Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: 17 September 2026

J. Cao
Montcao
C. Arango Gutierrez
NVIDIA

16 March 2026

Agent Identity Protocol: Agentic Authentication and Authorized Policy Enforcement
draft-aip-agent-identity-protocol-00

Abstract

This document defines the Agent Identity Protocol (AIP), an open standard for verifiable identity and policy enforcement for artificial intelligence (AI) agents.

Agent Identity Protocol (AIP) addresses the problem of AI agents operating with unbounded permissions -- running as users, inheriting full API key access, and executing tool calls with no verifiable identity boundary between human and non-human actors.

The protocol is structured as two cooperating layers. Layer 1 (Identity) gives every agent a unique identifier and a key pair registered with an AIP Registry; the agent signs every outbound action with that key. Layer 2 (Enforcement) interposes a proxy between the AI client and every tool server that verifies the signature, evaluates a declarative policy, and produces an allow, deny, or hold decision before any tool is reached.

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 17 September 2026.

Table of Contents

1. Introduction

AI agents are being deployed at scale with the same credentials as the humans who operate them. When an agent calls a tool -- writing a file, querying a database, sending a request to an external API -- there is nothing in the request that distinguishes it from a direct human action. The downstream service has no way to know it is talking to an agent, which agent, who authorized it, or what it is permitted to do.

This creates compounding problems as agents become more capable and more numerous. An agent that is compromised, misbehaves, or is manipulated into acting outside its intended scope has no technical boundary stopping it from using every credential it has been given. Audit logs attribute actions to human accounts rather than to agents, making incident investigation difficult. Multi-agent systems can accumulate permissions across delegation steps without any explicit record of what was authorized.

AIP closes this gap with two layers:

Layer 1 -- Agent Identity. At provisioning time, the agent is registered with an AIP Registry. The registry assigns the agent a unique identifier (the Agent ID) and records the agent's public key alongside the identity of the accountable principal. From that point forward, the agent signs every outbound tool call with its private key. Any party that can reach the registry can verify who the agent is.

Layer 2 -- Enforcement Proxy. An AIP Proxy sits between the AI client and every tool server. It intercepts every tool call, verifies the agent's signature against the registry, evaluates the call against a simple declarative policy (the AgentPolicy), and either forwards the call, blocks it, or holds it for human approval. The tool server is never reached until all checks pass. Every decision is written to an append-only audit log.

The two layers are independent. Layer 1 can be used without Layer 2 to provide signed, attributable agent actions in existing systems. Layer 2 requires Layer 1 for identity verification but adds no new requirements on tool servers.

AIP targets the Model Context Protocol (MCP) [MCP] as its primary tool-call interface but is designed to be applicable to any structured tool invocation mechanism.

AIP does NOT:

* Define a new transport protocol.

* Replace existing service-level authentication (OAuth 2.0, mTLS). It adds an agent-identity layer on top of existing mechanisms.

* Provide content moderation or model output filtering.

2. Terminology and 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.

Agent:
: An autonomous software process that uses a large language model or other AI system to reason over tasks and invoke external tools on behalf of a principal.

Agent ID:
: A unique, stable identifier assigned to an agent by an AIP Registry at registration time. The Agent ID is a UUID v4 [RFC4122] prefixed with the registry hostname, e.g., "reg.example.com/01933f4a-9b2c-7d8e-af01-3b5c6d7e8f9a".

Agent Record:
: The data structure stored in the AIP Registry that holds an agent’s Agent ID, public key, principal identifier, and metadata. Defined in Section 5.2.

AIP Proxy (or "Enforcement Proxy"):
: A transparent forward proxy that intercepts tool calls between an AI client and tool servers. It verifies the AIP Token and evaluates the AgentPolicy before forwarding or blocking the call.

AIP Registry:
: A server that stores Agent Records and exposes an HTTP API for registration, key lookup, and revocation.

AIP Token:
: A signed JSON object attached to every tool call by the agent. It carries the agent’s Agent ID, the tool being called, a nonce, a timestamp, and an key signature. Defined in Section 5.6.

AgentPolicy:
: A YAML configuration file that declares which tools an agent is permitted to call, argument constraints, DLP rules, and HITL requirements. Defined in Section 6.2.

DLP:
: Data Loss Prevention; scanning of tool call arguments and responses for sensitive data patterns.

HITL:
: Human-in-the-Loop; a control mode in which the proxy holds a tool call and waits for explicit approval from an operator before forwarding it.

IoA:
: Internet of Agents; the network of autonomous AI agents that act across organizational and infrastructure boundaries.

MCP:
: Model Context Protocol [MCP]; a structured protocol for tool-call communication between AI clients and tool servers.

Principal:
: The human operator or organization accountable for an agent.

Tool Call:
: A structured invocation of an external capability initiated by an agent, typically carrying a tool name and arguments.

Tool Server:
: A service that exposes tools callable by agents.

