explicitly reported step failure.

The JSON body profiles for claim, complete, and fail requests and responses are defined in approval-step-endpoints.schema.json. All request bodies in this section MUST use Content-Type: application/json. The Registry MUST ignore any client-supplied completion or failure timestamp for lifecycle purposes; claimed_at, completed_at, and failure timestamps are Registry receipt times.

Claim request: A forward step claim request body MUST conform to the claim_request profile and contain credential_token and action_parameters. The Registry MUST validate the Credential Token using Section 9 with the Registry claim endpoint as the Relying Party. The token issuer, token subject, and leaf Principal Token subject MUST equal the step actor. The Registry MUST recompute the step action_hash from action_parameters using Section 13.7 and MUST reject a mismatch with approval_step_action_mismatch. The Credential Token aip_scope set MUST contain every scope in the claimed step's scopes array; otherwise the Registry MUST reject with insufficient_scope.

On a successful forward claim, the Registry MUST atomically transition the forward step from pending to claimed, set claimed_at, store the issued SET jti, transition the envelope from approved to executing if this is the first claimed step, and return a claim_response containing the compact Step Execution Token. If the same actor repeats the same claim request with the same idempotency_key before the claim expires, the Registry MAY return the stored claim response. A different actor, different request body, expired claim, or missing matching idempotency record MUST NOT be treated as an idempotent retry.

Complete request: A complete request body MUST conform to the complete_request profile and contain step_execution_token. The Registry MUST validate the SET using the Section 9 SET validation profile, verify that the SET aip_approval_id and step identifier match the request path, verify that the SET sub equals the step actor, verify that the SET jti equals the stored jti for the active claim, and verify that the step or compensation step is currently claimed and has not passed its claim deadline. If any of these checks fail, the Registry MUST reject with approval_step_invalid or approval_step_not_claimed as applicable.

On a successful forward-step complete request, the Registry MUST transition the step to completed, set completed_at, persist any supplied result_hash or evidence_uri as audit metadata, and evaluate envelope completion. If all required forward steps are completed and no compensation is pending, the Registry MUST transition the envelope to completed. Exact retries with the same idempotency_key MAY return the stored complete_response; conflicting repeats MUST reject with approval_state_conflict.

Fail request: A fail request body MUST conform to the fail_request profile and contain step_execution_token and failure_code. The Registry MUST perform the same SET, actor, path, stored-jti, state, and deadline checks required for completion. On success, the Registry MUST transition the step to failed, persist the supplied failure metadata for audit, and apply the SAGA compensation rules. Exact retries with the same idempotency_key MAY return the stored fail_response; conflicting repeats MUST reject with approval_state_conflict.

Compensation endpoints: POST /v1/approvals/{id}/compensation-steps/{n}/claim, /complete, and /fail use the same request and response profiles as forward-step endpoints. The path {n} is the compensation step's compensation_index. Compensation claims are valid only when the envelope is compensating and the compensation step has been triggered by the SAGA rules in Section 13.9. SETs issued for compensation claims MUST contain aip_step_kind: "compensation" and aip_compensation_step, and MUST NOT contain aip_approval_step.

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. Build the compensation set from forward steps whose status is completed and whose compensation_index is non-null. The failed forward step is included in this set only if its status had previously reached completed before the failure was reported.
3. Sort that forward-step set by descending step_index.
4. For each sorted forward step, trigger the compensation step referenced by that forward step's compensation_index.
5. Notify the orchestrating agent

Compensation steps that are not referenced by any completed forward step MUST NOT be triggered. If the failed required step has no completed side effects and no completed predecessor has a non-null compensation_index, the Registry MUST transition the envelope directly to failed rather than compensating. If a completed forward step references a compensation step that cannot be found, the envelope is invalid and the Registry MUST treat the compensation attempt as failed.

Rollback ordering is based on the forward steps' descending step_index values, not on compensation_index order. The compensation_index value is used only to dereference the pre-authorised compensation step selected for each completed forward step.

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). The Registry MUST set the failed compensation step status to failed, set the envelope status to failed, persist an audit record containing the approval ID, compensation index, actor AID, failure timestamp, and failure reason, and return or expose compensation_failed for the failed transition.

The Registry MUST create a principal notification event for the terminal compensation failure. If the Approval Envelope contains a notification_uri, the Registry MUST attempt delivery to that URI. If RPNP is supported and a matching active subscription exists, the Registry MUST also emit an RPNP event. If no delivery channel is configured, the Registry MUST retain the notification event and expose it in the Approval Envelope status response until acknowledged by local policy. Human remediation of the failed business action remains outside AIP, but the failure state, audit record, and notification event are AIP protocol obligations.

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 the exact ASCII prefix did:aip:; reject with approval_envelope_invalid. This is a byte-for-byte prefix comparison against the first eight characters of the string; values whose first eight characters are did:aip: identify an agent AID and are not valid principal identifiers.

2. engagement_id is present but malformed: reject with approval_envelope_invalid. If engagement_id references an Engagement Object that does not exist, reject with engagement_not_found.

3. steps contains circular dependencies in triggered_by: reject with approval_envelope_invalid.

4. steps contains duplicate step_index values: reject with approval_envelope_invalid.

5. total_value != sum of step values, treating omitted step value fields as 0: reject with approval_envelope_invalid.

6. Any present step currency differs from the envelope currency: reject with approval_envelope_invalid.

7. approval_window_expires_at is in the past: reject with approval_envelope_expired.

8. created_at is in the future beyond 30-second clock skew: reject with approval_envelope_invalid.

9. approval_window_expires_at is not strictly after created_at, or is more than 72 hours after created_at: reject with approval_envelope_invalid.

10. compensation_index references a non-existent index: reject with approval_envelope_invalid.

11. compensation_steps contains duplicate compensation_index values or values that are not sequential starting at 1: reject with approval_envelope_invalid.

12. More than one forward step references the same non-null compensation_index: reject with approval_envelope_invalid.

13. creator_signature does not verify: reject with approval_envelope_invalid.

14. created_by agent is revoked: reject with agent_revoked.

15. steps contains more than 20 elements: reject with approval_envelope_invalid.

16. triggered_by is 0: reject with approval_envelope_invalid. An absent triggered_by field is a schema violation because the field is REQUIRED by the step schema and is rejected before this semantic validation rule is evaluated.

17. A forward or compensation step omits scopes, includes an empty scopes array, includes duplicate scope strings, or includes a scope that is not active in the synced AIP Scope Catalog: reject with approval_envelope_invalid.

The Registry MUST also reject envelopes with insufficient_scope unless the submitting orchestrating agent's Credential Token and current Capability Manifest both include the active approvals.create scope from the synced AIP Scope Catalog. Transaction, communication, filesystem, and agent-spawning scopes alone do not grant authority to submit Approval Envelopes.

Rule DESTRUCTIVE-1 (Mandatory Approval): Any operation involving a scope whose active AIP Scope Catalog entry has destructive: true 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.

The optional Endorsement Object notes field is signed audit context only. It MUST NOT be used as an input to reputation scoring, endorsement weighting, validation, or counter updates.

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. This is a reputation-input recommendation, not a protocol validation gate. Registries and Relying Parties cannot prove that an orchestrator failed to submit an expected Endorsement Object, and absence of such an endorsement MUST NOT cause Approval Envelope validation, Credential Token validation, or step-completion processing to fail.

14.1.1. Endorsement Acceptance Requirements

A conformant Registry that accepts endorsements, by implementing POST /v1/endorsements, MUST enforce the following minimum acceptance rules regardless of its reputation scoring algorithm:

1 The endorsement signature MUST be verified using signature_kid. The DID portion of signature_kid MUST equal from_aid, and the key MUST be Ed25519 verification material controlled by that AID. If key binding or signature verification fails, the Registry MUST reject with endorsement_invalid.

2 The from_aid MUST be a registered, non-revoked agent AID in the Registry. If not, the Registry MUST reject with unknown_aid or agent_revoked.

3 The to_aid MUST be a registered agent AID in the Registry. If not, the Registry MUST reject with unknown_aid.

4 The score field MUST be in the range 1 through 5 inclusive, as an integer or decimal value. Values outside this range MUST be rejected with endorsement_invalid.

14.2. Reputation Scoring

Reputation scoring algorithms, aggregation functions, and score decay policies are implementation-defined and are not subject to conformance testing. A Registry that implements the Reputation Output optional feature MUST expose a reputation score in the range 0.0 through 5.0 at GET /v1/agents/{aid}/reputation in a response containing at minimum:

* score: number, current reputation score in the range 0.0 through 5.0;

* endorsement_count: integer, total number of accepted endorsements;

* computed_at: string, ISO 8601 UTC timestamp of last score computation.

A Registry that implements this optional feature 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 aggregate reputation counters are single-resource fields and are not paginated. The revocation_history member is a collection and MUST use the cursor pagination rules in Section 17.20. A response MUST include no more than the effective limit revocation history entries and MUST include a pagination object for continuing the history scan. Absence of older history on the current page MUST NOT be interpreted as absence of older revocations when pagination.has_more is true.

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, unless the AID or token is otherwise revoked. Registries MUST retain retired public verification keys for at least the maximum Credential Token TTL plus clock skew after the key’s valid_until time, so that conforming unexpired tokens remain verifiable. Relying Parties MUST perform the expiration preflight in Section 9 before historical key lookup so expired tokens return token_expired even after retired key material is no longer retained.

Agents whose synced AIP Namespace Catalog entry has requires_task_id: true MUST have a non-null task_id. Namespace-specific lifecycle rules, including task-completion revocation expectations, are defined by the AIP Namespace Catalog. For the Draft-02 catalog, ephemeral

agents MUST be explicitly revoked on task completion. The Registry MUST also persist a lifecycle_expires_at deadline for each ephemeral agent at registration time. This deadline MUST be no later than the earlier of the decoded root Principal Token expires_at value and the initial Capability Manifest expires_at value. The Registry MUST automatically issue or materialise a full_revoke Revocation Object for the ephemeral AID with reason lifecycle_expired no later than 15 seconds after lifecycle_expires_at passes if the AID has not already been revoked. This auto-revocation requirement is a Registry conformance requirement; Relying Parties MUST still perform normal Principal Token, Capability Manifest, and revocation validation and MUST NOT rely on auto-revocation as the only expiry check.

16. Principal Chain

The principal chain is the sequence of delegation relationships that connects a root human or organisational principal, identified by a W3C DID, to a leaf agent, identified by a did:aip AID. Each link in the chain is represented by a Principal Token (Section 5.5). The chain is embedded in Credential Tokens and Step Execution Tokens as the aip_chain array.

Normative rules governing the principal chain are defined as follows:

* Chain structure and depth rules: Section 10.1.

* Capability scope inheritance across the chain: Section 10.2.

* Chain validation algorithm: Section 9.11 Steps 8a through 8l and Post-Checks A, B, and C.

* Grant ceremony that establishes the first chain link: Section 12.

* A2A identity chaining for multi-agent chains: Section 10.4.

This section serves as the normative index for the principal chain concept. Implementations MUST consult the sections above for all chain-related requirements.

17. Registry Interface

A conformant AIP Registry MUST implement the unconditional HTTP endpoints in this section and MUST implement each feature-conditional endpoint set for any feature, grant tier, or publication mode it advertises or enables.

17.1. Endpoint Inventory

A conformant AIP Registry MUST implement the following unconditional HTTP endpoints:

Method	Path	Description
POST	/v1/agents	Register a new AID (Registration Envelope)
GET	/v1/agents/{aid}	Retrieve Agent Registration Metadata 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
POST	/v1/agents/{aid}/heartbeat	Submit signed agent heartbeat
GET	/v1/agents/{aid}/revocation	Revocation status
POST	/v1/revocations	Submit RevocationObject
GET	/v1/crl	Origin AIP CRL document; preferred CRL retrieval URI is published as endpoints.crl
PUT	/v1/agents/{aid}/overlays	Submit Capability Overlay

GET	/v1/agents/{aid}/overlays	Retrieve current overlay
GET	/v1/scopes	Synced AIP Scope Catalog metadata
GET	/v1/namespaces	Synced AIP Namespace Catalog metadata
GET	/v1/registry-metadata	Retrieve Registry Metadata
GET	/v1/registry-trust/current	Retrieve current Registry Trust Record
GET	/v1/registry-trust/{version}	Retrieve immutable Registry Trust Record by version
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 Approval Envelope
POST	/v1/approvals/{id}/reject	Principal rejects Approval Envelope
POST	/v1/approvals/{id}/steps/{n}/claim	Claim step for execution
POST	/v1/approvals/{id}/steps/{n}/complete	Mark step completed
POST	/v1/approvals/{id}/steps/{n}/fail	Mark step failed
GET	/v1/approvals/{id}/steps/{n}	Retrieve step status for SET verification
POST	/v1/approvals/{id}/compensation-	Claim compensation

	steps/{n}/claim	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	
GET	/v1/approvals/{id}/compensation-steps/{n}	Retrieve compensation step status for SET verification	

Table 29: Unconditional Registry Endpoints

The following endpoint sets are feature-conditional. A Registry MUST implement every endpoint in the applicable set when it advertises or enables the corresponding feature. A Registry MUST NOT advertise a feature in /v1/registry-metadata, the Registry Trust Record, or other discovery metadata unless it implements the complete endpoint set for that feature.

Segmented CRL publication GET /v1/crl/segments/{segment_id} retrieves a signed CRL segment.

grant_tiers_supported includes G1 POST /v1/grants submits a G1 grant, and GET /v1/grants/{grant_id} retrieves G1 grant status.

grant_tiers_supported includes G3 POST /v1/oauth/authorize is the G3 authorization endpoint.

