

OAuth Working Group  
Internet-Draft  
Intended status: Standards Track  
Expires: 11 August 2026

A. Cruz  
Independent  
7 February 2026

Agent Authorization Profile (AAP) for OAuth 2.0  
draft-aap-oauth-profile-01

## Abstract

This document defines the Agent Authorization Profile (AAP), an authorization profile for OAuth 2.0 and JWT designed for autonomous AI agents. AAP extends existing standards with structured claims and validation rules so that systems can reason about agent identity, task context, operational constraints, delegation chains, and human oversight requirements. It does not introduce a new protocol; it specifies how to use OAuth 2.0, JWT, Token Exchange, and proof-of-possession mechanisms in agent-to-API (M2M) scenarios with context-aware, auditable authorization.

## 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 11 August 2026.

## Copyright Notice

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

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

extracted from this document must include Revised BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Revised BSD License.

## Table of Contents

1. Introduction . . . . .	4
1.1. Problem Statement . . . . .	5
1.2. Goals . . . . .	5
1.3. Non-Goals . . . . .	5
2. Terminology . . . . .	5
3. Conventions Used in This Document . . . . .	6
4. Overview and Relationship to Existing Standards . . . . .	7
5. JWT Claim Schema (AAP Profile) . . . . .	8
5.1. Claim Semantics . . . . .	8
5.2. Structured Sections (Claim Names) . . . . .	8
5.3. Example Claim Structures . . . . .	9
5.3.1. String Length Limits . . . . .	11
5.4. Complete Example Payload . . . . .	12
5.4.1. Not Before (nbf) Claim . . . . .	14
5.5. Action Name Grammar (ABNF) . . . . .	14
5.6. Standard Constraint Types and Semantics . . . . .	15
5.6.1. Rate Limiting Constraints . . . . .	16
5.6.2. Domain and Network Constraints . . . . .	17
5.6.3. Time-Based Constraints . . . . .	17
5.6.4. Delegation Constraints . . . . .	18
5.6.5. Data and Security Constraints . . . . .	18
5.7. Delegation Chain Semantics . . . . .	19
6. Threat Model Summary . . . . .	21
6.1. Agent Impersonation . . . . .	21
6.2. Capability Escalation . . . . .	21
6.3. Purpose Drift . . . . .	22
6.4. Malicious or Excessive Delegation . . . . .	22
6.5. Large-Scale Automated Misuse . . . . .	22
6.6. Prompt / Data Injection . . . . .	22
6.7. Lack of Traceability . . . . .	23
6.8. Use Outside Intended Environment . . . . .	23
6.9. Summary . . . . .	23
7. Resource Server Validation Rules . . . . .	23
7.1. Standard Token Validation . . . . .	24
7.2. Proof of Possession Validation . . . . .	25
7.3. Agent Identity Validation . . . . .	25
7.4. Task Binding Validation . . . . .	25
7.5. Capability Enforcement . . . . .	26
7.6. Oversight Requirement Enforcement . . . . .	26
7.7. Delegation Chain Validation . . . . .	26
7.8. Contextual Restrictions Enforcement . . . . .	27
7.9. Audit and Trace Propagation . . . . .	27

7.10. Failure Handling . . . . .	27
8. Authorization Server Requirements . . . . .	29
8.1. Authentication . . . . .	29
8.2. Token Binding . . . . .	29
8.3. Capabilities and Constraints . . . . .	29
8.4. Token Lifetime . . . . .	30
8.5. Token Exchange . . . . .	30
8.6. Revocation . . . . .	30
8.7. Proof-of-Possession . . . . .	30
8.8. Audit . . . . .	30
9. Example High-Level Flow . . . . .	30
10. Extensibility . . . . .	31
10.1. Backwards Compatibility with OAuth 2.0 . . . . .	31
10.2. Migration Path from Scopes to Capabilities . . . . .	31
11. Conformance Requirements . . . . .	32
12. Security Considerations . . . . .	32
12.1. Cryptographic Algorithms and Key Management . . . . .	32
12.2. Proof-of-Possession Requirements . . . . .	34
12.3. Token Lifetime and Revocation . . . . .	36
12.4. Constraint Enforcement . . . . .	37
12.5. Delegation Security . . . . .	37
12.6. Human Oversight Enforcement . . . . .	38
12.7. Authorization Server Security . . . . .	38
12.8. Resource Server Security . . . . .	39
12.9. Token Caching Security . . . . .	40
12.10. Key Compromise Incident Response . . . . .	40
12.11. Denial-of-Service Prevention . . . . .	41
12.12. Additional Security Considerations . . . . .	41
13. Privacy Considerations . . . . .	42
13.1. Personal Data in AAP Tokens . . . . .	42
13.2. Data Minimization Principles . . . . .	42
13.3. Retention and Lifecycle . . . . .	43
13.4. Cross-Domain Correlation . . . . .	44
13.5. Privacy-Preserving Error Messages . . . . .	45
13.6. Anonymization and Pseudonymization . . . . .	46
13.7. Consent and Transparency . . . . .	46
13.8. Third-Party Tool Privacy . . . . .	47
13.9. Privacy by Design Recommendations . . . . .	48
13.10. Regulatory Compliance Guidance . . . . .	49
14. IANA Considerations . . . . .	50
14.1. JWT Claims Registration . . . . .	50
14.2. OAuth Error Code Registration . . . . .	50
15. Implementation Status . . . . .	52
15.1. Reference Implementation . . . . .	52
15.2. Test Vectors . . . . .	52
16. Related Work and Comparison . . . . .	53
16.1. OAuth 2.0 Scopes . . . . .	53
16.2. Fine-Grained Authorization Systems (Zanzibar, ReBAC) . .	54

16.3.	Cloud Identity and Access Management . . . . .	54
16.4.	Service Mesh Authorization (Istio, Linkerd) . . . . .	55
16.5.	Capability-Based Security . . . . .	56
16.6.	OpenID Connect and Step-Up Authentication . . . . .	56
16.7.	OAuth 2.0 Rich Authorization Requests (RAR) . . . . .	57
16.8.	Summary: Where AAP Fits . . . . .	57
17.	References . . . . .	58
17.1.	Normative References . . . . .	58
17.2.	Informative References . . . . .	58
Appendix A.	Complete JSON Schema Reference . . . . .	59
A.1.	Schema Files . . . . .	59
A.2.	Using the Schemas . . . . .	61
A.3.	Schema Updates . . . . .	63
Appendix B.	Token Exchange Flow Example . . . . .	63
B.1.	Scenario . . . . .	63
B.2.	Step 1: Agent Obtains Original Token . . . . .	64
B.3.	Step 2: Agent Exchanges Token for Tool-Specific Token . . . . .	65
B.4.	Key Changes in Derived Token . . . . .	69
B.5.	Privilege Reduction Summary . . . . .	69
B.6.	Further Delegation (Depth=2) . . . . .	70
Appendix C.	AAP-Specific Error Codes . . . . .	70
C.1.	Error Code Reference . . . . .	70
C.2.	Error Response Format . . . . .	71
C.3.	Error Code Usage Examples . . . . .	71
C.4.	Error Handling Guidance for Clients . . . . .	72
Appendix D.	Conformance Checklist . . . . .	72
D.1.	Authorization Server Conformance Checklist . . . . .	72
D.2.	Resource Server Conformance Checklist . . . . .	74
D.3.	Optional Features (RECOMMENDED) . . . . .	76
D.4.	Testing Conformance . . . . .	77
Appendix E.	Implementation Examples . . . . .	77
E.1.	Example Policy Configuration . . . . .	77
E.2.	Example Token Validation Code (Pseudocode) . . . . .	78
Appendix F.	Test Vectors . . . . .	80
F.1.	Valid Token -- Basic Research Agent . . . . .	80
F.2.	Invalid Token -- Excessive Delegation . . . . .	81
F.3.	Edge Case -- Clock Skew Tolerance . . . . .	82
Author's Address	. . . . .	83

## 1. Introduction

The Agent Authorization Profile (AAP) is an authorization profile built on OAuth 2.0 [RFC6749] and JWT [RFC7519], designed to support secure, auditable, and context-aware authorization for autonomous AI agents. AAP extends existing standards with structured claims and validation rules that allow systems to reason about agent identity, task context, operational constraints, delegation chains, and human oversight requirements.

### 1.1. Problem Statement

Traditional OAuth-based systems were designed primarily for user-to-application-to-API interactions. Autonomous AI agents introduce different characteristics: actions can be autonomous and high frequency; human approval may not be present at execution time; risk depends on task context; and execution may be delegated across multiple tools or agents. Existing scope-based models are not expressive enough to represent these requirements safely and transparently.

### 1.2. Goals

- \* Provide explicit and verifiable identity for AI agents.
- \* Support capability-based authorization with enforceable constraints.
- \* Bind access tokens to specific tasks and declared purposes.
- \* Enable auditable delegation across agents and tools.
- \* Support the expression of human oversight requirements.
- \* Remain compatible with OAuth 2.0, JWT, Token Exchange, and proof-of-possession mechanisms.

### 1.3. Non-Goals

- \* Defining internal AI model behavior.
- \* Judging the correctness or ethics of agent decisions.
- \* Replacing organizational security or compliance frameworks.
- \* Standardizing log storage formats or SIEM integrations.