3. Problem Statement

3.1. The Identity Gap

When an AI agent calls a tool, it presents credentials that belong to a human account. The tool server cannot tell whether the actor is a human or an agent, which agent it is, or what limits apply to it. This creates four concrete problems:

(a) Security -- A compromised or manipulated agent can invoke any tool the human account can reach. There is no agent-specific authorization boundary.

(b) Auditability -- Logs record actions against a human account. After an incident, investigators cannot determine which actions were taken by a human versus an agent.

(c) Compliance -- Regulations increasingly require traceability of automated decision-making. Without agent-level identity, organizations cannot satisfy these requirements.

(d) Accountability -- Billing, rate limits, and quotas are scoped

to human accounts. Agent usage cannot be isolated or attributed.

3.2. The Enforcement Gap

Even where agent behavior policies exist, they are expressed as text in model system prompts. System prompts are not tamper-evident and can be bypassed by adversarial inputs to the model. There is no infrastructure-layer enforcement point that acts independently of the model.

3.3. Scope of This Specification

AIP closes both gaps. Layer 1 gives every agent a distinct, verifiable identity independent of the human principal’s credentials. Layer 2 enforces agent-specific policy at the tool-call boundary, outside the model’s trust domain, in a way that cannot be overridden by model outputs.

4. Protocol Overview

4.1. Architecture

AIP introduces two components into the agent-to-tool-server path: an AIP Registry (Layer 1) and an AIP Proxy (Layer 2).

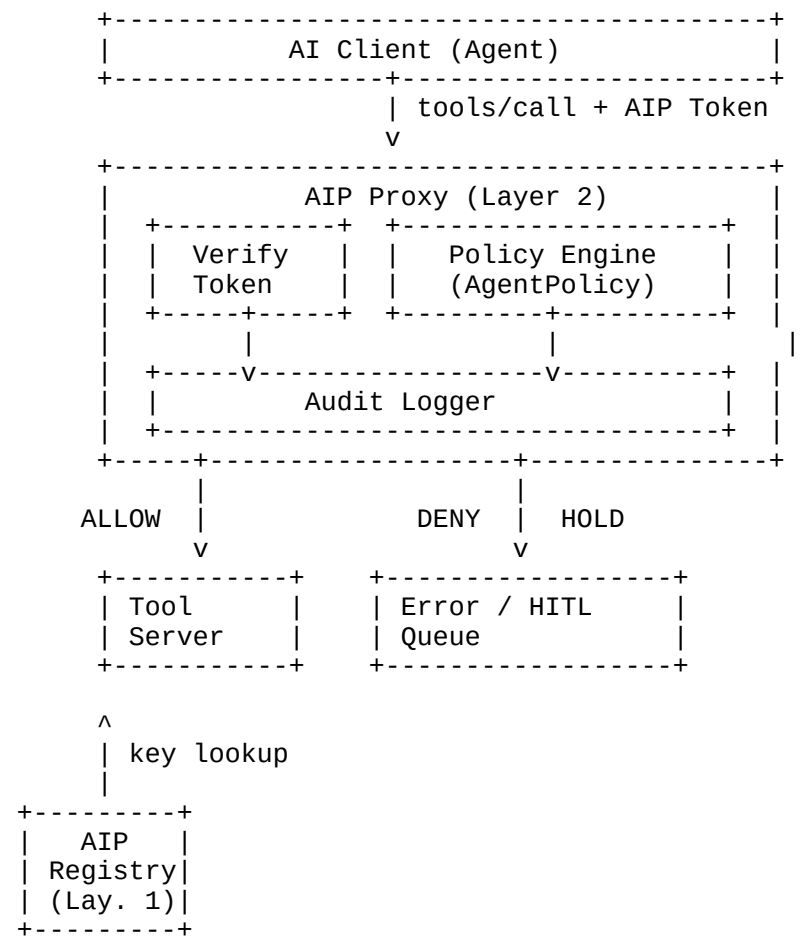

Figure 1: AIP Architecture

4.2. Call Lifecycle

The lifecycle of a single tool call under AIP is as follows:

1. Registration (once, at deploy time). The principal registers the agent with an AIP Registry. The registry assigns an Agent ID and stores the agent's public key. The agent stores its private key securely.

2. Token construction (per call). Before each tool call the agent constructs an AIP Token (Section 5.6): a small JSON object containing the Agent ID, tool name, argument hash, nonce, timestamp, and a key signature over the token.

3. Proxy intercept. The AIP Proxy receives the tool call request before it reaches the tool server.

4. Token verification (Layer 1). The proxy retrieves the agent's public key from the registry (or local cache) and verifies the signature. It also checks the nonce for replay and the timestamp for freshness.

5. Policy evaluation (Layer 2). The proxy checks the call against the AgentPolicy: is the tool on the allowlist? Do the arguments pass validation? Does the call trigger a HITL hold? Does the response contain sensitive data that must be redacted?