G3 or token exchange supported POST /v1/oauth/token is the G3 token endpoint and token exchange endpoint, and GET /.well-known/oauth-authorization-server returns AS metadata per [RFC8414]. Registries that support token exchange MUST also implement POST /v1/resources, GET /v1/resources/{resource_id}, PUT /v1/resources/{resource_id}, and DELETE /v1/resources/{resource_id} as defined in Section 17.13.

Engagement Objects supported POST /v1/engagements creates an Engagement Object, GET /v1/engagements/{id} retrieves it, POST /v1/engagements/{id}/countersign countersigns a proposed Engagement Object, and PUT /v1/engagements/{id} appends an Engagement change-log update.

RPNP supported POST /v1/subscriptions creates an RPNP subscription,

GET /v1/subscriptions/{id} retrieves subscription status, and DELETE /v1/subscriptions/{id} cancels the subscription.

Reputation Output supported GET /v1/agents/{aid}/reputation returns advisory reputation data with the response invariants in Section 14.2. A Registry that does not implement this feature MUST return HTTP 501 Not Implemented for this endpoint.

Endorsement Acceptance supported POST /v1/endorsements submits an Endorsement Object and applies the acceptance rules in Section 14.1.1. A Registry that does not implement this feature MUST return HTTP 501 Not Implemented for this endpoint.

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, except DID Document responses negotiated by Accept: application/did+json or Accept: application/did+ld+json from GET /v1/agents/{aid}, and CRL responses from GET /v1/crl or the signed endpoints.crl URI, which MUST use Content-Type: application/aip-crl+json.

All Registry responses from AIP protocol endpoints MUST include X-AIP-Version as defined in Section 8.1. If a request version is unsupported, the Registry MUST reject the request with unsupported_version and include X-AIP-Version in the error response.

Unless an endpoint profile explicitly requires OAuth-native error syntax, every Registry error response from an AIP protocol endpoint MUST conform to error-response.schema.json and the error response rules in Section 18.1.

All timestamps MUST be in ISO 8601 UTC format.

17.4. Agent Registration Metadata

Unless the client sends Accept: application/did+json or Accept: application/did+ld+json, GET /v1/agents/{aid} MUST return Content-Type: application/json with an Agent Registration Metadata response conforming to agent-registration.schema.json. If the AID is not registered, the Registry MUST return unknown_aid.

The response MUST include the requested aid, the current Agent Identity Object as identity, the grant_tier accepted during Registration Envelope validation, registered_at, updated_at, and links for the current public key, current Capability Manifest, live revocation status, and the registration_warnings array. The identity field MUST be the current stored Agent Identity Object at the time of the most recent successful registration or key rotation, and identity.aid MUST equal the top-level aid. The registration_warnings array MAY be empty. If Registration Check 15 accepted a Tier 2 registration without identity.model.attestation_hash, the array MUST include the persisted model_attestation_missing_tier2 warning until the registered Agent Identity Object includes a valid attestation hash.

A successful POST /v1/agents response MUST use HTTP 201 and MUST return the same Agent Registration Metadata representation that a subsequent GET /v1/agents/{aid} request would return. Any registration warning created during validation MUST therefore be visible in the registration response and in later metadata reads.

Relying Parties use the registered grant_tier value for each participating aip_chain AID during Step 9d Grant Tier Conformance validation.

17.5. Agent Key Rotation Endpoint

PUT /v1/agents/{aid} is used only for key rotation of an existing AID. The request body MUST be a complete Agent Identity Object, not a Registration Envelope. The request MUST use Content-Type: application/json and MUST include a DPoP proof for the exact PUT request URI. For an ordinary rotation, the DPoP proof MUST verify against the current active public key for the AID before the rotation is committed. If the path {aid} is not already registered, the Registry MUST reject with unknown_aid.

The DPoP proof for key rotation MUST include htu, htm, iat, and jti. The Registry MUST reject a missing proof with dpop_proof_required and MUST reject an invalid, stale, or replayed proof with invalid_token. The Registry MUST store enough recent DPoP jti values for each AID to reject replay within the accepted DPoP clock-skew window.

The Registry MUST reject the request with invalid_request unless all of the following conditions hold:

1. The percent-decoded path {aid} equals identity.aid.

2. The submitted identity.version equals the currently stored identity version plus exactly 1.

3. The submitted identity.public_key is an Ed25519 JWK conforming to the Agent Identity schema, and its kid equals identity.aid + "#key-" + identity.version.

4. The submitted object changes only version, public_key, and previous_key_signature; all other Agent Identity fields MUST be byte-for-byte identical to the currently stored Agent Identity Object.

5. previous_key_signature is present, is a non-empty base64url string, and verifies using the immediately previous stored public key over the complete submitted Agent Identity Object serialized per Section 2.1 with previous_key_signature temporarily set to "".

The version check and the identity update MUST be performed in one atomic compare-and-commit operation against the currently stored identity version. If another rotation commits first, or if the submitted identity.version is not exactly the currently stored version plus 1, the Registry MUST reject with identity_version_conflict (HTTP 409). A Registry MUST NOT accept a rotation that skips a version number or overwrites a newer stored identity.

Exact retries are idempotent. If the submitted Agent Identity Object is byte-for-byte identical to the current stored Agent Identity Object, the Registry MUST return HTTP 200 with the current Agent Registration Metadata and MUST NOT update timestamps, key states, or historical-key records. For this exact-retry case, the DPoP proof MAY verify against the current active key named by the submitted identity. A different request body for the same version is not an idempotent retry and MUST be rejected with identity_version_conflict.

On first successful commit, the Registry MUST atomically store the submitted Agent Identity Object as the current identity, mark the previous key as retired, retain the previous public verification key for historical validation according to Section 19.7, and update the registration metadata updated_at timestamp. The successful response MUST be HTTP 200 with Content-Type: application/json and a body equal to the Agent Registration Metadata representation that a subsequent GET /v1/agents/{aid} would return. The Registry MUST NOT process key rotation through POST /v1/agents.

17.6. Agent Public-Key Endpoints

A conformant AIP Registry MUST implement GET /v1/agents/{aid}/public-key and GET /v1/agents/{aid}/public-key/{key-id}. These endpoints provide the key material and validity interval required by Section 9 Step 3 and Section 9 Step 8d-2.

A successful response MUST use HTTP 200 with Content-Type: application/json and a body conforming to public-key-response.schema.json. The response MUST include aid, key_id, kid, jwk, valid_from, valid_until, and status. The kid value MUST equal {aid} + "#" + key_id, and jwk.kid MUST equal kid. The jwk member MUST contain Ed25519 public verification material.

GET /v1/agents/{aid}/public-key returns the current active verification key for {aid}. For the active key, status MUST be "active". The valid_until member MAY be null to indicate that the key has not yet been retired.

GET /v1/agents/{aid}/public-key/{key-id} returns the retained key version identified by {key-id}, where {key-id} is the JOSE kid fragment without the leading #. Historical responses MUST set status to "retired" when a later key version is active. The Registry MUST retain historical keys according to Section 19.4.2.

If {aid} is not registered, {key-id} is malformed, the requested key version is not found, or the requested key version is older than the Registry’s conformant retention window, the Registry MUST reject with unknown_aid (HTTP 404). Revocation, suspension, or deactivation of an AID MUST NOT suppress public-key lookup for retained keys; Relying Parties need those keys to verify signatures before applying revocation and lifecycle checks.

17.7. Agent Capability Manifest Endpoint

A conformant AIP Registry MUST implement GET /v1/agents/{aid}/capabilities. A successful response MUST use HTTP 200 with Content-Type: application/json and a body conforming to capability-manifest.schema.json. The returned manifest MUST be the current Registry-stored Capability Manifest for {aid}, and manifest.aid MUST equal the percent-decoded path {aid}.

If {aid} is not registered, the Registry MUST reject with unknown_aid (HTTP 404). If the Registry cannot return a stored manifest for a registered AID, or if the stored manifest is malformed, has an invalid signature, fails catalog constraint validation, or is not bound to {aid}, the Registry MUST reject with manifest_invalid. The Registry MUST return the current manifest even if expires_at has passed; Relying Parties then reject with manifest_expired under Section 9 Step 9.

A conformant AIP Registry MUST also implement PUT /v1/agents/{aid}/capabilities to replace the current Capability Manifest. The request body MUST conform to capability-manifest.schema.json, MUST have aid equal to the percent-decoded path {aid}, MUST use a version value exactly one greater than the Registry's current stored manifest version for that AID, and MUST have a valid signature from granted_by. On success, the Registry MUST store the submitted manifest as the new current manifest and return HTTP 200 with the stored manifest body. Unknown AIDs return unknown_aid; malformed manifests, invalid signatures, version conflicts, catalog constraint failures, and AID mismatches return manifest_invalid; submitted manifests whose expires_at has already passed return manifest_expired.

17.8. Agent Revocation Status Endpoint

A conformant AIP Registry MUST implement GET /v1/agents/{aid}/revocation. This endpoint returns the authoritative live revocation status used by Section 9 Step 7 for Tier 2 validation.

If the path {aid} does not identify a registered AID in the authoritative Registry, the Registry MUST reject with unknown_aid (HTTP 404). If the AID is registered, the Registry MUST return HTTP 200 with a JSON body conforming to revocation-status.schema.json, including when the AID has no active revocations.

The response fields are:

aid The registered AID being checked. This value MUST equal the

percent-decoded path {aid}.

checked_at ISO 8601 UTC timestamp at which the authoritative Registry evaluated the revocation state.

status "active" when no active revocation applies, "restricted" when only scope_revoke or delegation_revoke applies, and "revoked" when full_revoke applies to the AID, when principal_revoke targets the AID, or when principal_revoke targets the root Principal DID associated with the AID's current registration authority.

revoked Boolean. MUST be true when full_revoke or principal_revoke applies directly to the AID or through the AID's root Principal DID. Relying Parties MUST reject Credential Tokens for the AID when this value is true.

delegation_revoked Boolean. MUST be true when an active delegation_revoke applies. Relying Parties MUST reject delegation-chain use rooted at this AID when this value is true.

scopes_revoked Array containing the union of active scope_revoke scopes for the AID. The array is empty when no active scope revocation applies.

active_revocations Array of the signed Revocation Objects that currently affect this AID. The array is empty when status is "active". When a principal-wide principal_revoke affects the AID, this array MUST include the signed principal_revoke object whose target_id is the Principal DID.

A registered, non-revoked AID therefore returns HTTP 200 with status: "active", revoked: false, delegation_revoked: false, an empty scopes_revoked array, and an empty active_revocations array.

17.9. Revocation Submission Endpoint

A conformant AIP Registry MUST implement POST /v1/revocations. The request body MUST be a Revocation Object conforming to revocation-object.schema.json. Request validation, issuer authorization, idempotency, response behavior, CRL update effects, and child-propagation processing are defined by Section 11.2.

17.10. CRL Response

A conformant AIP Registry MUST implement GET /v1/crl as the canonical origin endpoint for the AIP Certificate Revocation List. The endpoint returns the signed CRL document defined in Section 11.3 and conforming to crl.schema.json. Registries MAY publish an absolute CDN or distributed-object URI in the signed Registry Trust Record's signed.endpoints.crl; the origin endpoint and any published distribution URI MUST return the same current CRL payload or a byte-for-byte older payload that remains valid before its next_update.

The CRL endpoint is not a generic paginated collection endpoint. Cursor pagination MUST NOT be applied to the revocations array of a signed CRL document because doing so would change the object covered by the CRL signature and could cause clients to validate against an incomplete revocation set.

A Registry whose active CRL is too large to publish as a single efficient document MAY publish a segmented CRL. In that mode, GET /v1/crl returns a signed CRL index document whose signed.publication_mode is "index", whose signed.revocations array is empty, and whose signed.segments array lists signed CRL segment documents. Each segment URI, including /v1/crl/segments/{segment_id} when used by the origin Registry, MUST return Content-Type: application/aip-crl+json and a signed CRL document whose signed.publication_mode is "segment".

The signed CRL index MUST identify the partitioning scheme with partition_key_alg: "sha-256-target-id-hex". Segment ranges are inclusive ranges over the lowercase hexadecimal SHA-256 digest of the Revocation Object target_id string. Segment ranges in one CRL index MUST be non-overlapping and together cover every active Revocation Object in the indexed CRL. A Relying Party using a segmented CRL for bounded-staleness validation MUST retrieve and verify every segment whose range contains the partition key for any target identifier it must check, including the token's root Principal DID and every AID in the validated aip_chain.

For each referenced segment, the Relying Party MUST verify the segment CRL signature against the Registry Trust Record identified by the segment's signed.trust_record_version, verify that the segment's registry_id, crl_id, sequence, issued_at, and next_update match the signed index, and verify that the JCS canonicalized segment document hashes to the segment reference's sha256 value. If any required segment is unavailable or fails verification, bounded-staleness validation MUST fail with registry_unavailable.

17.11. Agent Heartbeat Endpoint

A conformant AIP Registry MUST implement POST /v1/agents/{aid}/heartbeat. This endpoint records liveness for the AID identified by the path parameter and is the submission mechanism used by the optional Dead Man's Switch in Section 21.8.

The request MUST include Authorization: DPoP <token>, where <token> is a Credential Token issued by the same AID as the path parameter. The Registry MUST validate the Credential Token using the Section 9 validation algorithm with the Registry as the Relying Party. The token aud claim MUST identify either the Registry's registry_id or the heartbeat endpoint URI. The token aip_scope MUST include registry.heartbeat.

The request MUST include a DPoP proof bound to the HTTP method and heartbeat URI. If the DPoP proof is absent, reject with dpop_proof_required. If the proof is malformed or invalid, reject with invalid_token.