## 2. Terminology

Term	Definition
*Agent*	Autonomous software entity (e.g. LLM, bot) that acts as an OAuth client and performs actions on behalf of an operator.
*Authorization	Server that issues access tokens

Server (AS)*	in accordance with OAuth 2.0 and applies AAP policies.
*Capability*	Permitted action (e.g. action) together with its restrictions (constraints).
*Delegation*	Transfer of a subset of privileges to another entity (tool or sub-agent) via token exchange or other mechanism.
*Operator*	Organization or human role that registers and authorizes the agent.
*Proof-of-possession (PoP)*	Mechanism by which the client demonstrates possession of a key (e.g. DPOP, mTLS) when using the token.
*Resource Server (RS)*	Server that protects resources and validates AAP tokens before allowing access.
*Task*	Unit of work to which the token is bound (identifier, purpose, sensitivity, etc.).

Table 1

**\*AAP token:\*** An access token issued by an Authorization Server that conforms to this profile and contains AAP claims (e.g. agent, task, capabilities).

**\*Claim:\*** A name/value pair in a JWT [RFC7519] payload. AAP defines additional claim names and structures for agent identity, task binding, capabilities, oversight, delegation, context, and audit.

### 3. Conventions Used in This Document

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. For conformance: "MUST" and "SHALL" indicate mandatory requirements; "SHOULD" and "RECOMMENDED" indicate

recommended but not mandatory behavior; "MAY" and "OPTIONAL" indicate optional behavior.

#### 4. Overview and Relationship to Existing Standards

AAP operates within a standard OAuth architecture consisting of an Authorization Server (AS), Resource Servers (RS), and clients. In AAP, the client is an autonomous AI agent. Token issuance follows OAuth 2.0; typically the Client Credentials Grant [RFC6749] Section 4.4 is used for agent-to-API (M2M) flows. Client (agent) authentication MAY use standard client authentication (client secret, mTLS, etc.) or assertions (e.g. JWT-based client authentication) as per the deployment profile. When the agent is a workload identified by SPIFFE [SPIFFE], the AS MAY accept SVIDs or derived tokens as part of client authentication; AAP does not define a new flow but MAY integrate with SPIFFE. Tokens issued by the AS include additional structured claims that Resource Servers MUST evaluate before allowing operations.

AAP does not introduce a new identity or protocol scheme; it reuses existing standards and adds a layer of claims and validation rules.

- \* \*OAuth 2.0\* -- AAP uses the standard OAuth flow (AS, RS, client); the client is the agent; tokens are JWTs with additional AAP claims.
- \* \*OpenID Connect\* -- Agent identity MAY be based on OIDC sub and iss; AAP adds agent-, task-, and capability-specific claims.
- \* \*mTLS [RFC8705]\* -- RECOMMENDED for proof-of-possession and for agent authentication toward the AS and RS.
- \* \*DPoP [RFC9449]\* -- Alternative to mTLS for proof-of-possession; RECOMMENDED when mTLS is not feasible.
- \* \*SPIFFE\* -- OPTIONAL; the agent identifier (e.g. in agent) MAY be a SPIFFE ID (spiffe://trust-domain/...) when the deployment uses SPIFFE/SPIRE for workload identity.
- \* \*Token Exchange [RFC8693]\* -- Used for delegation and for privilege reduction on token re-issuance; the act (actor) claim MAY be used in the delegation chain.

## 5. JWT Claim Schema (AAP Profile)

AAP tokens extend standard JWT claims [RFC7519] with the following structured sections. The normative claim names are: agent, task, capabilities, oversight, delegation, context, audit. These names are registered in the IANA "JSON Web Token Claims" registry (see Section 14).

**\*Formal Schema:** Complete JSON Schema definitions for all AAP claims are provided in Appendix A and in the /schemas directory of the reference implementation. Implementations SHOULD validate tokens against these schemas to ensure conformance.

### 5.1. Claim Semantics

- \* **\*Agent identity\*** -- MAY be expressed via OIDC sub and iss, or via the agent claim. The agent claim MAY contain, among other fields, a SPIFFE ID (spiffe://trust-domain/...) when the deployment uses SPIFFE/SPIRE for workload identity.
- \* **\*Delegation\*** -- The delegation chain MAY use the standard act (actor) claim from [RFC8693]. Optionally, delegation MAY carry additional metadata (e.g. depth, origin) when more than act is required.
- \* **\*Oversight\*** -- Human oversight requirements are expressed as policy metadata in oversight (e.g. requires\_approval\_for for certain capability types, or max\_autonomous\_scope). AAP only carries the intent; enforcement is at the Resource Server or orchestrator (e.g. OIDC step-up with acr\_values or an external approval API), and is out of scope for this profile.
- \* **\*Audit\*** -- Trace identifiers in audit SHOULD be compatible with existing trace context propagation (e.g. W3C Trace Context, OpenTelemetry [OpenTelemetry]) so that logs can be correlated with distributed traces without defining a new audit schema.

### 5.2. Structured Sections (Claim Names)

- \* agent
- \* task
- \* capabilities
- \* oversight
- \* delegation (and/or act per [RFC8693])

- \* context
- \* audit

### 5.3. Example Claim Structures

The following examples illustrate concrete claim shapes using the normative claim names defined above. Standard JWT claims (iss, sub, aud, exp, iat, jti) are assumed.

**\*Agent identity\*** -- Identifies the autonomous agent and its execution context.

```
{
  "agent": {
    "id": "agent-researcher-01",
    "type": "llm-autonomous",
    "operator": "org:blogcorp",
    "model": {
      "provider": "provider-name",
      "id": "model-id",
      "version": "model-version"
    },
    "runtime": {
      "environment": "kubernetes",
      "attested": true
    }
  }
}
```

**\*Task binding\*** -- Binds the token to a specific task and purpose.

```
{
  "task": {
    "id": "task-id",
    "purpose": "research_and_draft_article",
    "topic": "example-topic",
    "data_sensitivity": "public",
    "created_by": "user-id"
  }
}
```

**\*Capabilities\*** -- Authorized actions and constraints.

```
{
  "capabilities": [
    {
      "action": "search.web",
      "constraints": {
        "domains_allowed": ["example.org"],
        "max_requests_per_hour": 50
      }
    },
    {
      "action": "cms.create_draft",
      "constraints": {
        "status": "draft_only"
      }
    }
  ]
}
```

#### \*Token Size Considerations\*

AAP tokens with multiple capabilities and constraints can produce large JWTs. Many HTTP servers impose limits on the Authorization header (nginx default: 8KB, Apache: 8KB, AWS ALB: 16KB). Implementations SHOULD monitor token size and:

- \* SHOULD warn when a serialized token exceeds 4KB
- \* SHOULD use token introspection [RFC7662] or reference tokens when capabilities exceed 10 entries
- \* MAY define a capabilities\_ref claim containing a URI that references an external capability set, reducing token size while maintaining capability semantics

\*Oversight\* -- Human approval requirements for certain actions.

```
{
  "oversight": {
    "requires_human_approval_for": [
      "cms.publish",
      "execute.payment"
    ],
    "approval_reference": "policy-id"
  }
}
```

\*Delegation\* -- Depth and chain of delegation.

```
{
  "delegation": {
    "depth": 0,
    "max_depth": 2,
    "chain": ["agent:agent-researcher-01"]
  }
}
```

*\*Context\** -- Network, time, and geo restrictions.

```
{
  "context": {
    "network_zone": "public-internet-only",
    "time_window": {
      "start": "ISO-8601",
      "end": "ISO-8601"
    },
    "geo_restriction": "none"
  }
}
```

*\*Audit\** -- Trace and session identifiers for logging.

```
{
  "audit": {
    "log_level": "full",
    "trace_id": "trace-id",
    "session_id": "session-id"
  }
}
```

#### 5.3.1. String Length Limits

To prevent abuse and ensure interoperability, implementations SHOULD enforce the following string length limits:

Claim Field	Min Length	Max Length
agent.id	1	128
agent.type	1	64
agent.operator	1	256
task.id	1	128
task.purpose	1	256
capabilities[].action	1	128
delegation.chain[] (each entry)	1	128
audit.trace_id	1	256

Table 2

Authorization Servers MUST reject token requests containing claim values exceeding these limits. Resource Servers SHOULD reject tokens with values exceeding these limits.