6. Decision. The proxy produces one of three outcomes: ALLOW -- forward the call to the tool server; DENY -- return an error to the agent, tool server not reached; HOLD -- queue the call for human approval.

7. Audit. Every decision is written to the audit log regardless of outcome.

4.3. Relationship to Existing Standards

* OAuth 2.0 [RFC6749] / OIDC [OIDC]: AIP does not replace service-level authentication. The agent still presents its OAuth token or API key to the tool server. AIP provides a separate agent identity layer that the proxy can verify independently.

* SPIFFE/SVID [SPIFFE]: In Kubernetes deployments, the AIP Proxy MAY use a SPIFFE SVID to authenticate to tool servers over mTLS, layering workload identity on top of AIP agent identity.

* JSON-RPC 2.0 [JSON-RPC]: Tool call and error messages use the JSON-RPC 2.0 wire format, compatible with MCP and similar protocols.

5. Layer 1: Agent Identity

5.1. Agent Registration

An agent is provisioned by its principal submitting a registration request to an AIP Registry. The request MUST include:

(a) The agent's public key, base64url-encoded [RFC4648];

(b) The principal identifier -- a string that uniquely identifies the accountable human or organization (e.g., an email address, an organization slug, or an OAuth subject claim); (TO BE WORKED ON FURTHER)

(c) A human-readable agent name (RECOMMENDED);

(d) An optional free-text description.

The registration request MUST be authenticated. Authentication MAY be via OAuth 2.0 bearer token, mTLS client certificate, or a pre-shared registration secret, at the registry operator's discretion.

On successful registration, the registry:

(a) Assigns a UUID v4 [RFC4122] as the agent's local identifier;

(b) Constructs the Agent ID as "<registry-host>/<uuid>"; (c) Stores the Agent Record (Section 5.2); (d) Returns the Agent ID to the principal.

The principal MUST store the Agent ID and configure the agent with both the Agent ID and its private key before deployment.

5.2. Agent Record

The Agent Record is the data structure the AIP Registry stores for each registered agent. It MUST contain:

agentId (string):
The Agent ID assigned at registration, in the form "<registry-host>/<uuid-v4>". Example: "reg.agentidentityprotocol.io/01933f4a-9b2c-7d8e-af01"

publicKey (string):
The agent's current public key, base64url-encoded.

principalId (string):
Identifier of the accountable principal.

name (string):
Human-readable agent name. Informational only; not authenticated.

description (string, optional):
Free-text description of the agent's purpose.

createdAt (string):
ISO 8601 UTC timestamp of registration.

keyHistory (array):
Append-only list of all public keys ever bound to this Agent ID. Each entry contains "publicKey", "activeFrom", and "revokedAt" (null if still active).

status (string):
One of "active" or "revoked".

Example Agent Record (JSON):

```
{
  "agentId": "reg.agentidentityprotocol.io/01933f4a-9b2c-7d8e-af01",
  "publicKey": "MCowBQYDK2VwAyEAz8vG...",
  "principalId": "acme-corp",
  "name": "ResearchAssistant-v1",
  "description": "Internal document retrieval agent",
  "createdAt": "2026-01-15T09:00:00Z",
  "keyHistory": [
    {
      "publicKey": "MCowBQYDK2VwAyEAz8vG...",
      "activeFrom": "2026-01-15T09:00:00Z",
      "revokedAt": null
    }
  ],
  "status": "active"
}
```

5.3. AIP Registry API

The AIP Registry MUST expose the following HTTP endpoints over TLS 1.3 (Section 9.4):

POST	/v1/agents	Register a new agent
GET	/v1/agents/{agentId}	Retrieve an Agent Record
PUT	/v1/agents/{agentId}/key	Rotate the agent’s public key
DELETE	/v1/agents/{agentId}	Revoke an agent
GET	/v1/revocations/stream	SSE stream for revocation events

The GET /v1/agents/{agentId} endpoint is the only endpoint that MUST be reachable by AIP Proxies at call-verification time. All other endpoints are used during provisioning and key management.

Responses from GET /v1/agents/{agentId} MUST include the full Agent Record. Proxies SHOULD cache this response for at least 30 seconds. The cache MUST be invalidated on receipt of a revocation event from the SSE stream.

5.4. Key Rotation

A principal rotates an agent's key by submitting a PUT request to /v1/agents/{agentId}/key, authenticated with the current private key. The request body MUST contain the new public key.

The registry MUST:
(a) Set "revokedAt" on the current keyHistory entry to the current timestamp;

(b)
Append a new keyHistory entry with the new public key;

(c) Update the "publicKey" field on the Agent Record; (d) Emit a rotation event on the revocations SSE stream so that proxies can invalidate their cached Agent Records immediately.

The Agent ID does NOT change on key rotation.

5.5. Agent Revocation

A principal revokes an agent by sending a DELETE request to /v1/agents/{agentId}. The registry MUST set the agent's status to "revoked" and emit a revocation event on the SSE stream.