After Credential Token validation, the Registry MUST verify that the path {aid} equals both the Credential Token issuer (iss) and the leaf Principal Token subject (aip_chain[n-1].sub). If either value differs from the path AID, reject with invalid_token. If the token does not contain registry.heartbeat, or if the current Capability Manifest does not grant registry.heartbeat, reject with insufficient_scope.

On success, the Registry MUST record last_heartbeat_at using the Registry's receipt time. Client-supplied body timestamps, if any, MUST NOT be used to advance liveness. A successful heartbeat MAY also update the agent's last_active reputation metadata.

17.12. 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 (Principal 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
GET	/v1/approvals/{id}/compensation-steps/{n}	Get compensation step status (used by Relying Parties for Step Execution Token verification)

Table 30

POST /v1/approvals validation: The Registry MUST validate the request body as an ApprovalEnvelopeSubmission, reject any read-only stored-resource fields in that body, and perform all checks defined in Section 13.10 before accepting an Approval Envelope. The Registry MUST return HTTP 201 with the StoredApprovalEnvelope, including the Registry-assigned status: "pending_approval" and initial step statuses, 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 Principal Wallet over the immutable approval signing input defined in Section 13.3. That signing input is the accepted ApprovalEnvelopeSubmission content plus creator_signature and principal_signature: ""; it excludes Registry-managed status, approved_at, and step status/timestamp fields. The signing key MUST be a verification method controlled by principal_id. The Registry MUST verify this signature against the resolved principal_id DID Document before transitioning the envelope to approved.

Atomicity requirement for approval: The Registry MUST implement POST /v1/approvals/{id}/approve and POST /v1/approvals/{id}/reject as atomic lifecycle transitions. Approval MUST be a conditional update from pending_approval to approved and MUST succeed only when the request is committed before approval_window_expires_at. Rejection MUST be a conditional update from pending_approval to rejected. Only one concurrent approval or rejection request for the same Approval Envelope can succeed. If another Principal Wallet, expiry job, cancellation cascade, approval, or rejection has already changed the envelope state, the Registry MUST reject the losing request with approval_state_conflict and MUST NOT overwrite the stored principal_signature, approved_at, or terminal state. If the envelope is still in pending_approval and approval_window_expires_at has passed, the Registry MUST reject an approval request with approval_envelope_expired and transition or leave the envelope in expired. A byte-for-byte retry of the winning approval request MAY return the stored approved envelope as an idempotent success.

Atomicity requirement for step claim: The Registry MUST implement step-claim operations atomically, for example 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 endpoint request and response bodies: Forward-step and compensation-step claim, complete, and fail request and response bodies MUST conform to the profiles in approval-step-endpoints.schema.json. Complete and fail requests MUST present the compact Step Execution Token issued for the active claim. The Registry MUST validate the SET, match it to the request path and stored claim jti, enforce the claim deadline, and process exact idempotent retries as defined in Section 13.8.

Step Execution Token format: The Step Execution Token returned by POST /v1/approvals/{approval_id}/steps/{step_id}/claim or POST /v1/approvals/{approval_id}/compensation/{compensation_step_id}/claim is a Registry-issued JWT with JOSE typ: "AIP-SET+JWT", alg: "EdDSA", and a kid matching a JWK in active_verification_keys.step_execution of the Registry Trust Record version current at issuance time. The protected JOSE header MUST also include aip_trv equal to that Registry Trust Record version. Its payload claims are described in Section 13.8. Its TTL MUST comply with the Step Execution Token TTL derivation rule in Section 5.4.1.

17.13. Resource Registration

AIP Registries that support token exchange (Section 8.4) MUST maintain a resource registry mapping target resource URIs to their authorization configuration. Each Registered Resource Record MUST conform to resource-record.schema.json and MUST include:

resource_uri string, REQUIRED. The RFC 9728 resource identifier.

authorization_server string, REQUIRED. The URI of the authorization server for this resource, either the AIP Registry itself or an external authorization server.

enterprise_idp_required boolean, OPTIONAL. When true, token exchange requests targeting this resource MUST use the Enterprise IdP Federation Profile (Section 8.4.5). Default: false.

enterprise_idp_token_endpoint string, CONDITIONAL. REQUIRED when enterprise_idp_required is true. The Enterprise IdP’s token endpoint URI.

enterprise_idp_client_id string, CONDITIONAL. REQUIRED when enterprise_idp_required is true. The Registry’s registered client_id with the Enterprise IdP.

Resource records are addressed by resource_id, the lowercase hexadecimal SHA-256 digest of the UTF-8 resource_uri value. The Registry MUST reject a create request whose path-derived or body-supplied resource_id does not equal that digest with registration_invalid. The Registry MUST reject duplicate resource_uri create requests with HTTP 409 unless the submitted record is byte-for-byte identical to the stored record, in which case it MAY return the existing record.

A Registry MUST NOT accept unauthenticated public writes to the resource registry. The registering caller MUST be authenticated as a Registry operator, the resource owner, or another party authorized by

local Registry policy. The Registry MUST validate that resource_uri, authorization_server, and any enterprise_idp_token_endpoint are absolute HTTPS URIs unless a local non-production policy explicitly allows another URI scheme.

When enterprise_idp_required is true, the Registry MUST reject the record unless enterprise_idp_federation_supported is true in Registry discovery metadata and both enterprise_idp_token_endpoint and enterprise_idp_client_id are present. Token exchange requests whose resource parameter resolves to such a record MUST follow the Enterprise IdP Federation Profile in Section 8.4.5. Token exchange requests for an unknown resource value MUST be rejected with invalid_target.

The resource registration endpoints have the following semantics: POST /v1/resources creates a Registered Resource Record, GET /v1/resources/{resource_id} retrieves one record, PUT /v1/resources/{resource_id} replaces one record atomically, and DELETE /v1/resources/{resource_id} removes the record from future token-exchange matching. Delete does not revoke already issued enterprise IdP access tokens; those tokens remain governed by the enterprise IdP's own lifetime and revocation mechanisms.

17.14. OAuth 2.1 Authorization Server

A conformant AIP Registry supporting G3 grants MUST implement an OAuth 2.1 Authorization Server [I-D.ietf-oauth-v2-1] 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. For G3 grant ceremonies, the authorization request MUST include aip_grant_request containing the compact-JWS-signed GrantRequest defined in Section 12.2, and the Authorization Server MUST enforce the G3 binding checks in Section 12.10 before issuing an authorization code.

4. The scope parameter MUST use AIP scope strings from the synced AIP Scope Catalog or accepted local extension policy.

5. The token endpoint MUST redeem an authorization code only against the immutable GrantRequest binding stored during authorization. It MUST reject unbound, expired, replayed, or binding-mismatched authorization codes with grant_request_invalid.

6. For G3 grant ceremonies, the token endpoint response MUST follow the G3 token endpoint response mapping in Section 12.10. The access_token value is a transport alias for the compact Principal Token and MUST be byte-for-byte identical to grant_response.principal_token.

7. The Principal Token payload returned by the token endpoint MUST include acr and amr claims reflecting the authentication performed.

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

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

If an authorization request is missing code_challenge, is missing code_challenge_method, or has a code_challenge_method value other than S256, the Authorization Server MUST reject the request with pkce_required.

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

17.14.1. Registry Metadata Additions

The /v1/registry-metadata response MUST also include:

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.

acr_equivalence_map (object)
: Optional JSON object whose keys are ACR value strings received in aip_chain[0] and whose values are arrays of ACR value strings that the key maps to. A key ACR value satisfies a requirement for any ACR value in its array. Registries MUST only declare downward equivalences, where received assurance is equal to or higher than required assurance under a documented ACR equivalence policy. If absent, exact ACR matching is required with no exceptions.

amr_values_supported (array)
: Supported amr values.

oauth_authorization_server (string)
: URI of the OAuth AS metadata document; present only if G3 is supported.

endpoints.resources (string)
: Resource registration collection endpoint. REQUIRED when token exchange is supported.

identity_proofing_required_for_tier2 (boolean)
: Whether this Registry requires G3 for Tier 2 operations. When true, enforced during Registration Check 14c, Validation Step 8 Post-Check C, and Validation Step 9d.

mtls_client_certificate_profile (string)
: Client-certificate binding profile for Tier 3 operations. When Tier 3 is accepted, this value MUST be "aip-san-uri-v1".

mtls_trust_anchors_uri (string)
: HTTPS URI documenting the Tier 3 client-certificate trust anchor set and revocation-checking mechanism. REQUIRED when Tier 3 is accepted.

enterprise_idp_federation_supported (boolean)
: Whether the Registry supports the Enterprise IdP Federation Profile in Section 8.4.5. SHOULD be present in all Registry Metadata documents. Absence is treated as equivalent to false. Gateways that surface this capability through /.well-known/oauth-protected-resource SHOULD ensure the value is consistent with the Registry’s declared support in this document. A mismatch between the two documents MUST be treated as a Registry configuration error.

enterprise_idp_token_exchange_grant_types_supported (array)
: Enterprise IdP grant types the Registry can use for secondary exchange. Values SHOULD include RFC 7523 JWT bearer and/or RFC 8693 token exchange when federation is supported.

enterprise_idp_issuers_supported (array)
: OIDC issuer identifiers or issuer patterns accepted for enterprise principal assertions and secondary token exchange, subject to local trust policy.

aip_catalog_uri (string)
: URI of the synced AIP Catalog bundle or mirror.

aip_catalog_version (string)
: Draft-aligned catalog version, e.g. draft-02.

aip_catalog_snapshot_id (string)
: Immutable Catalog Snapshot identifier.

aip_catalog_sha256 (string)
: Lowercase SHA-256 digest of the exact catalog bundle bytes, formatted as sha256:<64hex>.

step_claim_timeout_seconds (integer)
: Registry claim timeout for Approval Envelope steps. MUST be 1-600 seconds inclusive.

17.15. AIP Catalog Sync, Scope Map, and Namespace Map

Concrete AIP scope, scope-family, and namespace registrations are defined by an immutable AIP Catalog Snapshot. This specification defines the wire grammar, Registry sync requirements, integrity metadata, validation semantics, and minimum Draft-02 catalog content. The catalog defines the concrete entries and their metadata for each draft snapshot.

The canonical list of built-in AIP scopes, scope families, and namespaces is the set of entries with class: "standard" in the applicable Catalog Snapshot. Entries with class: "standard" in that snapshot are the only built-in AIP registrations for that draft line. A Registry, deployer, or Relying Party MUST NOT mint a new standard AIP scope, scope family, or namespace by local policy alone; interoperable standard entries MUST appear in an applicable AIP Catalog Snapshot.

The Draft-02 Catalog Snapshot is identified by catalog_version: "draft-02" and catalog_snapshot_id: "https://provai.dev/aip/catalog/draft-02". The authoritative Draft-02 Catalog Snapshot is the deterministic dist/catalog.json release artifact from the separate provai-dev/aip-catalog repository for the immutable Draft-02 release. A conformant Registry for this draft MUST publish, in /v1/registry-metadata, an HTTPS aip_catalog_uri from which the exact JSON Catalog Bundle can be retrieved and an aip_catalog_sha256 value containing the lowercase SHA-256 digest of the exact UTF-8 bytes retrieved from that URI. A mirror is conformant only when it serves bytes whose SHA-256 digest equals the published aip_catalog_sha256 value and whose JSON content carries catalog_version: "draft-02", aip_draft: "draft-02", and catalog_source_uri: "https://github.com/provai-dev/aip-catalog". The Registry metadata field aip_catalog_snapshot_id identifies the pinned snapshot; it is not required to be duplicated inside the catalog JSON bundle.

Draft-02 conformance MUST be based on the pinned Catalog Snapshot identified above. A Registry MUST NOT use a latest, latest-live, branch head, mutable tag, or other moving catalog input to determine standard Draft-02 scope, scope-family, namespace, or security metadata. Changes to any standard entry, including changes to tier, destructive, requires_dpop, ttl_max_seconds, grant_tier_min, constraint_schema, status, or class, require a new catalog_snapshot_id, a new aip_catalog_sha256, and a corresponding AIP draft or protocol version update before they affect Draft-02 conformance.

The Draft-02 JSON Catalog Bundle MUST be generated by the validation and distribution tooling in the aip-catalog repository. The bundle MUST contain top-level catalog_name, catalog_version, aip_draft, catalog_source_uri, scopes, scope_families, and namespaces members. The namespaces array MUST include standard entries for personal, enterprise, service, orchestrator, and ephemeral, and a reserved entry for registry. The scope_families array MUST include standard entries for transactions.* and communicate.*. The scopes array MUST include standard entries for transactions, web.forms_submit, email.send, web.download, spawn_agents.create, spawn_agents.manage, and approvals.create. A missing required standard entry, missing required security or provenance metadata field, stale generated bundle, or digest mismatch invalidates the Registry’s Draft-02 catalog conformance.

For standard AIP scope entries in the Draft-02 catalog, every active or experimental scope with destructive: true MUST also have requires_dpop: true. The Draft-02 catalog additionally marks non-destructive but high-risk external-effect scopes email.send, web.download, and web.forms_submit with requires_dpop: true. A Registry mirror MUST NOT weaken these requires_dpop values for standard scopes. The requires_dpop flag is additive: Tier 2 and Tier 3 operations require DPoP under Section 9 Step 10 regardless of this flag, while Tier 1 scopes and endpoints can use the flag or endpoint policy to require DPoP.