#### 5.4. Complete Example Payload

The following is a single JSON object representing the decoded payload of an AAP access token (claims only; signature and encoding are per RFC 7519). Standard JWT claims and all AAP claims are combined with consistent values (e.g. same agent.id and task.id referenced in audit).

```
{
  "iss": "https://as.example.com",
  "sub": "agent-researcher-01",
  "aud": "https://api.example.com",
  "exp": 1704067200,
  "iat": 1704063600,
  "jti": "token-unique-id-123",
  "agent": {
    "id": "agent-researcher-01",
    "type": "llm-autonomous",
    "operator": "org:blogcorp",
    "model": {
      "provider": "provider-name",
      "id": "model-id",

```

```
    "version": "model-version"
  },
  "runtime": {
    "environment": "kubernetes",
    "attested": true
  }
},
"task": {
  "id": "task-id",
  "purpose": "research_and_draft_article",
  "topic": "example-topic",
  "data_sensitivity": "public",
  "created_by": "user-id"
},
"capabilities": [
  {
    "action": "search.web",
    "constraints": {
      "domains_allowed": ["example.org"],
      "max_requests_per_hour": 50
    }
  },
  {
    "action": "cms.create_draft",
    "constraints": {
      "status": "draft_only"
    }
  }
],
"oversight": {
  "requires_human_approval_for": [
    "cms.publish",
    "execute.payment"
  ],
  "approval_reference": "policy-id"
},
"delegation": {
  "depth": 0,
  "max_depth": 2,
  "chain": ["agent:agent-researcher-01"]
},
"context": {
  "network_zone": "public-internet-only",
  "time_window": {
    "start": "2024-01-01T00:00:00Z",
    "end": "2024-01-01T23:59:59Z"
  },
  "geo_restriction": "none"
}
```

```
    },
    "audit": {
      "log_level": "full",
      "trace_id": "trace-id",
      "session_id": "session-id"
    }
  }
```

#### 5.4.1. Not Before (nbf) Claim

The nbf (Not Before) claim [RFC7519] Section 4.1.5 MAY be included in AAP tokens. This is useful for: - Scheduled tasks that should not start before a given time - Tokens pre-issued for future use (e.g., batch operations starting at a specific hour)

If present, Resource Servers MUST reject tokens presented before the nbf time minus clock skew tolerance. The same clock skew tolerance applied to exp validation (RECOMMENDED: 5 minutes) SHOULD be applied to nbf validation.

#### 5.5. Action Name Grammar (ABNF)

Action names in the action field of capabilities MUST conform to the following ABNF grammar [RFC5234]:

```
action-name = component *( "." component )
component = ALPHA *( ALPHA / DIGIT / "-" / "_" )
```

Where: - ALPHA is any ASCII alphabetic character (a-z, A-Z) - DIGIT is any ASCII digit (0-9) - Component names MUST start with an alphabetic character - Component names MAY contain hyphens and underscores after the first character - Action names are formed by concatenating components with dots (.)

Examples of valid action names: - search.web - cms.create\_draft - cms.publish - execute.payment - api.v2.users.read - data-pipeline.transform\_records

Examples of invalid action names: - search..web (double dot) - .search.web (starts with dot) - search.web. (ends with dot) - 9api.read (component starts with digit) - search.web\* (wildcard not allowed)

**\*Matching Semantics:**

Resource Servers MUST perform **\*exact string matching\*** on action names. Wildcard matching (e.g., cms.\* matching cms.publish) is NOT part of this specification but MAY be defined in future extensions.

Action names are *\*case-sensitive\**. `search.Web` and `search.web` are different actions.

Versioning MAY be expressed through namespace components: -  
`api.v1.search.web` - `api.v2.search.web`

## 5.6. Standard Constraint Types and Semantics

This section defines the semantics of standard constraint types used within capability constraints. Resource Servers MUST enforce these constraints according to the semantics defined here.

### *\*Constraint Enforcement Semantics\**

When multiple capabilities match the same action: - *\*OR semantics\**:  
If ANY capability grants the action, the request is authorized (subject to that capability's constraints) - Resource Server evaluates capabilities in order and uses the first match

When multiple constraints exist within a single capability: - *\*AND semantics\**: ALL constraints MUST be satisfied for the action to be authorized - If any constraint fails, the entire request MUST be denied

### *\*Empty Constraints\**

When constraints is an empty object `{}` or is absent from a capability, NO restrictions are applied to that capability. The capability grants the action without rate limits, domain restrictions, or time windows. An empty constraints object and a missing constraints key are semantically equivalent.

### *\*Constraint Precedence Rules\**

When multiple constraints of the same type exist (e.g., from capability-level and global policy), the following precedence rules apply: - *\*Numeric constraints\** (rate limits, size limits): Use the MORE restrictive (lower) value - *\*Allow-lists\** (`domains_allowed`, `allowed_methods`, `allowed_regions`): Use intersection - *\*Block-lists\** (`domains_blocked`): Use union - *\*Time windows\**: Use intersection of time ranges (narrower window)

## 5.6.1. Rate Limiting Constraints

Constraint Name	Type	Semantics	Example
max_requests_per_hour	integer	Fixed hourly quota. Window resets at minute 0 of each hour (clock hour). Failed requests count toward quota. Retries count as new requests.	50
max_requests_per_minute	integer	Sliding 60-second window from current request time backwards. Resource Server MUST track request timestamps.	10
max_requests_per_day	integer	Fixed daily quota. Window resets at 00:00:00 UTC. Failed requests count toward quota.	1000

Table 3

**\*Implementation Notes:** - Rate limits are per token (identified by jti claim) - Resource Servers SHOULD use distributed rate limiting for multi-instance deployments - On quota exceeded: Resource Server MUST return HTTP 429 with `aap_constraint_violation` error and a `Retry-After` header - Rate limit state SHOULD be cleared when token expires - Rate limit enforcement MUST use strict UTC timestamps. Clock skew tolerance applies only to token expiration (`exp` and `nbf` claims), NOT to rate limit windows

5.6.2. Domain and Network Constraints

domains\_allowed (array of strings): DNS suffix matching (rightmost matching). subdomain.example.org matches example.org in allowlist. Resource Server MUST extract domain from request target URL and validate. Example: ["example.org", "trusted.example"]

domains\_blocked (array of strings): Blocklist takes precedence over allowlist. If both are present, blocked domains MUST be checked first. Example: ["malicious.example"]

ip\_ranges\_allowed (array of CIDR strings): IP ranges in CIDR notation. Resource Server validates destination IP of request. Example: ["192.168.1.0/24"]

- \*Domain Matching Algorithm:\*
- 1. Extract domain from request target URL
  - 2. If domains\_blocked is present and domain matches any blocked entry: DENY
  - 3. If domains\_allowed is present:
    - a. Check if domain is exact match or has allowed domain as suffix
    - b. If match found: ALLOW (proceed to other constraints)
    - c. If no match: DENY
  - 4. If neither constraint present: proceed to other constraints

5.6.3. Time-Based Constraints

Constraint Name	Type	Semantics	Example
time_window.start	ISO 8601 string	Request timestamp MUST be after or equal to this time (inclusive). Resource Server uses its own clock with max 5-minute skew tolerance.	"2024-01-01T00:00:00Z"
time_window.end	ISO 8601	Request timestamp	"2024-12-31T23:59:59Z"

	string	MUST be before this time (exclusive).	
--	--------	---------------------------------------	--

Table 4

**\*Clock Skew Handling:\*** - Resource Server clock is authoritative - Resource Server MAY tolerate up to 5 minutes of clock skew - If request timestamp is outside window beyond skew tolerance: DENY with aap\_constraint\_violation

#### 5.6.4. Delegation Constraints

Constraint Name	Type	Semantics	Example
max_depth	integer (0-10)	Maximum delegation depth for this capability. 0 means no delegation allowed. Resource Server MUST validate delegation.depth <= max_depth.	2

Table 5

#### 5.6.5. Data and Security Constraints

Constraint Name	Type	Semantics	Example
max_response_size	integer	Maximum response size in bytes. Resource Server SHOULD enforce during response streaming.	10485760 (10MB)
max_request_size	integer	Maximum request payload size in bytes. Resource Server MUST validate before processing.	1048576 (1MB)

data_classification_max	enum string	Maximum data classification level accessible. Values: public, internal, confidential, restricted. Resource Server enforces based on resource classification.	"internal"
allowed_methods	array of strings	HTTP methods allowed. Resource Server MUST validate request method against this list.	["GET", "POST"]
allowed_regions	array of ISO 3166-1 alpha-2 codes	Geographic regions where requests are allowed. Resource Server validates based on request origin or target resource location.	["US", "CA", "GB"]

Table 6

### 5.7. Delegation Chain Semantics

The delegation claim tracks authorization delegation across agents and tools using OAuth Token Exchange [RFC8693].

\*Delegation Depth Calculation:\*

depth = 0: Original agent token (no delegation)

depth = 1: Token obtained via Token Exchange from depth=0 token

depth = n: Token obtained via Token Exchange from depth=n-1 token

**\*Authorization Server Requirements:\*** - AS MUST verify that `depth < max_depth` in the parent token BEFORE issuing a derived token with `depth = parent_depth + 1`. That is, the AS MUST reject the Token Exchange request if the parent token's `delegation.depth >= delegation.max_depth`. - AS MUST increment `delegation.depth` by 1 on each Token Exchange - AS MUST append the current agent/tool identifier to `delegation.chain` array - AS MUST copy and preserve `delegation.chain` from parent token - AS MUST NOT issue token if resulting depth would exceed capability's `max_depth` constraint - AS MUST NOT issue token if resulting depth exceeds `delegation.max_depth` claim

**\*Resource Server Requirements:\*** - RS MUST reject requests if `delegation.depth > delegation.max_depth` - RS MUST validate `delegation.depth` against capability-specific `max_depth` constraints - RS MUST validate that `delegation.chain` length equals `delegation.depth + 1`

**\*Delegation Chain Format:\***

```
{
  "delegation": {
    "depth": 2,
    "max_depth": 3,
    "chain": [
      "spiffe://trust.example.com/agent/researcher-01",
      "spiffe://trust.example.com/tool/web-scraper",
      "https://as.example.com/agents/translator"
    ],
    "parent_jti": "parent-token-jti-value"
  }
}
```

**\*Privilege Reduction Requirements:\***

When issuing a derived token via Token Exchange, the Authorization Server MUST reduce privileges by one or more of: - Removing capabilities (subset of parent capabilities) - Adding stricter constraints (lower rate limits, narrower domain lists) - Reducing token lifetime (shorter exp time) - Reducing `max_depth` (limit further delegation)

The Authorization Server MUST NOT grant capabilities not present in the parent token.