Proxies MUST reject AIP Tokens from revoked agents with error AIP-E012. Revoked Agent Records MUST be retained in the registry for audit log verification but MUST NOT be returned as "active".

5.6. The AIP Token (Agent Attestation Token)

The AIP Token is a compact signed JSON object that the agent constructs and attaches to every outbound tool call. It is the mechanism by which the agent asserts its identity to the proxy.

5.6.1. Token Fields

aipVersion (string, REQUIRED):
Protocol version. MUST be "1" for this specification.

agentId (string, REQUIRED):
The agent's Agent ID as registered with the AIP Registry.

tool (string, REQUIRED):
The name of the tool being called, exactly as declared in the tool server's manifest.

argumentsHash (string, REQUIRED):
Lowercase hex-encoded SHA-256 hash of the canonical JSON serialization of the tool call arguments. This binds the token to the specific arguments being passed.

nonce (string, REQUIRED):
A 128-bit cryptographically random value, hex-encoded. MUST be unique per token. MUST be generated by a CSPRNG.

timestamp (string, REQUIRED):
ISO 8601 UTC timestamp of token construction.

signature (string, REQUIRED):
Base64url-encoded signature over the canonical serialization of the token (Section 5.6.2).

5.6.2. Canonical Serialization

To produce the bytes that are signed:

(a) Construct a JSON object containing all token fields except

"signature";

(b) Serialize with no insignificant whitespace;

(c) Sort object keys lexicographically; (d) Encode as UTF-8.

The signature field is then set to the base64url encoding of the signature over these bytes.

5.6.3. Example Token (before base64url encoding for transport)

```
{
  "aipVersion": "1",
  "agentId": "reg.agentidentityprotocol.io/01933f4a-9b2c-7d8e-af01",
  "tool": "read_file",
  "argumentsHash": "e3b0c44298fc1c149afb4c8996fb924...",
  "nonce": "a3f8b2c1d4e5f607a8b9c0d1e2f3a4b5",
  "timestamp": "2026-02-24T14:30:00Z",
  "signature": "TUlJQ0lqQU5CZ2txaGtpRzl3MEJB..."
}
```

5.7. Token Verification

The AIP Proxy MUST perform the following checks in order. A failure at any step MUST produce a DENY decision with the corresponding error code (Section 7.2) and an audit log entry.

Step 1 -- Presence check.
Confirm the AIP Token is present in the request. On failure: AIP-E010.

Step 2 -- Agent Record lookup.
Resolve the Agent ID from the token against the AIP Registry (or local cache). Confirm the Agent Record status is "active". On failure: AIP-E011 (unresolvable) or AIP-E012 (revoked).

Step 3 -- Signature verification.
Compute the canonical serialization (Section 5.6.2) and verify the key signature against the public key in the Agent Record. This operation MUST be performed in constant time. On failure: AIP-E013.

Step 4 -- Nonce replay check.
Check the nonce against the proxy's local nonce cache (minimum TTL: 600 seconds). If the nonce has been seen before, reject. On failure: AIP-E004.

Step 5 -- Timestamp freshness.
Reject the token if its timestamp is more than 300 seconds in the past or 30 seconds in the future relative to the proxy clock. On failure: AIP-E005.

If all five steps pass, the token is verified and the call proceeds to policy evaluation (Section 6.3).

6. Layer 2: Enforcement Proxy

6.1. Proxy Architecture

The AIP Proxy is a forward proxy interposed between the AI client and one or more tool servers. It is transparent: it presents itself to the AI client as the tool server endpoint, and to the tool server as an authorized caller.

The proxy operates in one of two modes, set per agent in the AgentPolicy:

enforce: Policy violations result in DENY responses. The tool server is never reached for blocked calls. This is the default and RECOMMENDED mode.

monitor: Policy violations are logged but calls are forwarded

```
            regardless.  Used for baselining and policy development
            before switching to enforce mode.
```

The proxy MUST maintain locally:

- The AgentPolicy for each agent it serves;
- A bounded nonce cache (TTL >= 600 seconds, LRU eviction);
- A revocation cache (refresh interval <= 60 seconds);
- An append-only audit log.

6.2. AgentPolicy

The AgentPolicy is a YAML file that declares the enforcement rules for a specific agent. One AgentPolicy file corresponds to one Agent ID. A AgentPolicy file can apply to multiple Agent IDs.

6.2.1. Schema

```
<CODE BEGINS>
   agentId: <Agent ID>
   mode: <enforce | monitor>

   tools:
     allowed:
       - <tool-name>
     rules:
       - tool: <tool-name>
         action: <allow | ask | block>
         args:
           <arg-name>:
             pattern: <PCRE regex>
             maxLength: <integer>

   dlp:
     - name: <pattern name>
       regex: <PCRE regex>
       action: <redact | block>
       scope: <request | response | both>

   hitl:
     approvers:
       - <email or identifier of approver>
     timeout_seconds: <integer, default 300>  # seconds to wait for approval
     on_timeout: <deny | allow>               # action if no response received
<CODE ENDS>
```

6.2.2. Tool Allowlist

The tools.allowed list defines every tool the agent is permitted to call. Any call to a tool not on this list MUST be denied with AIP-E001 when the proxy is in enforce mode.