Deployers and Registry operators MAY expose non-standard scopes only as explicit extensions. Extension entries intended for cross-Registry interoperability MUST be registered in the AIP Catalog. Until accepted in an applicable catalog snapshot, such entries are local extensions. A deployer MUST NOT create a valid scope merely by including an unknown string in a Capability Manifest; the Registry MUST first accept and publish the extension entry in its Scope Map. Local or community extension scope identifiers MUST use the x. prefix followed by a namespace label controlled by the Registry or deployer, for example x.example.payments.release. The namespace label MUST either be an active namespace entry in the Registry’s Namespace Map

or a local label documented by the Registry's extension policy, and that policy MUST state how control of the label is established. The namespace label and all following segments MUST satisfy the scope string grammar below. Local extensions MUST NOT shadow, override, or weaken any standard catalog scope, scope family, namespace, or reserved prefix.

Extension scopes are not Tier 1 by default. Every exposed scope entry, including local and community extensions, MUST carry explicit values for tier, destructive, requires_dpop, ttl_max_seconds, grant_tier_min, constraint_schema, status, class, owner, source, and change_type. If a Registry cannot determine any required security or provenance metadata for an extension scope, it MUST NOT expose that scope as valid. Registration, grant, or validation attempts that depend on such an incomplete scope MUST be rejected with registration_invalid at registration time, grant_request_invalid during grant consent, or invalid_scope at token validation time.

The standard web.forms_submit scope is classified as Tier 2 in the Draft-02 catalog, with grant_tier_min: "G2" and ttl_max_seconds: 300. It does not inherit the generic web.* Tier 1 default because form submission can create external side effects including account changes, data submissions, and financial form submissions.

AIP does not define a URN namespace for scope identifiers. The canonical on-wire AIP scope value is the dot-notation scope string:

```
scope_string = dot-separated-scope-name
```

Scope strings in the Scope Map MUST match <dot-separated-scope-name>. This allows digits within each dot-separated segment after the first character. The same short string form MUST be used in OAuth scope parameters and within AIP Credential Tokens (aip_scope array). Catalog entries also include a uri member for documentation and catalog linking. Standard Draft-02 Catalog entries MUST use HTTPS URIs under https://provai.dev/aip/scopes/; these HTTPS URIs are not the on-wire OAuth scope values.

A conformant AIP Registry MUST implement GET /v1/scopes returning a JSON object with catalog sync metadata and an array of scope entries. The response object MUST include aip_draft, catalog_version, catalog_snapshot_id, catalog_source_uri, catalog_sha256, synced_at, scopes, and pagination. It MUST also include extension_policy_uri. The extension_policy_uri value MUST be non-null when the response includes any community scope entry or any locally documented extension scope and MAY be null otherwise. Pagination MUST follow Section 17.20 and cursors MUST be bound to the advertised catalog_version. Each scope entry MUST include id, uri, family,

description, tier, destructive, requires_dpop, ttl_max_seconds, grant_tier_min, constraint_schema, status, class, owner, introduced_in, updated_in, change_type, and source. The class value MUST be one of standard, community, or reserved.

Principal Wallet consent UIs MUST use the standardized display titles in Section 12.3 for standard Draft-02 scopes. For community or local extension scopes, the Registry's extension policy MUST provide a non-empty human-readable title and summary for each exposed extension scope before that scope can be used in a grant ceremony. The display title or summary MUST NOT broaden the authorization represented by the scope metadata.

When constraint_schema is non-null, it MUST be a JSON Schema Draft 2020-12 fragment associated with the containing catalog_version. Any change to a non-null constraint_schema MUST update the scope entry's updated_in value and MUST be reviewed as a catalog change.

A conformant AIP Registry MUST implement GET /v1/namespaces returning a JSON object with catalog sync metadata and an array of namespace entries. The response object MUST include aip_draft, catalog_version, catalog_snapshot_id, catalog_source_uri, catalog_sha256, synced_at, and namespaces. It MUST also include pagination. It MUST include extension_policy_uri when the response includes any community namespace entry or any locally documented extension namespace and MAY use null otherwise. Pagination MUST follow Section 17.20 and cursors MUST be bound to the advertised catalog_version. Each namespace entry MUST include id, description, reserved, spawnable, requires_task_id, lifecycle_rules, status, class, owner, introduced_in, updated_in, change_type, and source. The class value MUST be one of standard, community, or reserved.

A Registry MUST expose only catalog entries that are present in its synced catalog snapshot or in an explicitly documented local or community extension policy. A Relying Party MUST treat active entries as valid. It MAY accept experimental entries only when local policy allows experimental behavior. It MUST reject reserved, removed, unknown, or conflicting entries with invalid_scope for scopes or registration_invalid for registration-time namespace failures.

Registries MUST validate submitted Capability Manifests and Capability Overlays against each applicable non-null constraint_schema from the synced Scope Catalog. Registration submissions that fail this validation MUST be rejected with registration_invalid. Capability Overlay submissions that fail this validation MUST be rejected with overlay_exceeds_manifest. Relying Parties performing Step 9 or Step 9b validation MUST apply the same non-null constraint_schema fragments before treating manifest or overlay constraints as authoritative.

Registries MUST expose the constraint_schema fragments that correspond to their advertised catalog_version. A Registry MUST NOT silently substitute a schema fragment from a different catalog version for a standard scope. Local or community extension schema changes MUST update the affected entry's updated_in value and MUST be documented by the Registry's extension policy before the changed schema is used for registration or token validation.

Scope-family entries in the AIP Catalog define category notation such as transactions.* and communicate.*. Category notation is not a literal scope string and MUST NOT appear as a token aip_scope value unless it is separately registered as a concrete scope.

17.16. Capability Overlay Endpoints

PUT /v1/agents/{aid}/overlays submits a Capability Overlay for the AID in the path. The request body MUST conform to the Capability Overlay schema in Section 5.11, and the body aid value MUST equal the path {aid}.

The Registry MUST reject overlays whose issued_by DID uses did:key with overlay_issuer_invalid. The Registry MUST verify the overlay signature over the Section 2.1 signing input using the Ed25519 verification method identified by signature_kid. The DID portion of signature_kid MUST equal issued_by. If the signature is absent, malformed, cannot be verified, or verifies under a key not controlled by issued_by, the Registry MUST reject with overlay_signature_invalid.

For the tuple (aid, engagement_id, issued_by), a submitted overlay version MUST be strictly greater than the current active overlay version for that tuple. If the submitted version is less than or equal to the current active version, the Registry MUST reject with overlay_version_conflict. If the overlay expands the base Capability Manifest or fails catalog constraint validation, the Registry MUST reject with overlay_exceeds_manifest.

17.17. 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
POST	/v1/engagements/{id}/countersign	Countersign proposed Engagement Object
PUT	/v1/engagements/{id}	Update Engagement (append change log entry)

Table 31

Create validation: The Registry MUST authenticate the submitter as the hiring_operator or as a caller authorized by Registry policy, verify hiring_operator_signature and the initial change-log entry signatures using the Engagement signature verification profile in Section 5.12, validate that all referenced participant AIDs exist and are not revoked, and assign status: "proposed" with a first change log entry whose seq is 1 and whose action is engagement_created. The Registry MUST set version: 1 on the stored Engagement Object. A create request that requests status: "active" without a valid deploying_principal_signature MUST be rejected with engagement_countersign_required.

Countersign validation: POST /v1/engagements/{id}/countersign applies only to Engagement Objects in proposed status. The request body MUST contain deploying_principal_signature and a new change log entry with action: "engagement_countersigned". If deploying_principal_signature is absent, the Registry MUST reject the request with engagement_countersign_required. The Registry MUST verify that the authenticated submitter is the deploying_principal, verify the countersignature and the change log entry signature using the Engagement signature verification profile in Section 5.12, and verify the entry seq is exactly current_max + 1. If the entry seq is not exactly current_max + 1, the Registry MUST reject with change_log_sequence_invalid. On success, the Registry MUST append

the change log entry, store deploying_principal_signature, set status: "active", and set version to the appended entry's seq atomically.

Update validation: The Registry MUST verify the submitter is an active participant, the change log entry's seq is exactly current_max + 1, and the entry signature is valid under the Engagement signature verification profile in Section 5.12. If the seq value is missing, repeated, skipped, or otherwise not exactly current_max + 1, the Registry MUST reject with change_log_sequence_invalid. The Registry MUST reject modifications to existing entries with change_log_immutable. On success, the Registry MUST append the entry, apply the mutation represented by that entry, and set version to the appended entry's seq atomically. The Registry MUST reject any PUT request that changes top-level engagement state without appending exactly one accepted change-log entry.

17.18. RPNP Subscription Endpoints

A conformant AIP Registry supporting RPNP (Section 11.5) 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 32

Subscription creation request bodies MUST conform to rpnp.schema.json subscription_request, and successful responses MUST conform to subscription_response. Subscription creation MUST be authenticated using the subscriber_did-bound DPoP profiles defined in Section 11.5.2. The webhook_uri MUST use HTTPS. The Registry MUST reject non-HTTPS URIs with invalid_webhook_uri. The Registry MUST apply the RPNP subscription authorization, scope-filter, and quota checks defined in Section 11.5.2 before creating the subscription. On success, the Registry MUST return HTTP 201 with the assigned subscription_id.

GET /v1/subscriptions/{id} and DELETE /v1/subscriptions/{id} MUST use the same subscriber_did-bound authentication profile. The authenticated subscriber_did MUST equal the subscription owner unless

Registry administrator policy explicitly authorizes the caller to inspect or cancel subscriptions for other subscribers. Otherwise, the Registry MUST reject with subscription_auth_required. GET MUST return the current subscription_response. DELETE MUST transition the subscription to cancelled and return the updated subscription_response.

17.19. Key Management and Rotation

Registry trust evolution is defined in Section 7.4.7. 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.

17.20. Pagination and Large Responses

Registry endpoints that return collections and are not otherwise defined as signed snapshot artifacts MUST support cursor pagination. This requirement applies to collection-bearing endpoints such as GET /v1/scopes, GET /v1/namespaces, and collection members of GET /v1/agents/{aid}/reputation. It does not apply to single-resource endpoints or to signed CRL documents, which use the CRL segmentation rules above.

Paginated endpoints MUST accept limit and cursor query parameters. The limit value MUST be an integer from 1 to 1000 inclusive. If limit is absent, the default is 100. A Registry MAY apply a lower maximum for a specific endpoint, but the effective maximum MUST be at least 100. The cursor value is opaque to clients and MUST be used only with the same endpoint and query filter set that produced it. Invalid limit or cursor values MUST be rejected with invalid_request.

A paginated response MUST include the endpoint-specific collection member, such as scopes, namespaces, or revocation_history, containing no more than the effective limit. It MUST also include a pagination object with limit, next_cursor, and has_more. The next_cursor value MUST be null when has_more is false. Servers MUST use a deterministic ordering for each collection and MUST bind cursors to a stable collection snapshot. If a cursor refers to a snapshot that is no longer available, the Registry MUST reject the request with invalid_request.

18. Error Handling

18.1. Error Response Format

Unless a referenced OAuth binding normatively requires OAuth-native error syntax, every AIP protocol error response MUST use HTTP status code 4xx or 5xx, MUST use Content-Type: application/json, and MUST carry a body conforming to error-response.schema.json in the JSON Schema Index. Implementations MUST NOT return HTTP 200 for error conditions.

The error response body MUST include error, error_description, and aip_version. It MAY include error_uri, correlation_id, and details. The error value MUST be the exact registered error-code string. The error_description value is human-readable diagnostics only; clients MUST NOT parse it for protocol decisions. Machine-readable metadata MUST be carried in details using one of the detail object profiles defined below and in the schema.

Every AIP-aware error response MUST include an X-AIP-Version response header as defined in Section 8.1. The body aip_version value MUST equal the X-AIP-Version response header. For unsupported_version, the responder MUST set X-AIP-Version to a protocol version it supports for that endpoint and SHOULD include X-AIP-Supported-Versions. The details object for unsupported_version MUST include supported_versions and SHOULD include the rejected version values when they are available.

For error responses conforming to this specification, X-AIP-Version and the body aip_version value MUST be "0.3", except that a responder rejecting an unsupported version uses the supported version it selected for the error response.

EXAMPLE (informative):

```
{
  "error": "unsupported_version",
  "error_description": "Unsupported AIP version.",
  "aip_version": "0.3",
  "error_uri": "https://provai.dev/errors/unsupported_version",
  "correlation_id": "err_01HZX7M7N2M6M2T8K8F5Z3Q9Y1",
  "details": {
    "type": "unsupported_version",
    "requested_version": "0.2",
    "header_aip_version": "0.2",
    "token_aip_version": null,
    "supported_versions": ["0.3"]
  }
}
```

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.

invalid_request HTTP 400. Request syntax, query parameter, pagination cursor, or endpoint-specific request parameter is invalid.

unsupported_version HTTP 400. X-AIP-Version or token aip_version is missing, unsupported, or mutually inconsistent.

dpop_proof_required HTTP 401. DPoP proof is required for the request but the DPoP header is absent. Malformed or invalid DPoP proofs return invalid_token.

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 unavailable, signature failed, or AIP Scope Catalog constraint validation failed.

manifest_expired HTTP 403. Capability Manifest expires_at passed.

approval_envelope_invalid HTTP 400. Approval Envelope malformed, signature failed, dependency invalid, or value/hash mismatch.

approval_envelope_expired HTTP 403. The Approval Envelope was not approved before approval_window_expires_at.

approval_not_found HTTP 404. Approval Envelope ID not found.

approval_state_conflict HTTP 409. Approval Envelope state changed

before the requested approval or rejection transition could be committed.

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_not_claimed HTTP 409. Step Execution Token presented before the Registry has accepted a claim for that step.