**\*Token Refresh Strategy:\***

AAP tokens, especially delegated tokens with reduced lifetimes, may require refresh before expiration: - Agents SHOULD refresh tokens at 80% of the token's lifetime (e.g., at 48 minutes for a 60-minute token) - Refresh MUST use the same grant type that obtained the original token (Client Credentials Grant for original tokens) - Delegated tokens (obtained via Token Exchange) MUST NOT be refreshed; a new Token Exchange MUST be performed against the parent token - If the parent token has expired, the agent MUST obtain a new original token before performing Token Exchange

**\*Preventing Confused Deputy Attacks:\***

To prevent confused deputy attacks where a delegated token is replayed: - Each token MUST have a unique jti (JWT ID) - Delegation chain MUST be immutable (copied, never modified) - Token Exchange MUST record parent\_jti linking to parent token - Authorization Server MAY implement token family revocation (revoking parent revokes all descendants)

## 6. Threat Model Summary

AAP assumes environments where autonomous AI agents can access APIs, perform chained actions, and operate for extended periods without direct human intervention. The following threats are in scope; for each, agent-specific risks and AAP mitigations are noted.

### 6.1. Agent Impersonation

**\*Threat:\*** An attacker obtains agent credentials or steals a token and acts as an authorized agent.

**\*Agent-specific risk:\*** An agent may have broad permissions and act many times per minute, amplifying impact.

**\*Mitigations:\*** Short-lived tokens; Proof-of-Possession (mTLS or DPoP); attested workload identity when possible; strong agent identity claims (agent.id, agent.model, runtime.attested).

### 6.2. Capability Escalation

**\*Threat:\*** The agent attempts actions beyond what is authorized (e.g. publish instead of create draft).

**\*Agent-specific risk:\*** Agents may generate new strategies or calls not anticipated by the developer.

**\*Mitigations:** Structured capabilities with constraints (not broad scopes); mandatory validation of action + constraints by the Resource Server; task-bound tokens (task.purpose); explicit separation between automatic and human-supervised actions.

### 6.3. Purpose Drift

**\*Threat:** A token issued for one task is reused for another (e.g. token for "public health research" used for "extracting sensitive data").

**\*Mitigations:** Mandatory task claim with purpose; Resource Servers verify consistency between declared purpose and requested operation; short time windows; reject requests that do not match the declared context.

### 6.4. Malicious or Excessive Delegation

**\*Threat:** An agent delegates to tools or sub-agents with more privileges than intended.

**\*Agent-specific risk:** Agent ecosystems are often modular and chained.

**\*Mitigations:** OAuth Token Exchange with privilege reduction; delegation.depth and delegation.chain claims; maximum depth limit (max\_depth); prohibition of delegation for certain critical capabilities.

### 6.5. Large-Scale Automated Misuse

**\*Threat:** An authorized agent performs valid actions at harmful volume (spam, abusive scraping, mass resource creation).

**\*Mitigations:** Quantitative constraints in capabilities (max\_requests\_per\_hour, etc.); enforced by the Resource Server; monitoring and rapid token revocation; mandatory audit with traceability by task and agent.

### 6.6. Prompt / Data Injection

**\*Threat:** A third party manipulates external data to induce the agent to use its permissions in unwanted ways.

**\*Note:** AAP does not control the AI model but can limit impact.

\*Mitigations:\* Tokens bound to specific purpose; restriction of domains, action types, and volumes; separation of read vs. write vs. execute capabilities; human oversight required for high-impact actions.

#### 6.7. Lack of Traceability

\*Threat:\* Inability to reconstruct which agent did which action under which authorization.

\*Agent-specific risk:\* Decisions can be complex and chained.

\*Mitigations:\* Audit claims (audit.trace\_id, task.id); mandatory propagation of trace identifiers; inclusion of delegation chain in derived tokens.

#### 6.8. Use Outside Intended Environment

\*Threat:\* A valid token is used from a network, region, or environment other than the one authorized.

\*Mitigations:\* Context claims (context.network\_zone, time windows); additional validation by the Resource Server; combination with traditional network controls.

#### 6.9. Summary

AAP assumes that agents are potentially powerful and highly automated; risk depends not only on who accesses but on purpose, limits, and delegation chain; authorization MUST be contextual, restricted, and auditable. AAP extends OAuth from a broad-permission model toward verifiable operational contracts between organizations, agents, and services.

### 7. Resource Server Validation Rules

This section defines the validation rules that a Resource Server (RS) MUST apply before accepting a request authenticated with an AAP access token. These rules extend standard OAuth 2.x token validation with agent-specific, task-bound, and capability-aware checks.

\*Recommended Validation Order:\*

Resource Servers SHOULD validate tokens in the following order to fail fast on inexpensive checks:

1. Extract Bearer token [RFC6750] from Authorization header

2. Decode JWT header (reject unknown algorithms)
3. Verify signature using trusted AS public key
4. Check exp (with clock skew tolerance); check nbf if present
5. Check aud matches Resource Server identifier
6. Check iss is a trusted Authorization Server
7. Validate required AAP claims are present (agent, task, capabilities)
8. Validate agent identity (id, type, operator)
9. Validate task binding (id, purpose)
10. Validate delegation chain (depth, chain length) if delegation present
11. Match capability to requested action
12. Enforce capability constraints (rate limits, domains, time windows, etc.)

#### 7.1. Standard Token Validation

- \* The RS MUST verify the token signature using trusted Authorization Server keys.
- \* The RS MUST verify the token has not expired (exp claim) and is within acceptable clock skew.
- \* The RS MUST verify the audience (aud) matches the Resource Server.
- \* The RS MUST verify the issuer (iss) is trusted.
- \* The RS MUST verify the token has not been revoked if a revocation or introspection mechanism is in place ([RFC7009], [RFC7662] when introspection is used).

#### \*JWKS Caching and Refresh:\*

Resource Servers that obtain AS public keys via JWKS endpoint MUST implement the following caching behavior: - SHOULD cache JWKS responses for at least 5 minutes to avoid excessive requests - MUST refresh JWKS when encountering an unknown kid (key ID) in a token header - SHOULD implement exponential backoff for JWKS refresh

failures (starting at 1 second, maximum 5 minutes) - MUST NOT cache JWKS responses for more than 24 hours - SHOULD respect HTTP cache control headers (Cache-Control, Expires) from the JWKS endpoint response

## 7.2. Proof of Possession Validation

For AAP tokens, proof-of-possession is RECOMMENDED; for high-risk profiles it SHOULD be REQUIRED. Implementations MUST support at least one of: DPoP [RFC9449] or mTLS client authentication [RFC8705]. If mTLS or DPoP is used, the RS MUST validate that the requester demonstrates possession of the key bound to the token. Bearer-only usage is not sufficient for high-risk agent capabilities.

## 7.3. Agent Identity Validation

- \* The RS MUST ensure the agent claim is present and well-formed.
- \* The RS MUST verify agent.id is recognized or allowed by local policy.
- \* If present, the RS MUST evaluate agent.runtime.attested according to local trust requirements.
- \* If model information is included, the RS MUST ensure the model identifier is not on a deny list.

## 7.4. Task Binding Validation

- \* The RS MUST ensure the task claim is present for agent-issued tokens.
- \* The RS MUST verify the current request is consistent with task.purpose.
- \* The RS MUST reject requests that clearly fall outside the declared purpose or data sensitivity.
- \* The RS MAY enforce that the token is used only within the declared time window.

### \*Task Consistency Levels:\*

Implementations SHOULD apply task consistency validation at the following levels:

1. \*Structural (MUST):\* task.id and task.purpose are present and non-empty strings.

2. *\*Temporal (SHOULD):\** If `task.created_at` is present, it MUST NOT be in the future (beyond clock skew tolerance). If `task.expires_at` is present, the RS MUST reject tokens for tasks past their expiration.
3. *\*Semantic (MAY):\** The RS MAY implement application-specific logic to validate that the requested action is plausibly related to `task.purpose` (e.g., a research purpose should not trigger `data.delete` actions). This level is implementation-specific and not standardized by this profile.

#### 7.5. Capability Enforcement

The Resource Server MUST treat the capabilities claim as the authoritative source of permitted actions.

- \* The RS MUST match the requested operation to a `capability.action` entry.
- \* The RS MUST enforce all constraints associated with the matching capability.
- \* The RS MUST deny the request if no matching capability is found.
- \* The RS MUST apply quantitative limits such as rate limits or volume caps defined in constraints.

#### 7.6. Oversight Requirement Enforcement

- \* If the requested action appears in `oversight.requires_human_approval_for`, the RS MUST NOT complete the action automatically.
- \* The RS SHOULD return a response indicating that human approval is required.
- \* The RS MAY provide a reference to the approval workflow indicated by `approval_reference`.

#### 7.7. Delegation Chain Validation

- \* If a delegation claim is present, the RS MUST verify that `delegation.depth` does not exceed local policy limits.
- \* The RS MUST inspect `delegation.chain` to understand upstream actors.

- \* The RS SHOULD apply stricter policy if the chain includes untrusted or unknown actors.
- \* The RS MUST ensure delegated tokens do not contain broader capabilities than the original agent token.