6.2.3. Tool Rules

Each entry in tools.rules applies additional handling to a named tool:

allow: The tool is on the allowlist and is forwarded after argument validation. This is the default when no rule is specified.

ask: The call is held for HITL approval before forwarding, regardless of any other policy check.

block: The call is unconditionally denied. This overrides the allowlist and cannot be circumvented. Use this for tools that must never be reachable by the agent under any circumstances.

Argument rules (args) apply PCRE regex pattern matching and a maximum length check to named arguments before the call is forwarded. A violation produces AIP-E002.

6.2.4. DLP Rules

Each DLP rule specifies a regex pattern, an action, and a scope:

redact / request: Matching content in arguments is replaced with "[REDACTED:<name>]" and the call proceeds.

redact / response: Matching content in the tool response is replaced with "[REDACTED:<name>]" before the response is returned to the agent.

block / both: If a match is found anywhere in the request or response, the call or response is rejected with AIP-E008.

6.2.5. HITL Configuration

The hitl block identifies who may approve held calls, how long the proxy waits, and what to do on timeout. The default on_timeout is "deny". Setting on_timeout to "allow" SHOULD only be used for non-sensitive, low-impact tools.

6.3. Intercept Flow

The following is the normative sequence for every tool call received by the proxy.

1. Receive the tool call from the AI client.

2. Run token verification (Section 5.7).
   Any failure -> DENY with the corresponding AIP-Exxx code.

3. Check tools.allowed.
   Tool not present and mode == enforce -> DENY AIP-E001.

4. Check tools.rules for this tool.
   action == block -> DENY AIP-E003.
   action == ask   -> go to step 7 (HITL).

5. Validate arguments against any matching tools.rules.args.
   Violation -> DENY AIP-E002.

6. Run DLP scan on request arguments.
   block match -> DENY AIP-E008.
   redact match -> replace content, continue.

7. If action == ask: submit to HITL queue (Section 6.5).
   Approved -> continue to step 8.
   Denied   -> DENY AIP-E015.
   Timed out -> resolve per on_timeout.

8. ALLOW: forward the call to the tool server.

9. Receive the tool server response.

10. Run DLP scan on the response.
    block match -> return AIP-E008 to agent, suppress response.
    redact match -> replace content, continue.

11. Return the (possibly redacted) response to the agent.

12. Write audit log record (Section 6.7).

6.4. Decision Outcomes

ALLOW: The call passed all checks and was forwarded. The tool server response was returned to the agent.

DENY: The call was rejected before reaching the tool server. The proxy returns a JSON-RPC 2.0 error (Section 7.2). The tool server receives nothing.

HOLD: The call is queued. The proxy returns a pending response to the agent. The call is resolved when a human approver responds or the HITL timeout is reached.

6.5. Human-in-the-Loop (HITL)

When a call is held, the proxy MUST notify all addresses listed in hitl.approvers. The notification MUST include:

* The Agent ID and agent name;
* The tool name and (post-redaction) arguments;
* The policy rule that triggered the hold;
* A unique hold_id.

Notification delivery (email, webhook, Slack, web UI) is out of scope for this specification.

Approvers submit a decision to the proxy using the HITL API. Approval and denial endpoints are:

```
POST /v1/hitl/{hold_id}/approve
POST /v1/hitl/{hold_id}/deny
```

These endpoints MUST be authenticated. If no response is received within hitl.timeout_seconds, the proxy resolves the hold according to hitl.on_timeout and writes the outcome to the audit log.

6.6. Data Loss Prevention (DLP)

The DLP scanner applies the dlp rules from the AgentPolicy to both the inbound tool call arguments and the outbound tool server response. Rules are evaluated in the order they are listed. The first matching rule wins.

Implementations SHOULD provide a standard rule library covering common sensitive data types:

* Cloud provider credentials (AWS, GCP, Azure key patterns);
* Private key material (PEM headers);
* Common PII patterns (email, phone, SSN formats);
* Generic high-entropy secrets (long alphanumeric tokens).

6.7. Audit Logging

The proxy MUST write one log record per tool call outcome. Records MUST be appended to an append-only log and MUST NOT be modified after writing.

Log records SHOULD be hash-chained: each record includes the SHA-256 hash of the previous record, enabling tamper detection.

See Section 7.3 for the normative record format.

Log records MUST be retained for a minimum of 90 days.