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.

compensation_failed HTTP 409. An Approval Envelope compensation step failed and the envelope is in terminal failed state requiring out-of-band remediation.

engagement_cancelled HTTP 409. The referenced Approval Envelope or step was cancelled because its parent Engagement Object was terminated. The operation cannot proceed.

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, expired, missing from a G3 authorization request, signature failed, inconsistent with OAuth request binding, or cannot be resolved to trusted catalog scope display metadata for consent.

grant_rejected_by_principal HTTP 403. Principal declined the grant.

grant_nonce_mismatch HTTP 400. GrantResponse nonce does not match.

grant_tier_insufficient HTTP 403. Registered grant_tier is absent, unrecognised, or below the Grant Tier required for the operation’s security Tier.

registration_invalid HTTP 400. Registration Envelope malformed or failed validation, including missing or invalid grant_tier.

aid_already_registered HTTP 409. The submitted AID or public key is already associated with a registered, non-revoked AID.

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

identity_version_conflict HTTP 409. Agent Identity key rotation version is stale, skips a version, or conflicts with a committed rotation.

revocation_invalid HTTP 400. Revocation Object is malformed, uses a reserved externally submitted reason, has an invalid timestamp, or has an invalid or unverifiable signature.

revocation_unauthorized HTTP 403. Revocation issuer is not authorized for the target or revocation type.

revocation_conflict HTTP 409. Revocation identifier was previously accepted with different object content.

registry_unavailable HTTP 503. Registry or a required DID resolver could not be reached or timed out.

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

mtls_required HTTP 403. Tier 3 operation without mTLS.

invalid_scope HTTP 400. Requested scope is unknown, reserved, removed, retired (including bare spawn_agents), conflicting with the synced AIP Scope Catalog, or outside an explicitly documented local/private extension policy.

principal_did_method_forbidden HTTP 403. Principal DID method is not permitted for the operation Tier; Tier 2 and Tier 3 require did:web.

identity_proofing_insufficient HTTP 403. G3 identity proofing is absent or does not satisfy requested acr_values.

enterprise_policy_denied HTTP 403. The enterprise IdP denied the token exchange request due to Conditional Access, ABAC, or equivalent enterprise policy. The agent’s principal lacks sufficient enterprise-level authorization for the requested resource. Context: Section 8.4.5.2.

idp_client_misconfigured HTTP 500. The AIP Registry’s enterprise

IdP client credentials are invalid or expired. This is a Registry configuration error, not an agent error. Context: Section 8.4.5.2.

invalid_grant HTTP 400. The enterprise IdP rejected the secondary token exchange grant. Context: Section 8.4.5.2.

enterprise_assertion_missing HTTP 403. Tier 3 validation required an aip_principal_assertion claim, but the Credential Token did not contain one.

enterprise_assertion_invalid HTTP 403. The aip_principal_assertion JWT was malformed, could not be verified against the enterprise IdP JWKS, or used an issuer that is not accepted for the deployment.

enterprise_assertion_principal_mismatch HTTP 403. The verified aip_principal_assertion subject did not match the root Principal Token’s principal.id.

a2a_originator_invalid HTTP 400. The aip_originator_aid claim is absent in an A2A delegated context, does not conform to the did:aip ABNF, equals the acting agent’s own AID, or cannot be traced to the validated delegation chain. Context: Section 10.4.

endorsement_invalid HTTP 400. Endorsement Object validation failed, including invalid signature or out-of-range score. Context: Section 14.2.

gateway_config_invalid HTTP 400. A gateway OAuth Protected Resource Metadata document is internally inconsistent or violates AIP gateway metadata constraints. Context: Section 7.5.

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 or not using code_challenge_method S256.

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

overlay_exceeds_manifest HTTP 400. Overlay violates CO-1 attenuation rule or AIP Scope Catalog constraint validation.

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 Engagement Object countersignature.

engagement_signature_invalid HTTP 400. Engagement Object top-level signature or change-log entry signature is missing, malformed, bound to the wrong DID, uses an unsupported key, or fails verification.

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 request missing DPoP, using an invalid DPoP proof, or failing to bind subscriber_did to the DPoP key or Authorization credential.

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

The details member is optional unless this section states that it is REQUIRED for a specific error. When present, it MUST conform to one of the detail object profiles in error-response.schema.json. A detail object MUST NOT weaken, contradict, or replace the registered meaning of the top-level error value.

unsupported_version The response MUST include an unsupported_version detail object containing supported_versions. It SHOULD include requested_version, header_aip_version, and token_aip_version when those values are available.

rate_limit_exceeded The response MUST include Retry-After per [RFC6585] Section 4, the rate limit headers defined in Section 19.1, and a rate_limit detail object containing operation, limit, and reset_at.

registry_unavailable The response SHOULD include Retry-After per [RFC9110] and SHOULD include a registry_dependency detail object identifying the unavailable dependency and whether the condition is retryable.

chain_token_expired The response SHOULD include a delegation_depth detail object identifying the expired chain element's delegation_depth, and MAY include chain_index, principal_token_jti, and expires_at.

invalid_delegation_depth The response SHOULD include a delegation_depth detail object containing the submitted delegation_depth and, when applicable, max_delegation_depth.

invalid_scope and insufficient_scope The response SHOULD include a scope detail object listing the scope or scopes that caused rejection and, when catalog metadata influenced the decision, the catalog_snapshot_id and catalog_sha256 used.

manifest_invalid and overlay_exceeds_manifest When rejection is caused by scope-catalog constraint validation, the response SHOULD include a scope detail object listing the failing scopes and catalog snapshot metadata.

Conflict errors For identity_version_conflict, approval_state_conflict, revocation_conflict, overlay_version_conflict, and change_log_sequence_invalid, the response SHOULD include a conflict detail object identifying the resource and the current, submitted, or expected version, sequence, or state when available.

Approval step errors For approval_step_prerequisites_unmet, approval_step_already_claimed, approval_step_not_claimed, approval_step_action_mismatch, and approval_step_invalid, the response SHOULD include an approval_step detail object containing approval_id and step_id.

Grant request errors For grant_request_expired, grant_request_replayed, grant_request_invalid, and grant_nonce_mismatch, the response SHOULD include a grant detail object containing grant_request_id when the request identifier is known.

Field validation errors For invalid_request, registration_invalid, revocation_invalid, approval_envelope_invalid, overlay_signature_invalid, engagement_countersign_required, engagement_signature_invalid, change_log_immutable, pkce_required, invalid_webhook_uri, and invalid_target, the response SHOULD include a field detail object identifying the rejected field or parameter when a single field caused the rejection.

Subscription errors For subscription_auth_required, subscription_scope_forbidden, and subscription_limit_exceeded, the response SHOULD include a subscription detail object when a subscription identifier, subscriber DID, or limit value is known.

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 33

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 numeric limits below are RECOMMENDED default ceilings for unauthenticated or baseline clients. Registry operators MAY enforce lower ceilings when observed traffic patterns or threat models require stricter throttling, and MAY grant higher ceilings to authenticated or verified clients where this section allows higher limits. Rate-limit conformance is based on implementing the separate buckets, enforcing a documented effective limit for each bucket, and returning the response format in Section 19.1. The exact numeric thresholds are operational policy unless a requirement below uses MUST.

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 direct-origin reads and the published endpoints.crl retrieval URI):

* The published CRL retrieval URI MUST be served from a CDN or distributed infrastructure (Section 11.3). Direct-origin CRL reads to /v1/crl 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. A principal_revoke whose target_id is a Principal DID is principal-wide. A principal_revoke whose target_id is a specific agent AID is scoped to that AID’s authorisation and does not by itself block registration of unrelated AIDs for the same principal.

19.3.3. Registration flood from shared principals

Operational guidance (non-normative): a high number of registrations under a single principal DID can be an abuse signal. A Registry can apply local policy, such as deployer review or out-of-band business verification, when registrations from one principal exceed an implementation-defined threshold. A value such as 1,000 agents can be used as an example review trigger, but this specification does not define a protocol-visible proof of legitimate use, an interoperable review mechanism, or a conformance threshold for shared-principal registration volume.

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 the deployer DID is self-asserted, unauthenticated to the Registry, or otherwise not trusted by local Registry policy.

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 the historical key retention window defined in Section 19.7. This prevents adversaries from constructing tokens with ancient, never-rotated keys to force deep history lookups without invalidating any conforming unexpired token.

Relying Parties perform the Step 2a expiration preflight before historical key lookup. Therefore an expired token MUST be rejected with token_expired before a missing historical key can produce unknown_aid.

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 the Registry's published step_claim_timeout_seconds value MUST be automatically failed by the Registry. The timeout interval starts at the step's claimed_at timestamp. The step_claim_timeout_seconds value MUST be an integer from 1 to 600 seconds inclusive and MUST be published in /v1/registry-metadata.

This prevents a claimed step from blocking the workflow indefinitely due to a crashed or unresponsive agent. Timeout-induced failure MUST trigger the same envelope failure and compensation evaluation as an explicit step failure reported by the actor.

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.

19.7. Historical Key Retention Requirements

A Registry that accepts key rotation for an AID MUST retain each retired public verification key for historical validation until at least:

```
retention_not_before =
  retired_key.valid_until
  + max_credential_token_ttl_for_registry
  + accepted_clock_skew
```

max_credential_token_ttl_for_registry is the largest effective Credential Token TTL that the Registry could have accepted for any active scope at the time the key was retired, after applying the Tier ceilings in Section 8.2 and the synced AIP Scope Catalog. The accepted_clock_skew value MUST be at least 30 seconds. Registries MAY retain retired keys longer than this minimum. The RECOMMENDED operational retention window is 90 days after valid_until, but that window MUST be extended when the formula above yields a later time.

Historical key retention is a validation-availability rule, not an authority extension. Agents MUST stop issuing Credential Tokens with a retired private key immediately after a successful rotation, and Relying Parties MUST still reject tokens whose exp, delegation chain, or revocation state fails validation.

20. Versioning and Compatibility

AIP uses separate identifiers for Internet-Draft document revisions and wire protocol compatibility. The suffix in an Internet-Draft name, such as -00, -01, or -02, is a document revision identifier only. It is not an AIP protocol version and MUST NOT appear in aip_version, X-AIP-Version, Credential Tokens, GrantRequests, Registry metadata, or other protocol messages.

The aip_version claim and the X-AIP-Version HTTP header carry the AIP protocol compatibility version. This Internet-Draft revision defines AIP protocol version 0.3. Multiple Internet-Draft revisions MAY define the same protocol version when they contain editorial changes, clarifications, examples, schema corrections, or other changes that do not intentionally create a new wire compatibility class.

AIP protocol versions follow Semantic Versioning compatibility intent, represented on the wire as MAJOR.MINOR. Before v1.0, MINOR versions MAY include breaking changes. A future Internet-Draft revision that intentionally introduces breaking wire behavior, incompatible validation semantics, or incompatible message schemas MUST allocate a new protocol version before publication, for example by moving from 0.3 to 0.4. Backward-compatible clarifications and non-breaking errata MAY retain the existing aip_version.

AIP Catalog snapshot identifiers, such as draft-02, are also separate from aip_version. A Registry MAY report both its AIP protocol version and its synced Catalog snapshot, but verifiers MUST NOT treat a Catalog snapshot identifier or Internet-Draft revision as a substitute for protocol version negotiation.

Internet-Draft revision	AIP protocol version	Catalog snapshot
draft-singla-agent-identity-protocol-00	0.1	draft-00
draft-singla-agent-identity-protocol-01	0.2	draft-01
draft-singla-agent-identity-protocol-02	0.3	draft-02

Table 34: Draft Revision to Protocol Version Mapping

Catalog snapshots are immutable for conformance purposes. A Registry claiming Draft-02 conformance MUST advertise aip_catalog_version: "draft-02", aip_catalog_snapshot_id: "https://provai.dev/aip/catalog/draft-02", and a pinned aip_catalog_sha256 digest as defined in Section 17.15. A change to any standard Catalog entry or standard Catalog security metadata requires a new Catalog Snapshot identifier and MUST NOT be silently introduced under the existing Draft-02 snapshot.

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 17.15).

* Registration Envelopes MUST include grant_tier.

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

* The /v1/registry-metadata response MUST include registry_id, registry_trust_uri, registry_name, and endpoints. Registry Trust Records are versioned separately and MUST support sequential updates.

* Section 19.6 (Graduated Backoff Requirements): the backoff formula has been corrected to use exponential backoff with a base interval of 1 second, a multiplier of 2, and a maximum interval of 3600 seconds, replacing the previous linear backoff formula. This change ensures that Registry lookup storms are dampened under sustained failure conditions.

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

A Relying Party or Registry conforming only to protocol version 0.3 MUST reject any request, Credential Token, Step Execution Token, GrantRequest, or other AIP protocol message whose aip_version or X-AIP-Version value is absent, not "0.3", or mutually inconsistent, unless the implementation has explicitly implemented that other protocol version. The error response MUST be unsupported_version and SHOULD include X-AIP-Supported-Versions and the unsupported value in the details object. Implementations MUST NOT treat an unknown future version, including a numerically greater value such as 0.4, as forward-compatible by default.

20.1. Tier Conformance

Tier	Revocation	DPoP	mTLS	grant_tier	Principal DID Method
1	CRL (15 min)	NOT REQUIRED	NOT REQUIRED	G1, G2, or G3	Any non-did:aip W3C DID method; see requirements below
2	Real-time	REQUIRED	NOT REQUIRED	G2 or G3; see requirements below	did:web; see requirements below
3	Real-time	REQUIRED	REQUIRED	G3	did:web; see

	+ OCSP				requirements below	

Table 35

Tier 1 Principal DID method For Tier 1, the principal MAY use any W3C DID method except did:aip. Principals using did:key are permitted for Tier 1 only.