#### 7.8. Contextual Restrictions Enforcement

- \* If context.network\_zone is present, the RS MUST verify the request originates from an allowed environment when technically feasible.
- \* The RS MUST enforce time window constraints if context.time\_window is present.
- \* The RS MUST apply additional checks for geo or network restrictions when provided.

#### 7.9. Audit and Trace Propagation

- \* The RS MUST extract audit.trace\_id and propagate it to internal logs.
- \* The RS MUST log agent.id, task.id, action performed, and authorization decision outcome.
- \* The RS MUST ensure logs are protected against tampering according to organizational policy.

#### 7.10. Failure Handling

- \* If any mandatory validation step fails, the RS MUST deny the request.
- \* Error responses SHOULD avoid leaking sensitive authorization details.
- \* Repeated violations MAY trigger rate limiting or temporary blocking of the agent identity.

\*Error responses:\* On validation failure, the RS MUST use the following HTTP status codes:

HTTP Status	Error Code	Condition
401 Unauthorized	invalid_token	Signature invalid, token expired, audience mismatch,

		issuer untrusted, missing required claims
403 Forbidden	aap_invalid_capability	No matching capability for requested action
403 Forbidden	aap_domain_not_allowed	Domain constraint violation
403 Forbidden	aap_capability_expired	Time window constraint violation
403 Forbidden	aap_approval_required	Human oversight required
403 Forbidden	aap_excessive_delegation	Delegation depth exceeded
403 Forbidden	aap_invalid_delegation_chain	Malformed delegation chain
403 Forbidden	aap_task_mismatch	Request inconsistent with task purpose
403 Forbidden	aap_agent_not_recognized	Agent identity not recognized by policy
403 Forbidden	aap_invalid_context	Context restriction violated
413 Payload Too Large	request_too_large	Payload exceeds max_request_size constraint
429 Too Many Requests	aap_constraint_violation	Rate limit constraint exceeded

Table 7

**\*Note:** Rate limit violations MUST return HTTP 429 (Too Many Requests) with a Retry-After header, NOT HTTP 403. Other constraint violations (domain, time window) MUST return HTTP 403.

The response body SHOULD follow a structure such as error and optional error\_description (e.g. RFC 6749 / RFC 6750 style) without revealing internal authorization details. Avoid including in error\_description the exact authorization rule that failed, so as not to leak information to an attacker. See Appendix C for the complete error code reference.

## 8. Authorization Server Requirements

This section defines the requirements that an Authorization Server (AS) MUST satisfy to issue AAP access tokens. These extend standard OAuth 2.0 token issuance with agent-specific binding, capabilities, and audit.

### 8.1. Authentication

- \* Strongly authenticate agents before issuing tokens (e.g. Client Credentials Grant per RFC 6749 Section 4.4).
- \* Support client authentication via client secret, mTLS, or assertions (e.g. JWT-based client authentication) as per the deployment profile.
- \* Optionally integrate with SPIFFE SVIDs when the agent is a workload identified by SPIFFE [SPIFFE].

### 8.2. Token Binding

- \* Bind issued tokens to a specific task and declared purpose.
- \* Include task identifier and purpose in the token so that Resource Servers can verify request consistency.

### 8.3. Capabilities and Constraints

- \* Embed in the token only capabilities and constraints authorized by AS policy for the agent and task.
- \* Do not issue tokens with broader capabilities than the operator has authorized for the given task.

#### 8.4. Token Lifetime

- \* Limit token lifetimes according to risk level (e.g. shorter for high-impact or high-frequency use).
- \* Apply policy-based rules for expiration and acceptable clock skew.

#### 8.5. Token Exchange

- \* Support token exchange per [RFC8693] when delegation or privilege reduction is required.
- \* Enforce reduction of privileges when mapping `subject_token` to `issued_token` (e.g. subset of capabilities); mapping rules are AS policy.
- \* Use the `act` (actor) claim when building delegation chains as per RFC 8693.

#### 8.6. Revocation

- \* Support rapid revocation per [RFC7009] (Token Revocation) and/or [RFC7662] (Introspection) as appropriate to the deployment.
- \* Ensure Resource Servers can check revocation status when policy requires it (e.g. via introspection endpoint or revocation list).

#### 8.7. Proof-of-Possession

- \* For AAP tokens, support at least one of DPoP [RFC9449] or mTLS [RFC8705] when the profile requires proof-of-possession.
- \* Issue tokens that indicate PoP binding when the client has demonstrated key possession during the token request.

#### 8.8. Audit

- \* Record issuance events (agent identity, task, capabilities granted, timestamp) for audit and traceability.
- \* Support correlation with trace identifiers included in the token (audit) where applicable.

### 9. Example High-Level Flow

1. An operator defines allowed capabilities and policies for an agent.

2. The agent authenticates with the Authorization Server (e.g. Client Credentials Grant).
3. The agent requests a task-bound access token; optional token exchange (RFC 8693) for delegation or privilege reduction.
4. The Authorization Server issues a JWT containing AAP claims.
5. The agent calls Resource Servers using the token (with DPoP or mTLS when proof-of-possession is required).
6. Resource Servers validate the token and enforce capabilities and constraints.
7. All actions are logged with trace identifiers for audit.

## 10. Extensibility

AAP is designed as a profile and allows additional claims or constraints to be defined by industry groups or organizations, provided they do not weaken core validation requirements.

### 10.1. Backwards Compatibility with OAuth 2.0

AAP is designed as a profile on top of OAuth 2.0 and JWT. The following backwards compatibility properties hold:

- \* AAP tokens are valid JWTs and MAY be presented to non-AAP Resource Servers. Non-AAP RSes will ignore AAP claims and process only standard JWT claims (iss, sub, aud, exp, iat).
- \* Non-AAP RSes MUST NOT be treated as enforcing AAP constraints. The scope claim MAY be included alongside capabilities for backwards compatibility with legacy systems.
- \* AAP-aware Resource Servers MUST validate AAP claims when present. If a token contains both scope and capabilities, the RS MUST use capabilities as the authoritative source of permissions and MUST ignore scope for AAP-governed actions.

### 10.2. Migration Path from Scopes to Capabilities

Organizations adopting AAP from traditional OAuth 2.0 SHOULD follow a phased transition:

1. \*Phase 1 (Dual Issuance):\* Authorization Server includes both scope and capabilities in tokens. Legacy RSes use scope; AAP-aware RSes use capabilities.

2. \*Phase 2 (Capabilities Preferred):\* Resource Servers validate capabilities when present, fall back to scope only for legacy tokens.
3. \*Phase 3 (Capabilities Required):\* Remove scope from tokens; require capabilities for all agent-issued tokens.

During migration, agents SHOULD request tokens with both scope and capabilities parameters to ensure compatibility with both legacy and AAP-aware Resource Servers.

## 11. Conformance Requirements

A conforming implementation satisfies the requirements of this profile for its role (Authorization Server or Resource Server).

- \* \*Authorization Server:\* A conforming AS MUST satisfy all requirements in Section 8. It MUST issue tokens that include the AAP claims required by the deployment profile and MUST support proof-of-possession (DPoP and/or mTLS) when the profile requires it. It SHOULD support token exchange [RFC8693] and revocation [RFC7009] [RFC7662] as appropriate.
- \* \*Resource Server:\* A conforming RS MUST apply all rules in Section 7 (standard token validation, proof-of-possession when required, agent identity, task binding, capability enforcement, oversight, delegation chain, contextual restrictions, audit and trace propagation, failure handling). It MUST deny requests when any mandatory validation step fails and MUST NOT leak sensitive authorization details in error responses.
- \* A conforming implementation MAY support additional claims or options provided they do not weaken the requirements above.

## 12. Security Considerations

This section addresses security considerations specific to AAP beyond those covered in OAuth 2.0 Security Best Current Practice and related standards.

### 12.1. Cryptographic Algorithms and Key Management

\*Token Signing (REQUIRED):\*

Authorization Servers MUST sign AAP tokens using asymmetric cryptography. The following algorithms are REQUIRED:

- \* **\*ES256\*** (ECDSA with P-256 curve and SHA-256): RECOMMENDED for new deployments
  - Provides 128-bit security level
  - Smaller signatures than RSA
  - Fast signature verification
- \* **\*RS256\*** (RSA with SHA-256): REQUIRED for backward compatibility
  - Minimum 2048-bit RSA keys
  - 3072-bit or 4096-bit RSA keys RECOMMENDED for long-lived keys or high-security environments

Authorization Servers MUST NOT use symmetric signing algorithms (HS256, HS384, HS512) for AAP tokens. Symmetric algorithms require sharing the signing key between AS and RS, which violates trust boundaries: any RS that validates tokens could forge new ones. Only asymmetric algorithms (ES256, RS256, EdDSA) are permitted.