7. Wire Formats

7.1. AIP-Token Header

For HTTP-based tool transports, the AIP Token is conveyed in the request header:

```
AIP-Token: <base64url-encoded UTF-8 JSON token>
```

base64url encoding is defined in [RFC4648] Section 5 (no padding).

For MCP stdio transports, the AIP Token is conveyed as a "_aip" field at the top level of the JSON-RPC request object:

```
{
  "jsonrpc": "2.0",
  "method": "tools/call",
  "id": 1,
  "params": {
    "name": "read_file",
    "arguments": { "path": "/data/report.txt" }
  },
  "_aip": {
    "aipVersion": "1",
    "agentId": "reg.agentidentityprotocol.io/01933f4a...",
    "tool": "read_file",
    "argumentsHash": "e3b0c44298...",
    "nonce": "a3f8b2c1d4e5f607...",
    "timestamp": "2026-02-24T14:30:00Z",
    "signature": "TUlJQ0lq..."
  }
}
```

7.2. Error Response Format

Error responses MUST conform to JSON-RPC 2.0 [JSON-RPC]. AIP error codes are in the range -32001 to -32099:

Code	ID	Meaning
-32001	AIP-E001	Tool not in allowlist
-32002	AIP-E002	Argument validation failed
-32003	AIP-E003	Tool unconditionally blocked
-32004	AIP-E004	Nonce replay detected
-32005	AIP-E005	Timestamp out of range
-32008	AIP-E008	DLP violation
-32010	AIP-E010	AIP Token missing
-32011	AIP-E011	Agent ID not found in registry
-32012	AIP-E012	Agent revoked
-32013	AIP-E013	Signature verification failed
-32015	AIP-E015	HITL approval denied
-32016	AIP-E016	HITL timed out
-32099	AIP-E099	Internal proxy error

Example error response:

```
{
  "jsonrpc": "2.0",
  "id": 1,
  "error": {
    "code": -32001,
    "message": "AIP-E001: tool not in allowlist",
    "data": {
      "aipCode": "AIP-E001",
      "agentId": "reg.agentidentityprotocol.io/01933f4a...",
      "tool": "exec_command"
    }
  }
}
```

7.3. Audit Log Record

Each record is a single JSON object on one line (JSONL [JSONL]):

```
{
  "v": 1,
  "ts": "<ISO 8601 UTC>",
  "eventId": "<UUID v4>",
  "prevHash": "<SHA-256 hex of previous record, or null>",
  "decision": "<ALLOW | DENY | HOLD>",
  "errorCode": "<AIP-Exxx or null>",
  "agentId": "<Agent ID>",
  "principalId": "<principal identifier>",
  "tool": "<tool name>",
  "argumentsHash": "<SHA-256 hex>",
  "policyName": "<AgentPolicy agentId value>",
  "verificationStep": "<1-5 or null if passed>",
  "dlp": [
    { "rule": "<rule name>", "scope": "<request|response>",
      "action": "<redacted|blocked>" }
  ],
  "holdId": "<UUID or null>",
  "proxyVersion": "<semver>"
}
```

8. Deployment Topologies

8.1. Localhost Proxy

The localhost proxy is the simplest deployment: a single binary running on the developer's machine alongside the AI client. It binds to 127.0.0.1:8787 (configurable), supports HTTP and MCP stdio interception, and uses a local SQLite database for the nonce cache, revocation cache, and audit log. Policy is loaded from a YAML file at ~/.aip/policy.yaml.

This mode is suitable for individual developers and is the recommended starting point for adopting AIP. A Go proxy implementation is available at the working group's GitHub repository.

8.2. Kubernetes Sidecar

For production deployments, the AIP Proxy runs as a sidecar container in the same pod as the agent container. Outbound tool-server traffic is redirected through the proxy on port 15001 via iptables REDIRECT rules. Policy is loaded from a Kubernetes ConfigMap or Secret. Storage uses Redis for the distributed nonce cache and a PersistentVolumeClaim for the audit log. A Prometheus metrics endpoint is exposed at /metrics.

Exported metrics include: aip_calls_total, aip_denials_total, aip_holds_total, and aip_verification_latency_seconds.

8.3. Enterprise Federation

In enterprise environments the AIP Proxy integrates with existing identity infrastructure:

* OIDC mapping: The registry maintains a table mapping Agent IDs to OIDC subject claims. When a tool server requires an OIDC token, the proxy can present one on the agent's behalf after AIP verification passes, without the agent holding OIDC credentials directly.

* SPIFFE/mTLS: The proxy is issued a SPIFFE SVID and uses it to establish mTLS connections to tool servers, providing transport-layer mutual authentication in addition to AIP application-layer identity.

* Central policy management: AgentPolicy files are managed via a policy API with versioning, rollback, and change auditing, rather than local YAML files.

9. Security Considerations

9.1. Cryptographic Algorithm

AIP is built with the idea of using Ed25519 [RFC8032] for all signatures. Ed25519 was chosen because it is fast (sign and verify in under 1ms), produces short keys and signatures (32 and 64 bytes respectively), is deterministic (no per-signature randomness required), and has broad library support across all major languages and platforms. However other cryptographic algorithms should be supported.