Principal DID method restrictions did:aip is NEVER a valid Principal DID method for any Tier. The principal.id field MUST NOT use the did:aip method. Principals using any DID method other than did:web for Tier 2 or Tier 3 agents MUST be rejected with principal_did_method_forbidden. For Tier 2 and Tier 3, the did:web DID Document MUST contain the AIPRegistry service entry required by Section 7.3.

Tier 2 Registry identity-proofing policy If identity_proofing_required_for_tier2 is true in Registry metadata, Tier 2 permits only G3 for that Registry.

The table defines minimum Grant Tier conformance. A higher-assurance Grant Tier MAY satisfy a lower security Tier, but a lower-assurance Grant Tier MUST NOT satisfy a higher security Tier. Relying Parties enforce this relationship for each participating aip_chain AID at runtime in Section 9, Step 9d.

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.

TS-10: Step Execution Token Misuse
An adversary forges, replays, or re-targets a Step Execution Token to execute an Approval Envelope step outside the authorized actor, audience, or step context. Mitigation: SET validation profile, Registry Trust Record step-execution keys, Step 10a verification, DPoP binding, and per-step status checks.

TS-11: Approval Lifecycle Race
Concurrent approval, rejection, expiry, cancellation, or claim operations produce inconsistent Approval Envelope state. Mitigation: atomic lifecycle transitions, compare-and-set or equivalent locking, terminal-state rules, and approval_state_conflict.

TS-12: Orchestrator Cascade
A compromised or over-privileged orchestrator uses one approval or delegation to trigger a larger cascade of dependent actions than the principal intended. Mitigation: pre-declared Approval Envelope steps, per-step actor and action-hash verification, step count limits, dedicated destructive-scope approval, and SAGA compensation rules.

TS-13: Overlay Injection
An adversary injects or replays a Capability Overlay to expand or mis-scope an agent's effective capabilities. Mitigation: Overlay signatures, issuer validation, monotonic overlay versions, CO-1 attenuation-only validation, engagement scoping, and catalog constraint validation.

TS-14: Engagement Tampering
An adversary modifies Engagement Object participants, lifecycle state, approval gates, or change-log entries to alter authorization context. Mitigation: required countersignatures, append-only signed change logs, strict sequence validation, participant authorization checks, and termination cascade rules.

TS-15: Model Substitution
An agent deployer registers with a trusted model identifier but executes a different model binary or version manifest. Mitigation: model.attestation_hash SHOULD be present for Tier 2 deployments and MUST be present for Tier 3 deployments. The Registry SHOULD accept a Tier 2 registration without model.attestation_hash only when it persists and returns the structured model_attestation_missing_tier2 registration warning defined in Section 6.2. The Registry MUST reject a Tier 3 registration when model.attestation_hash is absent.

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

Table 36: Mandatory-to-Implement Cryptography

This specification does not define an AIP key-exchange or encryption handshake. X25519 and other key-agreement mechanisms are reserved for a future extension and are not mandatory-to-implement in Draft-02. Implementations MUST NOT treat support for any key-agreement algorithm as a condition for conformance to this specification.

This specification reserves ES256 (ECDSA P-256) and RS256 (RSA-PKCS1), as defined by [RFC7518], for possible future extensions but does not define registration, resolution, rotation, AID derivation, or DPoP binding rules for those suites in protocol version 0.3. Credential Tokens, Principal Tokens, Step Execution Tokens, DPoP proofs, Agent Identity keys, and Registry public-key responses conforming to this specification MUST use EdDSA over Ed25519. Implementations MUST reject ES256, RS256, and other algorithm values for those artifacts.

Prohibited: none, HS256/384/512, RS512, MD5, SHA-1. The alg header MUST be explicitly specified. A DID verification method used to sign or verify a Credential Token, Principal Token, Step Execution Token, or DPoP proof MUST be an Ed25519 signing verification method. X25519 keys are key-agreement keys and MUST NOT be used for signing.

21.3. Proof-of-Possession (DPoP)

DPoP is REQUIRED for every Tier 2 or Tier 3 operation. It is also REQUIRED when any requested Tier 1 scope has requires_dpop: true in the synced AIP Scope Catalog, or when the target endpoint requires DPoP independently of scope metadata. DPoP proofs MUST use EdDSA.

Operation or request class	DPoP requirement
Tier 1 operation with no DPoP-required scope and no endpoint policy	Not required by AIP.
Tier 1 operation containing any scope whose synced Catalog entry has requires_dpop: true	Required.
Tier 1 Registry endpoint that independently requires DPoP, including key rotation, heartbeat submission, and RPNP subscription authentication	Required.
Tier 2 operation	Required for every request.
Tier 3 operation	Required for every request, in addition to mTLS.

Table 37: DPoP Applicability

Endpoint-level DPoP requirements are independent of the operation's security Tier. For example, PUT /v1/agents/{aid} key rotation and POST /v1/agents/{aid}/heartbeat require DPoP even when the affected agent was originally registered only for Tier 1 scopes.

For standard AIP scopes, the Draft-02 catalog marks all destructive scopes and selected high-risk external-effect scopes as DPoP-required. This includes deletion scopes, filesystem mutation and execution scopes, transactions, communication scopes, spawn-agent scopes, and the non-destructive but high-risk email.send, web.download, and web.forms_submit scopes. The web.forms_submit scope is Tier 2 in the Draft-02 catalog because form submissions can create external account, data, or financial effects.