**\*Proof-of-Possession Algorithms:\***

When using DPoP [RFC9449]: - Agents MUST use ES256 for DPoP proof generation - RS256 MAY be supported for legacy clients - DPoP proof lifetime SHOULD be short (maximum 60 seconds)

When using mTLS [RFC8705]: - TLS 1.3 REQUIRED; TLS 1.2 MAY be supported with restricted cipher suites - Cipher suites: ECDHE\_ECDSA or ECDHE\_RSA with AES\_GCM - Client certificates MUST be validated against trusted CA or certificate pinning

**\*Key Sizes:\***

Key Type	Minimum	Recommended	Security Level
RSA	2048-bit	3072-bit	~112-bit / ~128-bit
ECDSA	P-256	P-384	128-bit / 192-bit
Symmetric (client secrets)	128-bit entropy	256-bit entropy	128-bit / 256-bit

Table 8

**\*Key Rotation:\***

- \* AS signing keys SHOULD be rotated every 90 days
- \* Previous keys MUST be retained for token validation during overlap period (RECOMMENDED: 24 hours after rotation)
- \* Resource Servers MUST support validation with multiple concurrent AS public keys
- \* Key rotation MUST be coordinated via JWKS (JSON Web Key Set) endpoint [RFC7517]

**\*Key Storage:\***

- \* Authorization Server private signing keys MUST be stored in Hardware Security Modules (HSM) or equivalent secure key management services for production deployments
- \* Agent client credentials (secrets, private keys) SHOULD be stored in secure vaults (e.g., HashiCorp Vault, cloud KMS)
- \* Keys MUST NOT be logged, included in error messages, or transmitted over insecure channels

**12.2. Proof-of-Possession Requirements****\*Risk Assessment:\***

Bearer tokens (tokens without proof-of-possession) present elevated risk for autonomous agents due to: - High request rates amplify impact of stolen tokens - Agents may operate unattended for extended periods - Token theft from agent memory or logs enables replay attacks

**\*Deployment Profiles:\***

AAP defines three security profiles for proof-of-possession:

Profile	PoP Requirement	Use Case
*Strict*	REQUIRED (mTLS or DPOP)	Production systems, confidential data, high-risk capabilities
*Standard*	RECOMMENDED	Development, internal tools, low-risk capabilities
*Legacy*	OPTIONAL	Migration scenarios, backward compatibility

Table 9

**\*Profile Selection Criteria:\***

Proof-of-Possession SHOULD be REQUIRED when any of the following apply:

- Capability includes data\_classification\_max >= "confidential"
- Capability includes write, delete, or execute actions
- oversight.level >= "approval"
- Token lifetime > 1 hour
- Agent accesses resources in different trust domains

**\*Token Binding (cnf claim):\***

When PoP is used, tokens MUST include the cnf (confirmation) claim [RFC7800]:

For DPoP:

```
{
  "cnf": {
    "jkt": "0ZcOCORZNYy-DWpqq30jZyJGHTN0d2Hg1BV3uiguA4I"
  }
}
```

For mTLS:

```
{
  "cnf": {
    "x5t#S256": "bwck0esc3ACC3DB2Y5_lESsXE8o9ltc05089jdN-dg"
  }
}
```

Resource Servers MUST validate the proof matches the cnf claim before accepting the token.

## 12.3. Token Lifetime and Revocation

**\*Token Lifetime Guidelines:\***

Capability Risk Level	Recommended Lifetime	Maximum Lifetime
Read-only, public data	60 minutes	120 minutes
Read-only, internal data	30 minutes	60 minutes
Write, low-risk	15 minutes	30 minutes
Write, high-risk	5 minutes	15 minutes
Execute, delete	5 minutes	10 minutes

Table 10

Token lifetime SHOULD be reduced for:

- Derived tokens (via Token Exchange): 50% of parent lifetime
- Tokens with high delegation depth: Reduce by 25% per depth level
- Tokens without proof-of-possession: 50% of normal lifetime

**\*Revocation Requirements:\***

Authorization Servers MUST support token revocation via one or both of:

- **\*Revocation endpoint\*** [RFC7009]: Agent or operator can revoke token
- **\*Token introspection\*** [RFC7662]: RS queries AS for token validity

**\*Rapid Revocation:\***

For the purposes of this specification, "rapid revocation" is defined as:

- Maximum 60 seconds from revocation request to enforcement by all Resource Servers
- Authorization Server MUST distribute revocation events to Resource Servers within 30 seconds
- Resource Server MUST apply revocation updates within 30 seconds of receipt

**\*Revocation Distribution:\***

For multi-RS deployments, AS SHOULD use:

- Push-based revocation (AS pushes to RS) rather than pull-based (RS polls AS)
- Revocation event stream (e.g., Server-Sent Events, WebSockets, message queue)
- Fallback to token introspection if revocation list fails to propagate

#### \*Token Family Revocation:\*

When a token is revoked: - Authorization Server MAY revoke all descendant tokens (derived via Token Exchange) - This is achieved by tracking parent\_jti linkage - Token family revocation enhances security but requires AS to maintain token graph

### 12.4. Constraint Enforcement

#### \*Server-Side Enforcement (REQUIRED):\*

All capability constraints MUST be enforced by the Resource Server, NOT trusted to agent logic. Agents are potentially adversarial; they may attempt to bypass constraints.

#### \*Rate Limiting in Distributed Systems:\*

For Resource Servers deployed across multiple instances: - Rate limits MUST be enforced consistently across all instances - Use distributed rate limiting (e.g., Redis, shared counter service) - Accept eventual consistency with conservative limits (deny when uncertain)

#### \*Constraint Validation Failures:\*

When a constraint is violated, Resource Server MUST: - Return HTTP 403 (Forbidden) or 429 (Too Many Requests) as appropriate - Include error code aap\_constraint\_violation in response - Log violation event with agent.id, task.id, violated constraint - NOT return details of constraint values in error response (privacy)

### 12.5. Delegation Security

#### \*Privilege Reduction (REQUIRED):\*

When issuing derived tokens via Token Exchange [RFC8693], the Authorization Server MUST reduce privileges by one or more of: -  
\*Capability removal\*: Subset of parent capabilities only -  
\*Constraint tightening\*: Lower rate limits, narrower domain lists, shorter time windows -  
\*Lifetime reduction\*: Shorter exp time than parent token -  
\*Depth limit reduction\*: Lower max\_depth to limit further delegation

The Authorization Server MUST NOT grant capabilities not present in the parent token. "Privilege escalation via delegation" MUST be prevented.

#### \*Delegation Depth Enforcement:\*

Both Authorization Server and Resource Server MUST enforce delegation depth limits: - AS MUST NOT issue token if resulting delegation.depth would exceed delegation.max\_depth - RS MUST reject requests if delegation.depth > delegation.max\_depth - Defense-in-depth: Both layers validate to prevent bypass

**\*Confused Deputy Prevention:\***

To prevent confused deputy attacks where delegation chains are replayed: - Each token MUST have unique jti (JWT ID) - Derived tokens MUST include delegation.parent\_jti linking to parent token - AS MUST validate parent token exists and is not expired/revoked before issuing derived token - Delegation chain MUST be immutable (copied and appended, never modified by client)

## 12.6. Human Oversight Enforcement

**\*Approval Workflow Security:\***

When oversight.requires\_human\_approval\_for includes an action: - Resource Server MUST NOT execute action automatically - RS MUST return HTTP 403 with error code aap\_approval\_required - Response SHOULD include approval\_reference URL for requesting approval - Approval mechanism is out of scope for AAP but may use OIDC step-up authentication, external workflow systems, etc.

**\*Approval Bypass Prevention:\***

- \* Approval requirement is in token (signed by AS), not in request (untrusted)
- \* Agent cannot bypass approval by modifying request headers
- \* Resource Server MUST validate oversight claim before execution

## 12.7. Authorization Server Security

**\*AS as Critical Component:\***

The Authorization Server is a critical security component. Compromise of the AS enables arbitrary token minting.

**\*AS Hardening Requirements:\***

- \* Minimal software installation (reduce attack surface)
- \* Regular security patching (OS, libraries, dependencies)

- \* Network isolation (AS should not be internet-accessible if possible; use API gateway)
- \* Access control (admin access requires MFA, logging)
- \* Monitoring (detect anomalous token issuance patterns)

**\*Token Issuance Monitoring:\***

Authorization Server SHOULD monitor and alert on: - Sudden increase in token issuance rate (possible compromise) - Tokens with unusual capability combinations (possible policy bypass) - Tokens issued to unknown agents (possible credential theft) - Issuance outside normal business hours (possible unauthorized access)

## 12.8. Resource Server Security

**\*Input Validation:\***

Resource Servers MUST validate: - Token signature with trusted AS public key - Token expiration (exp) with acceptable clock skew (RECOMMENDED: <=5 minutes) - Audience (aud) matches Resource Server identifier - Agent identity (agent.id) is recognized or allowed by policy - All constraints in matching capability

**\*Error Handling (Privacy-Preserving):\***

Resource Servers MUST NOT leak authorization details in error responses:

**\*Bad\* (leaks constraint values):**

```
{
  "error": "aap_constraint_violation",
  "error_description": "Rate limit exceeded"
}
```

**\*Good\* (privacy-preserving):**

```
{
  "error": "aap_constraint_violation",
  "error_description": "Request violates capability constraints"
}
```

Detailed violation information SHOULD be logged server-side for audit, not returned to client.