All Ed25519 operations MUST use a constant-time implementation to prevent timing side-channels (Section 9.3).

9.2. Prompt Injection Resistance

The principal threat model for AIP at Layer 2 is the prompt injection attack: malicious content in the agent's context causes it to attempt tool calls outside its intended scope.

The allowlist in the AgentPolicy prevents the agent from reaching tools it was not provisioned for, regardless of what the model produces. The "block" action provides an unconditional deny for specified tools that cannot be overridden by any model output, injected prompt, or delegation claim. The proxy operates entirely outside the model's trust boundary; the model cannot influence the proxy's decisions.

AIP does not protect against:

- A compromised agent runtime that routes calls around the proxy;
- Tool servers that accept calls from sources other than the proxy.

9.3. Transport Security

All communication between AIP Proxies and the AIP Registry MUST use TLS 1.3 [RFC8446]. Server certificates MUST be validated against the system trust store. Certificate pinning is RECOMMENDED for registry connections in production deployments.

Proxy-to-tool-server connections MUST use TLS 1.2 [RFC5246] or later. TLS 1.3 is RECOMMENDED.

9.4. Private Key Storage

Agent private keys MUST be stored in a secure key store. In order of preference:

1. Hardware Security Module (HSM) or Trusted Platform Module (TPM);
2. OS keychain (macOS Keychain, Windows DPAPI, Linux Secret Service); 3. Encrypted file with passphrase derived using Argon2id [RFC9106].

Private keys MUST NOT be stored in environment variables, plaintext configuration files, or source code repositories.

9.5. Registry Trust

A proxy MUST be configured with an explicit list of trusted registry hostnames and the TLS certificate fingerprint (or CA) for each. Agent Records from registries not on this list MUST be rejected.

9.6. Nonce Cache Sizing

The nonce cache must be large enough to hold all unique nonces generated within the TTL window (600 seconds). Implementations MUST use a bounded cache with LRU eviction and MUST NOT silently drop old nonces without ensuring they are outside the TTL window.

9.7. Revocation Latency

The proxy’s revocation cache has a maximum refresh interval of 60 seconds (Section 6.1). In the worst case, a revoked agent can continue making calls for up to 60 seconds after revocation. Deployments with stricter requirements SHOULD subscribe to the registry’s SSE revocation stream (Section 5.3) and invalidate the cache on receipt of a revocation event.

9.8. Denial-of-Service

The proxy adds a small amount of latency to every tool call (one cache lookup and one key verification). To prevent the proxy from becoming an availability bottleneck:

* Agent Record lookups MUST be served from the local cache except on cache miss or invalidation;

* The proxy SHOULD enforce a per-agent call rate limit to prevent a runaway agent from flooding the registry with cache-miss lookups.

10. Privacy Considerations

10.1. Agent Record Visibility

Agent Records are visible to any party that can query the registry. Principals SHOULD use non-identifying agent names for agents that handle sensitive workloads.

10.2. Audit Log Sensitivity

Audit logs contain Agent IDs, tool names, and argument hashes. While argument values are not logged in plaintext, the combination of Agent ID and tool name may itself be sensitive in some contexts. Audit logs MUST be access-controlled appropriately.

10.3. DLP Redaction

When DLP redaction is applied to a response, the proxy modifies the data the agent receives. Operators MUST ensure that redaction does not cause the agent to produce incorrect or harmful downstream actions due to missing context.

10.4. Registry Data Retention

Registry operators MUST publish a data retention policy. Agent Records for revoked agents SHOULD be pseudonymized after the minimum retention period required for audit log verification.

11. IANA Considerations

11.1. HTTP Header Field

This document defines the "AIP-Token" HTTP header field. Registration is requested in the "Permanent Message Header Field Names" registry per [RFC3864]. The header field name is "AIP-Token", applicable protocol is "http", status is "standard", the author/ change controller is the IETF, and the specification document is this document (Section 7.1).

11.2. Media Type

The media type "application/aip+json" is requested for AIP Tokens serialized as standalone JSON documents. The type name is "application", the subtype name is "aip+json", the required parameter is "version" (value "1"), encoding considerations are UTF-8, and security considerations are described in Section 9.

12. References

12.1. Normative References

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <https://www.rfc-editor.org/info/rfc2119>.

[RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <https://www.rfc-editor.org/info/rfc8174>.

[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/info/rfc8032>.

[RFC8446] Rescorla, E., "The Transport Layer Security (TLS) Protocol Version 1.3", RFC 8446, DOI 10.17487/RFC8446, August 2018, <https://www.rfc-editor.org/info/rfc8446>.

[RFC5246] Dierks, T. and E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.2", RFC 5246, DOI 10.17487/RFC5246, August 2008, <https://www.rfc-editor.org/info/rfc5246>.