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, no query or fragment>",
  "iat": "<Unix timestamp>",
  "ath": "<BASE64URL(SHA-256(token))>"
}
```

Relying Party validation: Verify alg is EdDSA, verify htm/htu/iat, check jti replay, verify ath, verify signature, and verify the DPoP public key binding for the presented token type. For an agent-issued Credential Token, jwk.kid MUST match the Credential Token protected header kid. For a Step Execution Token, jwk.kid MUST identify Ed25519 public key material controlled by the SET sub actor AID; it MUST NOT be the Registry step-execution key that signed the SET.

The htu claim is the DPoP target URI binding. It MUST match the absolute target URI of the current request, excluding query and fragment components and using the normalization rules in [RFC9449]. A proof captured for one Relying Party, Registry, host, path, or method is not transferable to another target: validation MUST fail on htu or htm before replay-cache state is considered.

The ath claim binds the proof to the presented token. Its value MUST be the unpadded base64url encoding of the SHA-256 hash of the exact compact Credential Token or Step Execution Token value from the Authorization header, excluding the authorization scheme, surrounding whitespace, and DPoP proof. When DPoP is required, the authorization scheme is DPoP.

DPoP proof jti replay state is verifier-local. A verifier MUST maintain a DPoP replay cache separate from the Credential Token replay cache and keyed by (kid, jti). The cache window MUST be at least the accepted DPoP proof age window: 300 seconds before receipt plus 30 seconds of future clock skew, unless a server-provided DPoP

nonce is required under [RFC9449]. When nonce validation is used, the replay cache MUST retain entries for at least the nonce lifetime. AIP does not require a global or cross-Registry DPoP replay cache because cross-target replay is rejected by htu, htm, and ath validation.

21.4. Key Management

Private keys MUST NOT be stored in plaintext at rest, transmitted in protocol messages, or included in logs. Private keys SHOULD be stored in an HSM, secure enclave, OS-level keychain, or an equivalent protected key store where available.

A private key satisfies the at-rest protection requirement when at least one of the following conditions is met:

1. The key is generated and used inside an HSM, secure enclave, operating-system keychain, cloud KMS, or comparable protected key store that prevents ordinary application processes from reading the raw private key material.

2. The key is stored only as ciphertext produced by envelope encryption using AES-256-GCM, or an equivalent authenticated-encryption scheme with at least 128-bit security strength, unique nonce or IV use per encryption key, integrity authentication of the ciphertext, and associated data that binds the ciphertext to the AID, key identifier, key version, and intended key purpose.

A protected key store or encryption design is equivalent for this specification only if it enforces access control for key use and key administration, separates key-encryption keys from encrypted private-key blobs, supports key-encryption-key rotation without exposing plaintext private keys, records administrative key export or deletion events, and does not depend on hard-coded wrapping keys or wrapping keys stored in the same plaintext database row, file, or object as the encrypted private key. Backup copies of private keys MUST satisfy the same controls as primary copies.

The protected-key-store recommendation is a local implementation conformance requirement, not a wire-verifiable protocol condition. Registries and Relying Parties cannot determine from Credential Tokens or Registry API requests whether an implementation uses an HSM, secure enclave, keychain, or equivalent mechanism. They MUST NOT treat protected-key-store use as a Registry acceptance check or Credential Token validation precondition. Deployment profiles MAY require external audit, attestation, or certification of key-storage controls using the audit evidence profile below.

Deployment key-management audit evidence: A deployment profile that requires audit evidence for local key-management controls MUST define an audit interval and MUST collect, at minimum, the following evidence without recording private key material, seed material, or plaintext recovery secrets:

1. Key inventory records identifying each AID, key identifier, identity.version, key creation time, key activation time, key retirement time when applicable, and the protected storage boundary used for the key.

2. Control evidence that private keys are not stored in plaintext at rest, transmitted in protocol messages, or included in logs. Acceptable evidence includes configuration attestations, HSM or secure-enclave policy exports, operating-system keychain policy exports, or signed operator attestations bound to a deployment profile.

3. Rotation records for every key rotation, including the old key identifier, new key identifier, submitted Agent Identity version, previous_key_signature verification result, Registry acceptance time, and the time at which issuance with the retiring private key stopped.

4. Compromise-response records for every key_compromised event, including the Revocation Object identifier, affected AID, time of revocation submission, and delegation re-establishment records if a replacement AID is registered.

5. For sub-agent delegation where the parent generates the child key, provisioning and deletion records containing the parent AID, child AID, provisioning time, secure channel or key-wrapping mechanism identifier, deletion time for any parent-held child private-key copy, and the local process or operator identity that performed the deletion.

Audit records for key_compromised events MUST NOT include verbatim Credential Token or Step Execution Token payloads. Token metadata (AID, jti, scope list, iat, exp) is sufficient for key-compromise timeline reconstruction and is the maximum permissible audit record content for token-related events.

Audit evidence MAY be reviewed by deployment operators, external auditors, or certification programs, but it is not exchanged as part of Credential Token validation. A Registry or Relying Party MUST NOT reject an otherwise valid protocol message solely because this audit evidence is absent from the wire message. Unless a deployment claims the Deployment Key-Management Audit Profile in Section 23.3, compliance with these local key-management controls is self-attested by the implementation's conformance claim.

Key rotation: Generate a new keypair, increment version, include previous_key_signature signed by the retiring key, and submit the new public key to the Registry. The Registry MUST retain the retiring public verification key according to the historical key retention rule in Section 19.7. The agent MUST stop issuing new tokens with the retiring private key after rotation succeeds.

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. The issuer AID is the replay domain: an agent MUST NOT reuse a jti across different delegation chains, principals, grants, scopes, or audiences.

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

21.6. Tier 3 mTLS Certificate Profile

Tier 3 operations require mutual TLS with an X.509 client certificate that binds the TLS client to the effective actor AID. The binding MUST use the certificate subjectAltName extension, not the subject DN or common name.

1. The Relying Party MUST validate the client certificate path per [RFC5280] to a trust anchor explicitly configured for the target Registry, resource server, or deployment profile. The public Web PKI MUST NOT be treated as sufficient for AIP actor binding unless that CA set is explicitly configured as the deployment's Tier 3 client-certificate trust anchor set.

2. The certificate MUST be within its validity interval, MUST permit digital signatures in keyUsage when keyUsage is present, and MUST contain the id-kp-clientAuth extended key usage when extendedKeyUsage is present.

3. The certificate MUST contain exactly one URI subjectAltName value that matches the did:aip AID syntax. That URI SAN value is the certificate actor AID. The Relying Party MUST compare it byte-for-byte to the effective actor AID after ordinary URI SAN string extraction; it MUST NOT use subject DN, common name, email SAN, DNS SAN, or organization fields for actor binding.

4. If the client certificate is absent, the Relying Party MUST reject with mtls_required. If path validation fails, required key-usage or EKU checks fail, the AIP URI SAN is absent, more than one AIP URI SAN is present, or the AIP URI SAN does not equal the effective actor AID, the Relying Party MUST reject with invalid_token.

5. The Relying Party MUST perform certificate revocation checking using OCSP per [RFC6960] or a deployment-profile mechanism that provides at least equivalent freshness and issuer authorization. An equivalent deployment-profile mechanism MUST, at minimum, bind status to the certificate issuer or delegated status authority, provide signed or otherwise authenticated certificate status, define a maximum status freshness interval no weaker than the OCSP policy it replaces, fail closed on unavailable or indeterminate status, and be documented in the deployment's mtls_trust_anchors_uri material. A revoked client certificate MUST be rejected with agent_revoked. An unavailable, stale, or indeterminate certificate revocation status MUST be rejected with invalid_token.

A Registry or deployment that accepts Tier 3 registrations or advertises Tier 3 operation support MUST document the client-certificate trust anchor set and revocation-checking mechanism. If this information is published through Registry Metadata, the fields are mtls_client_certificate_profile and mtls_trust_anchors_uri.

21.7. 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. The Registry MUST auto-revoke ephemeral agents when their lifecycle_expires_at deadline passes, as defined in Section 15. A Registry that leaves an ephemeral AID active after that deadline is non-conformant, even if Relying Parties would separately reject expired Principal Tokens or Capability Manifests during validation.

Parent agents that generate child-agent private keys during sub-agent delegation MUST delete any retained copy after successful provisioning. This requirement is not wire-verifiable by Registries or Relying Parties; it is an implementation conformance requirement for the parent agent's key-management boundary. Deployment profiles that audit this requirement MUST use the sub-agent provisioning and deletion evidence defined in Section 21.4.

21.8. Registry Security

All write operations MUST be authenticated. Revocation issued_by, kid, issuer authorization, and signature verification MUST follow the ordered Revocation Submission Validation algorithm in Section 11.2.

Registry read endpoint availability is an operational service-level objective, not a wire-protocol conformance condition. Deployment profiles MAY define measurable availability targets, measurement intervals, exclusions, evidence, and remedies. This specification does not define conformance failure solely by an uptime percentage. 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.

21.9. 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 an authenticated heartbeat to POST /v1/agents/{aid}/heartbeat within a configured window (RECOMMENDED: 24 hours). The heartbeat authentication and liveness timestamp rules are defined in Section 17.11.

Registries that enable the Dead Man's Switch for an AID MUST measure the timeout from the Registry-recorded last_heartbeat_at value or, if no heartbeat has been accepted, from the AID registration time. When issuing an automatic revocation for heartbeat timeout, the Registry SHOULD use revocation reason heartbeat_timeout.

Timing attack mitigation: Short ttl_max_seconds values for sensitive scopes in the synced AIP Scope Catalog limit exploitation windows.

21.10. 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.

21.11. 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 or Step Execution 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. JSON Schema and Catalog Artifact Index

The following JSON Schema definitions are normatively referenced throughout this specification. They are part of the archived publication package for this draft in JSON Schema Draft 2020-12 format under schemas/. The Draft-02 AIP Catalog Bundle is maintained as the deterministic dist/catalog.json release artifact in the separate aip-catalog (https://github.com/provai-dev/aip-catalog) repository. The working repository at aip-rfc (https://github.com/provai-dev/aip-rfc) is a convenience mirror; a branch head, mutable tag, or /latest schema URI is not a normative artifact.

For this draft, the immutable schema identifier for each listed schema is the $id value ending in /draft-02. Implementations claiming Draft-02 conformance MUST validate against those /draft-02 identifiers. If a local filename, repository URL, or mirror URL conflicts with the schema identified by the $id, the $id and the archived publication package are authoritative.

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

agent-registration.schema.json Defines the default GET /v1/agents/{aid} Agent Registration Metadata response (Section 17.4).

aip-registry.schema.json Defines the Registry Metadata returned by GET /v1/registry-metadata (Section 17.14.1).

aip-gateway-configuration.schema.json Defines the AIP gateway OAuth Protected Resource Metadata returned by GET /.well-known/oauth-protected-resource (Section 7.5).

public-key-response.schema.json Defines the Public-Key response returned by GET /v1/agents/{aid}/public-key and GET /v1/agents/{aid}/public-key/{key-id} (Section 17.6).

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).

step-execution-token.schema.json Defines the Step Execution Token JWT payload structure (Sections 5.4.1 and 13.8).

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).

resource-record.schema.json Defines the Registered Resource Record used by the Registry resource registration endpoints (Section 17.13).

registry-trust-record.schema.json Defines the Registry Trust Record structure used for Registry trust bootstrapping and continuity (Section 7.3).

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

revocation-status.schema.json Defines the live Revocation Status response for GET /v1/agents/{aid}/revocation (Section 17.8).

crl.schema.json Defines the signed AIP CRL document returned by GET

/v1/crl and signed endpoints.crl distribution URIs (Section 11.3 and Section 17.10).

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

error-response.schema.json Defines the standard AIP JSON error response body (Section 18.1).

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

approval-step-endpoints.schema.json Defines request and response body profiles for Approval Envelope forward-step and compensation-step claim, complete, and fail endpoints (Section 13.8 and Section 17.12).

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).

rpnp.schema.json Defines Registry Push Notification Protocol subscription request, subscription response/status, and push payload structures (Section 11.5).

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

The Draft-02 AIP Catalog Bundle is the immutable catalog artifact identified by catalog_snapshot_id: "https://provai.dev/aip/catalog/draft-02". Its authoritative bytes are the bytes of the immutable dist/catalog.json release artifact in the provai-dev/aip-catalog repository, or a mirror whose SHA-256 digest exactly matches the aip_catalog_sha256 value published by the Registry. Catalog retrieval, mirroring, and digest validation rules are defined in Section 17.15. A Registry MUST NOT use a mutable catalog source to determine Draft-02 conformance.

23. Conformance

An implementation claiming conformance to this specification MUST identify each conformance class and optional feature profile it implements. A conformance claim MUST state the AIP protocol compatibility version (0.3 for this specification), the implemented class names from this section, any optional profiles, and the AIP Catalog Snapshot identifier and digest used for catalog-bound validation.

A conformant implementation MUST satisfy every MUST and REQUIRED requirement that applies to each claimed class or profile. A SHOULD or RECOMMENDED requirement applies unless the implementation documents a specific reason for deviation and the deviation does not violate any MUST requirement. Optional profiles are not required for a base class claim, but an implementation that claims an optional profile MUST satisfy all requirements listed for that profile.

23.1. Common Requirements

Every conformance class MUST implement the conventions, identifiers, algorithms, and serialization rules in Section 2, Section 3, and Section 4 that are needed for the messages it sends, receives, signs, or validates. Implementations that process JSON schemas MUST use the Draft-02 schema identifiers listed in Section 22. Implementations that send or receive AIP-aware HTTP messages MUST implement the version-header rules in Section 8.1 and Section 20. Implementations that return AIP error responses MUST use Section 18.

23.2. Conformance Classes

Class	Implementation role	Required requirement set
AIP Registry	Authoritative registry, resolver, and metadata service	Registration processing in Section 6, did:aip resolution and Registry trust records in Section 7, revocation submission and CRL publication in Section 11 and Section 17.9, unconditional Registry endpoints and catalog sync in Section 17, rate limiting in Section 19, versioning in Section 20, Registry security

		requirements in Section 21, and Token Payload Non-Retention. A conformant Registry MUST NOT persist Credential Token or Step Execution Token payloads, meaning the full JWT payload object, in any log, database, or audit record beyond completion of the validation algorithm for that token. A Registry MAY retain token metadata (AID, jti, aip_scope list, exp timestamp, and validation outcome) for audit purposes. This requirement can be demonstrated through audit log schema inspection showing no payload field in stored validation records.
AIP Relying Party	Service or agent that accepts AIP Credential Tokens or Step Execution Tokens	Credential Token validation in Section 9, Tier conformance in Section 20.1, revocation checking in Section 11.4, DPoP and mTLS checks when required by Section 21, catalog-bound scope validation in Section 17.15, and applicable rate-limit client behavior in Section 19.
AIP Agent Issuer	Agent runtime or signer that issues Credential Tokens for its own AID	Credential Token structure and issuance rules in Section 8.1, token lifetime and refresh rules in Section 8.2 and Section 8.3, delegation-chain rules in Section 10, DPoP proof

		construction when required by Section 21, and private-key handling requirements in Section 21.
AIP Agent Deployer	Party that constructs GrantRequests, agent identity material, Capability Manifests, and Registration Envelopes	Agent Identity and Capability Manifest construction in Section 5, AIP-GRANT initiation and GrantResponse validation in Section 12, and Registration Envelope submission requirements in Section 6.
Principal Wallet	Principal-controlled grant, consent, and root Principal Token signing component	A Principal Wallet is conformant if it satisfies all of the following: (1) it holds the principal’s DID private key in a manner consistent with the key-protection requirements in Section 21.4; (2) it implements the AIP-GRANT consent ceremony (Section 12) for at least one grant tier, G1, G2, or G3; (3) for scopes whose AIP Scope Catalog entry has destructive: true, it MUST display a mandatory two-step confirmation flow requiring explicit principal acknowledgement before signing the GrantRequest, including at minimum display of the destructive scope’s Catalog description and a distinct user action separate from initial grant submission; (4) it MUST NOT sign a GrantRequest whose

		requested scopes are denied by local principal policy or whose display metadata cannot be resolved from the trusted AIP Scope Catalog; and (5) its conformance claim MUST identify which grant tiers are implemented and whether Tier 3 mTLS client certificates are supported.
AIP OAuth Authorization Server	OAuth AS used for G3 grant ceremonies or token exchange	OAuth metadata and endpoint requirements in Section 17.14, G3 binding checks in Section 12.10, token exchange rules in Section 8.4, and applicable OAuth error handling required by the referenced OAuth specifications.
AIP RPNP Subscriber	Webhook receiver for Registry Push Notification Protocol deliveries	Subscription request authentication, HMAC verification, JWT payload verification, timestamp tolerance, replay handling, and fallback behavior in Section 11.5.

Table 38: AIP Conformance Classes

23.3. Optional Feature Profiles

Profile	Claiming implementation	Additional required requirement set