### 12.9. Token Caching Security

Agents and intermediaries MAY cache tokens for reuse within their lifetime. The following security guidance applies:

- \* Tokens SHOULD be stored in memory only and MUST NOT be persisted to disk in plaintext
- \* Cached tokens SHOULD be evicted before expiration (at 80% of remaining lifetime) to avoid using tokens that may expire during a request
- \* Token caches MUST be cleared on process termination
- \* Delegated tokens MUST NOT be cached longer than their parent token's expiration
- \* Agents MUST NOT share cached tokens across different task contexts

### 12.10. Key Compromise Incident Response

When an AS signing key is compromised or suspected of compromise, the following incident response procedure SHOULD be followed:

1. *\*Immediate rotation:* Generate and deploy a new key pair. Update the JWKS endpoint to include the new key.
2. *\*JWKS update:* Publish updated JWKS with the new key. The compromised kid SHOULD be removed from the JWKS endpoint or marked with a revocation indicator.
3. *\*RS propagation:* All Resource Servers MUST refresh their JWKS cache within the cache TTL. RSes SHOULD treat tokens signed with the compromised kid as invalid.
4. *\*Token invalidation:* All outstanding tokens signed with the compromised key SHOULD be considered invalid. If token introspection is in use, the AS MUST return `active: false` for tokens signed with the compromised key.
5. *\*Agent notification:* Agents MUST request new tokens after key rotation. The AS SHOULD reject new token requests until the rotation is complete.

Organizations SHOULD have a documented key compromise response plan and SHOULD test it periodically.

### 12.11. Denial-of-Service Prevention

AAP token validation is computationally more expensive than standard OAuth due to larger JWT payloads, multiple validation steps (agent, task, delegation, capabilities), and stateful constraint enforcement (rate limiting).

Resource Servers SHOULD implement the following DoS mitigations: - Reject tokens larger than 16KB before parsing (pre-validation size limit) - Rate-limit the number of token validation attempts per source IP - Cache validation results for recently-seen JTIs (with TTL equal to remaining token lifetime) to avoid re-validating the same token - Implement circuit breakers for JWKS endpoint requests to prevent cascading failures when the AS is unavailable

Authorization Servers SHOULD: - Rate-limit the token endpoint itself (per client, per IP) - Monitor and alert on anomalous token issuance patterns (sudden volume spikes, unusual capability combinations)

### 12.12. Additional Security Considerations

#### \*Token Logging:\*

- \* Tokens MUST NOT be logged in plaintext in application logs
- \* Use token redaction in logging libraries (replace with hash or truncated value)
- \* If logging is necessary for debugging, use separate secure audit log with strict access control

#### \*Clock Skew:\*

- \* Resource Servers SHOULD tolerate up to 5 minutes of clock skew for exp and nbf validation
- \* Skew tolerance MUST NOT exceed 5 minutes (to limit window for expired token use)
- \* Organizations SHOULD use NTP or equivalent for clock synchronization

#### \*Compliance Considerations:\*

AAP enables compliance with regulations requiring: - Explicit authorization (GDPR Article 6) - Purpose limitation (GDPR Article 5) - Audit trails (SOC 2, ISO 27001, HIPAA) - Access controls (PCI-DSS, FedRAMP)

However, AAP alone is not sufficient for compliance; organizational policies, procedures, and controls are also required.

### 13. Privacy Considerations

AAP tokens and audit logs may contain information that could identify agents, organizations, individuals, or sensitive operational details. This section provides guidance on privacy protection in AAP implementations.

#### 13.1. Personal Data in AAP Tokens

AAP tokens MAY contain personal data under GDPR, CCPA, and similar privacy regulations:

**\*Potentially Personal Claims:\*** - `agent.operator`: Organization identifier (may be personal if sole proprietor or individual developer) - `task.created_by`: User ID who initiated the task - `task.purpose`: May contain user-identifying details if not carefully crafted - `audit.trace_id`: Potentially correlatable across requests (tracking) - `delegation.chain`: Reveals agent interaction patterns and organizational structure - `context.location`: Geographic location information

**\*Data Controller and Processor:\*** - Authorization Server operator is typically the data controller for token claims - Resource Server operators are data processors when validating tokens - Delegation across organizations may create complex controller/processor relationships

#### 13.2. Data Minimization Principles

Implementations SHOULD apply the principle of data minimization (GDPR Article 5(1)(c)):

**\*Guideline: Include Only What Is Necessary\***

Claim	Privacy-Preserving Approach	Privacy-Risky Approach
agent.id	Use pseudonymous ID: urn:uuid:550e8400-...	Use descriptive name: agent-for-alice@example.com
task.id	Use UUID: 550e8400-e29b-41d4-...	Use descriptive ID: task-research-for-alice-project-secret
task.purpose	Use category: research or content-creation	Use full description: Research climate data for Alice's PhD thesis on...
task.created_by	Use pseudonymous ID: user:u123456	Use email: alice.smith@example.com
audit.trace_id	Use per-task UUID (rotated)	Use stable agent-wide ID (enables long-term correlation)

Table 11

**\*Unnecessary Claims SHOULD Be Omitted:\*** - Don't include agent.name if agent.id is sufficient for authorization - Don't include task.metadata unless necessary for constraint enforcement - Don't include context.location.ip\_address unless geofencing is required

### 13.3. Retention and Lifecycle

**\*Token Retention:\*** - Tokens expire per exp claim (typically minutes to hours) - Expired tokens SHOULD be immediately discarded by agents - Tokens SHOULD NOT be persisted to disk (memory-only storage)

**\*Audit Log Retention:\*** - Audit logs SHOULD follow organizational retention policy - Minimum retention: As required by compliance framework (e.g., SOC 2: 1 year) - Maximum retention: Only as long as necessary for audit and investigation - After retention period: Logs MUST be securely deleted or anonymized

**\*Delegation Chain Retention:\*** - Delegation chains SHOULD NOT be retained in logs beyond audit window - When logging delegation events, consider hashing agent IDs rather than storing plaintext

**\*Right to Erasure (GDPR Article 17):\*** - If task.created\_by contains personal data, organizations MUST support erasure requests - Consider using pseudonymous IDs that can be de-linked from personal data - Audit logs may be exempt from erasure if required for legal compliance; consult legal counsel

#### 13.4. Cross-Domain Correlation

**\*Threat:\*** Malicious Resource Servers in different organizations could correlate requests across services using stable identifiers in AAP tokens (e.g., audit.trace\_id, agent.id).

**\*Impact:\*** - Agent behavior profiling (what capabilities, what resources, what patterns) - Competitive intelligence (infer agent strategies from request patterns) - Privacy violation (correlate agent activity across unrelated services)

**\*Mitigations:\***

**\*Trace ID Rotation (REQUIRED for Cross-Domain):\*** - When token audience changes trust domain, generate new audit.trace\_id - Example: Token for api.example.com has trace\_id: abc123; delegated token for external.example has trace\_id: xyz789 - Correlation within single organization preserved; cross-organization correlation prevented

**\*Domain-Specific Agent IDs:\*** - Use different agent.id per Resource Server trust domain - Example: Agent presents agent:internal-001 to internal APIs, agent:partner-facing-alpha to partner APIs - Authorization Server maintains mapping; Resource Servers see domain-specific IDs

**\*Minimize Identifiable Information in Delegated Tokens:\*** - When delegating to external tool, remove non-essential claims - Example: Strip task.created\_by, context.location when crossing trust boundary - Token Exchange request SHOULD specify required\_claims (minimal set)

**\*Trace ID Scope Claim (RECOMMENDED):\***

```
{
  "audit": {
    "trace_id": "550e8400-e29b-41d4-a716-446655440000",
    "trace_id_scope": "task"
  }
}
```

This signals the intended correlation boundary: - task: Trace ID unique per task (rotated between tasks) - session: Trace ID unique per agent session (rotated on agent restart) - agent: Trace ID stable

for agent lifetime (enables long-term correlation; privacy-risky) -  
domain: Trace ID unique per trust domain (rotated when crossing  
domains)

### 13.5. Privacy-Preserving Error Messages

Resource Servers MUST NOT leak authorization details in error responses that could enable privacy violations or capability profiling.

**\*Avoid:\*** - "Agent does not have capability 'delete.data'" (reveals capabilities) - "Rate limit: 51 requests, max allowed 50" (reveals constraint values) - "Domain blocked: malicious.example is in blocklist" (reveals policy details) - "Task purpose 'research for Alice' does not match action 'publish'" (reveals task details)

**\*Prefer:\*** - "Insufficient permissions" (generic, privacy-preserving) - "Request violates capability constraints" (doesn't specify which constraint) - "Authorization failed" (minimal information disclosure)

**\*Detailed Error Information:\*** - Log server-side with full details (for audit and debugging) - Include error correlation ID in response for support tickets - Client can reference correlation ID when contacting support; support team accesses server logs

**\*Example Privacy-Preserving Error Response:\***

```
{
  "error": "insufficient_permissions",
  "error_description": "The request could not be authorized",
  "error_correlation_id": "err-550e8400-e29b-41d4-a716-446655440000"
}
```

Server log (not returned to client):

```
{
  "error_correlation_id": "err-550e8400-e29b-41d4-a716-446655440000",
  "error": "aap_constraint_violation",
  "constraint_violated": "domains_allowed",
  "requested_domain": "malicious.example",
  "allowed_domains": ["example.org"],
  "agent_id": "agent-researcher-01",
  "task_id": "task-12345",
  "timestamp": "2024-01-01T12:00:00Z"
}
```

### 13.6. Anonymization and Pseudonymization

#### \*Pseudonymization (GDPR Article 4(5))\*

Pseudonymization is the processing of personal data such that it can no longer be attributed to a specific data subject without additional information (kept separately and secured).