[RFC4648] Josefsson, S., "The Base16, Base32, and Base64 Data Encodings", RFC 4648, DOI 10.17487/RFC4648, October 2006, <https://www.rfc-editor.org/info/rfc4648>.

[RFC4122] Leach, P., Mealling, M., and R. Salz, "A Universally Unique IDentifier (UUID) URN Namespace", RFC 4122, DOI 10.17487/RFC4122, July 2005, <https://www.rfc-editor.org/info/rfc4122>.

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

[RFC9106] Biryukov, A., Dinu, D., Khovratovich, D., and S. Josefsson, "Argon2 Memory-Hard Function for Password Hashing and Proof-of-Work Applications", RFC 9106, DOI 10.17487/RFC9106, September 2021, <https://www.rfc-editor.org/info/rfc9106>.

[JSON-RPC] JSON-RPC Working Group, "JSON-RPC 2.0 Specification", January 2013.

[RFC3864] Klyne, G., Nottingham, M., and J. Mogul, "Registration Procedures for Message Header Fields", BCP 90, RFC 3864, DOI 10.17487/RFC3864, September 2004, <https://www.rfc-editor.org/info/rfc3864>.

12.2. Informative References

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

[OIDC] Sakimura, N. et al., "OpenID Connect Core 1.0", November 2014.

[SPIFFE] CNCF SPIFFE Project, "SPIFFE: Secure Production Identity Framework for Everyone", 2024.

[JSONL] Granger, N., "JSON Lines", 2014.

Appendix A. Example AgentPolicy

A complete AgentPolicy for a research assistant agent.

```
   <CODE BEGINS>
      agentId: reg.agentidentityprotocol.io/01933f4a-9b2c-7d8e-af01
      mode: enforce

      tools:
        - allowed:
          - read_file
          - list_directory
          - web_search
          - git_status
          - write_file
          - create_issue
        - rules:
          - tool: write_file
            action: ask
          - tool: exec_command
            action: block
          - tool: delete_file
            action: block
          - tool: create_issue
            action: ask

      dlp:
        - name: aws-access-key
          regex: "AKIA[A-Z0-9]{16}"
          action: block
          scope: both
        - name: private-key-pem
          regex: "-----BEGIN [A-Z ]* PRIVATE KEY-----"
          action: block
          scope: both
        - name: generic-token
          regex: "[a-zA-Z0-9_\\-]{40,}"
          action: redact
          scope: response

      hitl:
        - approvers:
          - ops@acme.example
        - timeout_seconds: 300  # wait 5 minutes
        - on_timeout: deny
   <CODE ENDS>
```

Appendix B. Example AIP Token

An AIP Token for a read_file call, shown before base64url encoding for transport in the AIP-Token header.

```
{
  "aipVersion": "1",
  "agentId": "reg.agentidentityprotocol.io/01933f4a-9b2c-7d8e-af01",
  "tool": "read_file",
  "argumentsHash": "e3b0c44298fc1c149afb4c8996fb92427ae41e4649b934ca495991b7852b855",
  "nonce": "a3f8b2c1d4e5f607a8b9c0d1e2f3a4b5",
  "timestamp": "2026-02-24T14:30:00Z",
  "signature": "TUlJQ0lqQU5CZ2txaGtpRzl3MEJBUUVGQUFOQ0E..."
}
```


Appendix C. Error Code Reference

AIP-E001 Tool not in allowlist

The requested tool is not listed in tools.allowed and the proxy is in enforce mode.

AIP-E002 Argument validation failed

A tool argument failed a pattern or maxLength check defined in tools.rules.args.

AIP-E003 Tool unconditionally blocked

The tool has action: block in tools.rules.

AIP-E004 Nonce replay

The nonce in the AIP Token was already seen within the 600-second cache window.

AIP-E005 Timestamp out of range

The token timestamp is more than 300 seconds old or more than 30 seconds in the future.

AIP-E008 DLP violation

A DLP rule with action: block matched content in the request or response.

AIP-E010 AIP Token missing

The tool call arrived with no AIP-Token header or _aip field.

AIP-E011 Agent ID not found

The agentId in the token could not be resolved at the registry.

AIP-E012 Agent revoked

The Agent Record for this agentId has status: revoked.

AIP-E013 Signature verification failed

The key signature did not verify against the public key in the Agent Record.

AIP-E015 HITL approval denied

A human approver explicitly denied the held call.

AIP-E016 HITL timed out

The HITL hold expired and on_timeout is set to deny.

AIP-E099 Internal proxy error

An unexpected error occurred in the proxy. See proxy logs for details.

Authors' Addresses

James Cao
Montcao
Email: james@montcao.com
URI: https://agentidentityprotocol.io

Carlos Eduardo Arango Gutierrez
NVIDIA
Email: eduardoa@nvidia.com
URI: https://agentidentityprotocol.io