G3 Grant Profile	AIP Registry, AIP OAuth Authorization Server, Principal Wallet, or Agent Deployer	G3 ceremony, PKCE, OAuth binding, authorization-code binding, and identity-proofing claim requirements in Section 12.10 and Section 17.14.

Approval Envelope Registry Profile	AIP Registry	Approval Envelope storage, lifecycle, step claim, complete, fail, compensation, SET issuance, and idempotency requirements in Section 13 and the corresponding endpoints in Section 17.
Approval Envelope Relying Party Profile	AIP Relying Party	Step Execution Token validation and Approval Envelope step verification in Section 9 Step 10a and the SET validation profile.
Engagement Object Profile	AIP Registry, Agent Deployer, or Relying Party	Engagement Object, Capability Overlay, participant lifecycle, countersignature, and engagement-gate requirements in Section 5.12, Section 5.11, and Section 17.17.
RPNP Registry Profile	AIP Registry	RPNP subscription, delivery, retry, degraded-state, and endpoint requirements in Section 11.5 and Section 17.18.
Dead Man’s Switch Profile	AIP Registry or Agent Issuer	Heartbeat endpoint, authenticated liveness, timeout, and revocation behavior in Section 21.8 and Section 17.
Deployment Key-Management Audit Profile	Principal Wallet, Agent Issuer, Agent Deployer, or deployment operator	Protected-key-store evidence, key inventory, rotation, compromise-response, and sub-agent provisioning/deletion evidence in Section 21.4. This profile is not a wire validation profile and does not change Registry or Relying Party acceptance rules.
Endorsement Acceptance (Optional	AIP Registry	A Registry that implements POST /v1/endorsements MUST enforce the minimum acceptance

Feature)		rules in Section 14.1.1. Reputation scoring beyond these minimums is implementation-defined and not subject to conformance testing. A Registry that does not implement the endorsement endpoint is still a conformant Registry; it MUST respond to POST /v1/endorsements with HTTP 501 Not Implemented.
Reputation Output (Optional Feature)	AIP Registry	A Registry that implements GET /v1/agents/{aid}/reputation MUST return the required fields and numeric range invariants in Section 14.2. The conformance surface for this profile is limited to endpoint availability, required fields, response types, pagination of revocation_history, and the score range 0.0 through 5.0. The scoring algorithm, weighting, aggregation, and decay policy remain implementation-defined.

Table 39: AIP Optional Feature Profiles

23.4. Conformance Testing Methodology

A conformance claim MUST be supported by test evidence for every claimed class and optional feature profile. At minimum, the implementation MUST execute schema-validation tests for every JSON object it emits or accepts, positive and negative tests for each applicable validation step in Section 9, and endpoint behavior tests for every Registry, Relying Party, wallet, deployer, or subscriber endpoint included in the claim.

Test evidence MUST include the AIP protocol version, Catalog Snapshot identifier and digest, implementation version, tested conformance classes, optional profiles, test execution timestamp, and pass/fail status for each applicable MUST-level requirement. A failed MUST-level test invalidates the corresponding conformance claim until corrected or until the implementation removes that class or profile from its claim.

Until a separate AIP test-vector document is published, implementations MAY self-attest by publishing a machine-readable conformance claim and a reproducible test report. Self-attestation does not create certification by this specification, but it is the minimum evidence required for a conformant claim.

23.5. Conformance Claim Content

A conformance claim SHOULD be machine-readable. When published as JSON, it SHOULD include at least: aip_version, draft, classes, profiles, catalog_snapshot_id, catalog_sha256, implementation identifier, test_report_uri, test execution timestamp, and contact or audit URI. A Registry MAY publish this information in or alongside /v1/registry-metadata. Absence of a conformance claim MUST NOT be interpreted as conformance to any class or optional profile.

24. IANA Considerations

This document requests IANA registration only of the media types in Section 24.7. AIP uses existing registered mechanisms for HTTP authorization and discovery: the Bearer and DPoP authentication schemes, OAuth Authorization Server Metadata, and OAuth Protected Resource Metadata. The did:aip DID method, AIP scope identifiers, AID namespaces, grant tier values, and AIP error code values are not IANA registries in this version of the specification; their change control is described in the subsections below.

24.1. Non-IANA DID Method Registry

The W3C DID Method Registry is not an IANA registry. This document requests no IANA action for the did:aip DID method. If the did:aip method is submitted to the W3C DID Method Registry, the following registration information applies.

Field	Value
Method Name	did:aip
Status	Draft
Controller	The principal or deployer that controls the registered Agent Identity Object and the current private key corresponding to the AID public key.
DID Syntax	did:aip:<namespace>:<32-lowercase-hex-agent-id>; the complete ABNF is defined in Section 4.1.
Create	Create is performed through POST /v1/agents using a Registration Envelope validated by Section 6.
Read	Read is performed through DID resolution against the authoritative Registry and the GET /v1/agents/{aid} and public-key endpoints defined in Section 17.
Update	The DID identifier is immutable. The only protocol-defined update operation is key rotation through PUT /v1/agents/{aid} with previous-key authorization as defined in Section 17.5.
Deactivate	Deactivate is represented by accepted Revocation Objects and DID Document deactivation metadata as defined in Section 11 and Section 7.
Security Considerations	AIDs are derived from Ed25519 public key material, key rotation is versioned and previous-key signed, historical keys are retained for validation, and revocation is checked according to the Tier rules in this specification.
Privacy Considerations	AID resolution exposes agent metadata necessary for validation. Registries MUST follow the token payload non-retention and privacy-preserving audit requirements in Section 21 and Section 23.

Table 40: did:aip Method Registration

24.2. Existing HTTP and Discovery Registrations

AIP Credential Tokens and Step Execution Tokens are presented using the registered Bearer HTTP authentication scheme [RFC6750] for requests that do not require proof-of-possession, and using the registered DPoP HTTP authentication scheme [RFC9449] for DPoP-bound requests. AIP Registries that act as OAuth Authorization Servers publish authorization-server metadata using the registered oauth-authorization-server well-known URI suffix defined by [RFC8414]. AIP-protected gateways publish protected resource metadata using the registered oauth-protected-resource well-known URI suffix defined by [RFC9728].

This document does not define a new HTTP authentication scheme and does not define a new Well-Known URI. Registry Metadata is published at the ordinary Registry API endpoint /v1/registry-metadata relative to the Registry's registry_id origin.

24.3. AIP Scope Identifiers

AIP scope identifiers are OAuth scope-token compatible dot-notation strings defined by the applicable AIP Catalog Snapshot. This document does not define a URN namespace for AIP scope values.

Concrete AIP scope registrations are maintained in the external AIP Catalog for the applicable draft snapshot. This document does not request IANA maintenance of individual AIP scope values.

24.4. AID Namespace Catalog

Concrete did:aip namespace registrations are maintained in the external AIP Catalog for the applicable draft snapshot. IANA is not requested to maintain individual AIP namespace values in this document.

24.5. AIP Grant Tier Values

Grant tier values are protocol values defined by this specification, not an IANA registry. This document requests no IANA action for grant tier values. Future changes to these values require a specification update that defines the new value, its registration and authorization behavior, and its conformance impact.

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

Table 41: Grant Tier Values

24.6. AIP Error Codes

AIP error codes are protocol values defined by this specification, not an IANA registry. This document requests no IANA action for AIP error code values. Future changes to these values require a specification update that defines the code, the default HTTP status, the applicable endpoint or validation step, and the corresponding error response requirements.

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.

invalid_request (HTTP 400)
: Request syntax, query parameter, pagination cursor, or endpoint-specific request parameter is invalid.

unsupported_version (HTTP 400)
: X-AIP-Version or token aip_version is missing, unsupported, or mutually inconsistent.

dpop_proof_required (HTTP 401)
: DPoP proof is required for the request but the DPoP header is absent. Malformed or invalid DPoP proofs return invalid_token.

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 unavailable, signature failed, or AIP Scope Catalog constraint validation failed.

manifest_expired (HTTP 403)
Capability Manifest expires_at passed.

approval_envelope_invalid (HTTP 400)
Approval Envelope malformed, signature failed, dependency invalid, or value/hash mismatch.

approval_envelope_expired (HTTP 403)
The Approval Envelope was not approved before approval_window_expires_at.

approval_not_found (HTTP 404)
Approval Envelope ID not found.

approval_state_conflict (HTTP 409)
Approval Envelope state changed before the requested approval or rejection transition could be committed.

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_not_claimed (HTTP 409)
Step Execution Token presented before the Registry has accepted a claim for that step.

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.

compensation_failed (HTTP 409)
: An Approval Envelope compensation step failed and the envelope is in terminal failed state requiring out-of-band remediation.

engagement_cancelled (HTTP 409)
: The referenced Approval Envelope or step was cancelled because its parent Engagement Object was terminated. The operation cannot proceed.

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, expired, missing from a G3 authorization request, signature failed, inconsistent with OAuth request binding, or cannot be resolved to trusted catalog scope display metadata for consent.

grant_rejected_by_principal (HTTP 403)
: Principal declined the grant.

grant_nonce_mismatch (HTTP 400)
: GrantResponse nonce does not match.

grant_tier_insufficient (HTTP 403)
: Registered grant_tier is absent, unrecognised, or below the Grant Tier required for the operation’s security Tier.

registration_invalid (HTTP 400)
: Registration Envelope malformed or failed validation, including missing or invalid grant_tier.

aid_already_registered (HTTP 409)
: The submitted AID or public key is already associated with a registered, non-revoked AID.

unknown_aid (HTTP 404)
: AID not registered in any accessible Registry.

identity_version_conflict (HTTP 409)
: Agent Identity key rotation version is stale, skips a version, or conflicts with a committed rotation.

revocation_invalid (HTTP 400)
: Revocation Object is malformed, uses a reserved externally submitted reason, has an invalid timestamp, or has an invalid or unverifiable signature.

revocation_unauthorized (HTTP 403)
: Revocation issuer is not authorized for the target or revocation type.

revocation_conflict (HTTP 409)
: Revocation identifier was previously accepted with different object content.

registry_unavailable (HTTP 503)
: Registry or a required DID resolver could not be reached or timed out.

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

mtls_required (HTTP 403)
: Tier 3 operation without mTLS.

invalid_scope (HTTP 400)
: Requested scope is unknown, reserved, removed, retired (including bare spawn_agents), conflicting with the synced AIP Scope Catalog, or outside an explicitly documented local/private extension policy.

principal_did_method_forbidden (HTTP 403)
: Principal DID method is not permitted for the operation Tier; Tier 2 and Tier 3 require did:web.

identity_proofing_insufficient (HTTP 403)
: G3 identity proofing is absent or does not satisfy requested acr_values.

enterprise_policy_denied (HTTP 403)
: The enterprise IdP denied the token exchange request due to Conditional Access, ABAC, or equivalent enterprise policy.

idp_client_misconfigured (HTTP 500)
: The AIP Registry’s enterprise IdP client credentials are invalid or expired.

invalid_grant (HTTP 400)
: The enterprise IdP rejected the secondary token exchange grant.

a2a_originator_invalid (HTTP 400)
: aip_originator_aid is absent in an A2A delegated context, malformed, self-referential, or not bound to the validated delegation chain.

endorsement_invalid (HTTP 400)
: Endorsement Object validation failed, including invalid signature or out-of-range score.

gateway_config_invalid (HTTP 400)
: A gateway OAuth Protected Resource Metadata document is internally inconsistent or violates AIP gateway metadata constraints.

enterprise_assertion_missing (HTTP 403)
: Tier 3 Credential Token does not contain the required aip_principal_assertion claim.

enterprise_assertion_invalid (HTTP 403)
: aip_principal_assertion is malformed, has an untrusted issuer, or fails enterprise IdP signature verification.

enterprise_assertion_principal_mismatch (HTTP 403)
: aip_principal_assertion.sub does not match the root Principal Token principal identifier in aip_chain[0].

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 or not using code_challenge_method S256.

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

overlay_exceeds_manifest (HTTP 400)
: Overlay violates CO-1 attenuation rule or AIP Scope Catalog constraint validation.

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 Engagement Object countersignature.

engagement_signature_invalid (HTTP 400)
Engagement Object signature missing, malformed, bound to the wrong DID, or failed verification.

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 request missing DPoP, using an invalid DPoP proof, or failing to bind subscriber_did to the DPoP key or Authorization credential.

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.

24.7. Media Types

IANA is requested to register the following media types in the "Media Types" registry using the registration procedure defined by [RFC6838].

Type	Description
application/aip+jwt	AIP Credential Token (JWT format)
application/aip-set+jwt	AIP Step Execution Token (JWT format)
application/aip-crl+json	AIP Certificate Revocation List (signed JSON format)

Table 42: AIP Media Types

The typ header value for AIP Credential Tokens is AIP+JWT per RFC 7515. The typ header value for AIP Step Execution Tokens is AIP-SET+JWT.

24.7.1. application/aip+jwt

Type name:
application

Subtype name:
aip+jwt

Required parameters:
N/A

Optional parameters:
N/A

Encoding considerations:
7bit; values use compact JWT/JWS serialization.

Security considerations:
See Section 21 and the security considerations of [RFC7515] and [RFC7519]. AIP Credential Tokens are authorization credentials and can be bearer credentials unless bound to proof-of-possession material.

Interoperability considerations:
Implementations need the AIP Credential Token profile and validation rules in Section 8 and Section 9.

Published specification:
This document.

Applications that use this media type:
AIP agents, Registries, relying parties, and OAuth authorization servers that exchange or validate AIP Credential Tokens.

Fragment identifier considerations:
The syntax and semantics of fragment identifiers are not defined for this media type.

Additional information:
Magic number(s): N/A; file extension(s): N/A; Macintosh file type code(s): N/A.

Person and email address to contact for further information:
Paras Singla <paras.singla@inviscel.com>

Intended usage:
COMMON

Restrictions on usage:
None

Author:
Paras Singla <paras.singla@inviscel.com>

Change controller:
IETF

24.7.2. application/aip-set+jwt

Type name:
application

Subtype name:
aip-set+jwt

Required parameters:
N/A

Optional parameters:
N/A

Encoding considerations:
7bit; values use compact JWT/JWS serialization.

Security considerations:
See Section 21 and the security considerations of [RFC7515] and [RFC7519]. AIP Step Execution Tokens authorize individual approval-envelope steps and require audience, actor, step, status, expiry, and proof-of-possession validation before use.

Interoperability considerations:
Implementations need the Step Execution Token validation profile in Section 9.21.

Published specification:
This document.

Applications that use this media type:
AIP Registries, agents, and relying parties that claim or execute approval-envelope steps.

Fragment identifier considerations:
The syntax and semantics of fragment identifiers are not defined for this media type.

Additional information:
Magic number(s): N/A; file extension(s): N/A; Macintosh file type code(s): N/A.

Person and email address to contact for further information:
Paras Singla <paras.singla@inviscel.com>

Intended usage:
COMMON

Restrictions on usage:
None

Author:
Paras Singla <paras.singla@inviscel.com>

Change controller:
IETF

24.7.3. application/aip-crl+json

Type name:
application

Subtype name:
aip-crl+json

Required parameters:
N/A

Optional parameters:
N/A

Encoding considerations:
8bit; values are JSON encoded using UTF-8.

Security considerations:
See Section 21 and Section 11. AIP CRL documents carry revocation status and require signature, freshness, issuer, and pagination validation before relying parties use them for authorization decisions.

Interoperability considerations:
Implementations need the CRL structure and revocation-status validation rules in Section 11.

Published specification:
This document.

Applications that use this media type:
AIP Registries and relying parties that publish, fetch, cache, or verify revocation lists.

Fragment identifier considerations:
The syntax and semantics of fragment identifiers are not defined for this media type.

Additional information:
Magic number(s): N/A; file extension(s): N/A; Macintosh file type code(s): N/A.

Person and email address to contact for further information:
Paras Singla <paras.singla@inviscel.com>

Intended usage:
COMMON

Restrictions on usage:
None

Author:
Paras Singla <paras.singla@inviscel.com>

Change controller:
IETF

25. 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.

[RFC6750] Jones, M. and D. Hardt, "The OAuth 2.0 Authorization Framework: Bearer Token Usage", RFC 6750, DOI 10.17487/RFC6750, October 2012, <https://www.rfc-editor.org/rfc/rfc6750>.

[RFC6838] Freed, N., Klensin, J., and T. Hansen, "Media Type Specifications and Registration Procedures", BCP 13, RFC 6838, DOI 10.17487/RFC6838, January 2013, <https://www.rfc-editor.org/rfc/rfc6838>.

[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>.

[RFC7523] Jones, M., Campbell, B., and C. Mortimore, "JSON Web Token (JWT) Profile for OAuth 2.0 Client Authentication and Authorization Grants", RFC 7523, April 2015, <https://www.rfc-editor.org/rfc/rfc7523>.

[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>.

[I-D.ietf-oauth-v2-1] Hardt, D., Parecki, A., and T. Lodderstedt, "The OAuth 2.1 Authorization Framework", Work in Progress, Internet-Draft, draft-ietf-oauth-v2-1-15, March 2026, <https://datatracker.ietf.org/doc/html/draft-ietf-oauth-v2-1-15>.

[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>.

[RFC9728] Jones, M. B., Hunt, P., and A. Parecki, "OAuth 2.0 Protected Resource Metadata", RFC 9728, DOI 10.17487/RFC9728, April 2025, <https://www.rfc-editor.org/rfc/rfc9728>.

[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>.

26. 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.

[MCP-STATELESS]
Anthropic, "MCP Stateless Transport Specification (draft)", 2026.

[MS-ENTRAAGENTID]
Microsoft, "Overview of agent identities in Microsoft Entra", April 2026, <https://learn.microsoft.com/en-us/entra/agent-id/agent-identities>.

[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.

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: Implementation Considerations (Non-Normative)

AIP implementations may integrate the Enterprise IdP Federation Profile (Section 8.4.5) with enterprise identity providers by registering as an OAuth client with the enterprise IdP and using the RFC 7523 JWT Bearer grant or RFC 8693 Token Exchange grant type as described in Section 8.4.5.

For stateless or horizontally-scaled deployments, the shared token cache backend requirement is defined normatively in Section 8.2.1. Development deployments can use any single-instance cache store consistent with the definition of a non-production deployment in Section 3.

Specific implementation guides for individual products or platforms are published separately by their respective maintainers and are not part of this specification.

Author's Address

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