#### \*AAP Pseudonymization Techniques\*

Data Type	Pseudonymization Approach
User IDs (task.created_by)	Use UUID mapped to real user ID in separate database
Agent names	Use agent:0001 instead of agent-for-alice
Task purposes	Use task category codes instead of free-text descriptions
Trace IDs	Use cryptographic hash of (user ID + task ID + salt)

Table 12

#### \*Anonymization (Irreversible)\*

For audit logs past retention period: - Replace agent.id with hash (if agent identity no longer needed) - Remove task.created\_by entirely - Aggregate statistics (e.g., "1000 requests by research agents") instead of individual records

### 13.7. Consent and Transparency

#### \*Agent Operator Transparency\*

When an agent acts on behalf of a human user: - User SHOULD be informed that an agent will perform actions - User SHOULD be shown the task purpose and capabilities granted - User SHOULD be able to review and revoke agent authorizations

#### \*Example User Notification\*

Your request to "research climate change impacts" has been assigned to an AI agent.

The agent will be able to:

- Search web resources from example.org (max 50 requests/hour)
- Create draft articles in the CMS

The agent will NOT be able to:

- Publish articles (requires your approval)
- Access data outside example.org
- Perform actions unrelated to research

You can revoke this authorization at any time in your account settings.

**\*Token Transparency:\***

For compliance with transparency requirements (GDPR Article 13-14): - Organizations SHOULD document what claims are included in AAP tokens - Organizations SHOULD inform users when their actions trigger agent authorization - Organizations SHOULD provide access to audit logs (subject to security and legal constraints)

### 13.8. Third-Party Tool Privacy

When delegating to third-party tools (external organizations):

**\*Data Sharing Agreement:\*** - Delegation to external tool constitutes data sharing - Organizations SHOULD have data processing agreements (DPAs) with tool providers - Token Exchange to external AS SHOULD trigger data sharing notification/consent

**\*Claim Filtering:\*** - Authorization Server SHOULD filter claims when delegating to external tool - Remove non-essential claims: task.created\_by, agent.operator internal details - Retain only authorization-essential claims: capabilities, delegation.depth

**\*Example Filtered Delegation:\***

Original token (internal):

```
{
  "agent": {
    "id": "agent-001",
    "operator": "org:acme-corp"
  },
  "task": {
    "id": "task-123",
    "purpose": "research",
    "created_by": "user:alice"
  },
  "capabilities": [
    { "action": "search.web" }
  ]
}
```

Delegated token (external tool):

```
{
  "agent": {
    "id": "delegated-from:acme-corp",
    "operator": "org:acme-corp"
  },
  "task": {
    "id": "task-123-external",
    "purpose": "research"
  },
  "capabilities": [
    {
      "action": "search.web",
      "constraints": {
        "domains_allowed": ["example.org"]
      }
    }
  ],
  "delegation": {
    "depth": 1,
    "chain": ["agent-001", "external-tool"]
  }
}
```

### 13.9. Privacy by Design Recommendations

**\*For Authorization Server Implementations:\***

1. Default to short trace ID rotation (per-task, not per-agent)
2. Provide configuration options for privacy levels (minimal, standard, full disclosure)

3. Support claim filtering on Token Exchange
4. Log privacy-impacting events (cross-domain delegation, long trace ID retention)

**\*For Resource Server Implementations:\***

1. Never log tokens in plaintext
2. Redact personal data in error responses
3. Provide audit log access controls (only authorized personnel)
4. Support audit log anonymization after retention period

**\*For Agent Implementations:\***

1. Request minimal capabilities (least privilege reduces privacy risk)
2. Specify minimal required\_claims on Token Exchange
3. Rotate trace IDs when crossing trust boundaries
4. Discard tokens immediately upon expiration (don't cache expired tokens)

#### 13.10. Regulatory Compliance Guidance

**\*GDPR Compliance:\*** - AAP supports GDPR principles: purpose limitation (task binding), data minimization (constrained capabilities), accountability (audit logs) - Organizations MUST implement additional controls: consent management, data subject rights, DPIAs

**\*CCPA Compliance:\*** - AAP audit logs may constitute "personal information" if they identify consumers - Organizations MUST support consumer rights: access, deletion, opt-out of sale - Consider using pseudonymous IDs to reduce CCPA applicability

**\*HIPAA Compliance (Healthcare):\*** - AAP tokens accessing Protected Health Information (PHI) MUST use proof-of-possession - Audit logs MUST meet HIPAA retention requirements (6 years) - Delegation chains provide required access audit trail

**\*Note:\*** AAP is an authorization protocol, not a complete privacy framework. Organizations MUST implement privacy policies, procedures, and controls beyond AAP technical mechanisms.

## 14. IANA Considerations

This document registers the following claim names in the IANA "JSON Web Token Claims" registry established by [RFC7519].

### 14.1. JWT Claims Registration

The following claims are registered per the "Specification Required" policy defined in [RFC7519] Section 10.1:

Claim Name	Claim Description	Change Controller	Reference
agent	Agent identity and metadata	IETF	[this document] Section 5.1
task	Task binding information	IETF	[this document] Section 5.2
capabilities	Granted capabilities with constraints	IETF	[this document] Section 5.3
delegation	Delegation chain tracking	IETF	[this document] Section 5.4
oversight	Human oversight requirements	IETF	[this document] Section 5.8
audit	Audit and tracing metadata	IETF	[this document] Section 5.9
context	Execution context restrictions	IETF	[this document] Section 5.10

Table 13

### 14.2. OAuth Error Code Registration

This document registers the following error codes in the OAuth Extensions Error Registry:

Error Code	Usage Location	Protocol Extension	Reference
aap_invalid_capability	Resource access error response	AAP	[this document] Appendix C
aap_constraint_violation	Resource access error response	AAP	[this document] Appendix C
aap_approval_required	Resource access error response	AAP	[this document] Appendix C
aap_excessive_delegation	Resource access error response	AAP	[this document] Appendix C
aap_domain_not_allowed	Resource access error response	AAP	[this document] Appendix C
aap_task_mismatch	Resource access error response	AAP	[this document] Appendix C
aap_agent_not_recognized	Resource access error response	AAP	[this document] Appendix C
aap_invalid_delegation_chain	Resource access error response	AAP	[this document] Appendix C
aap_capability_expired	Resource access error response	AAP	[this document] Appendix C

aap_invalid_context	Resource access error response	AAP	[this document] Appendix C
---------------------	-----------------------------------------	-----	-------------------------------------

Table 14

## 15. Implementation Status

This section records the status of known implementations of the protocol defined by this specification at the time of posting of this Internet-Draft, and is based on a proposal described in [RFC7942].

### 15.1. Reference Implementation

- \* \*Organization:\* AAP Project
- \* \*Implementation:\* Python reference implementation (Authorization Server + Resource Server)
- \* \*Description:\* Complete implementation of AAP token issuance, Token Exchange, and Resource Server validation with constraint enforcement
- \* \*Maturity:\* Alpha
- \* \*Coverage:\* All MUST-level requirements in Sections 7 and 8
- \* \*Licensing:\* Apache 2.0
- \* \*Contact:\* See project repository
- \* \*URL:\* See /reference-impl/ in the project repository

### 15.2. Test Vectors

A comprehensive test vector suite is available for interoperability testing: - 15 test vector files covering valid tokens, invalid tokens, constraint violations, and edge cases - 80+ individual test cases - Coverage of all specification sections referenced in Sections 5, 7, and 8

Implementations SHOULD pass all test vectors in valid-tokens/ and edge-cases/ categories. Implementations MUST correctly reject all test vectors in invalid-tokens/ and constraint-violations/ categories with the specified error codes.

## 16. Related Work and Comparison

This section positions AAP within the broader ecosystem of authorization, access control, and identity systems. AAP is designed to complement and extend existing standards rather than replace them.

### 16.1. OAuth 2.0 Scopes

Traditional OAuth 2.0 scopes [RFC6749] provide coarse-grained authorization through simple string tokens (e.g., read:articles, write:cms). While effective for user-delegated access, scopes have limitations for autonomous agent scenarios:

**\*OAuth Scopes Characteristics:** - Simple string-based permissions - No built-in constraint mechanism - No task context binding - No delegation tracking - Limited expressiveness for complex policies

**\*AAP Enhancements:** - **\*Structured capabilities:** Actions (search.web) with typed, enforceable constraints (max\_requests\_per\_hour: 50, domains\_allowed: ["example.org"]) - **\*Task binding:** Scopes don't express purpose or context; AAP tokens bind authorization to specific tasks with declared purposes - **\*Delegation chains:** OAuth has no built-in delegation tracking; AAP provides delegation.depth and delegation.chain for auditable multi-hop authorization - **\*Operational limits:** Scopes are binary (granted/not granted); AAP constraints enable quantitative limits (rate, volume, time windows)

**\*Comparison Examples:**

Research agent web scraping:

- \* OAuth: scope=read:web (unlimited web access)
- \* AAP: capabilities with action: search.web, domains\_allowed, max\_requests\_per\_hour: 50

Content creation with approval:

- \* OAuth: scope=write:cms (no draft vs. publish)
- \* AAP: action: cms.create\_draft plus oversight.requires\_human\_approval\_for: [cms.publish]

Delegated tool calls:

- \* OAuth: Requires new OAuth flow per delegation

\* AAP: Token Exchange with automatic delegation.depth increment and privilege reduction

\*Compatibility:\* AAP maintains backward compatibility by optionally including OAuth scope claim alongside AAP capabilities.

## 16.2. Fine-Grained Authorization Systems (Zanzibar, ReBAC)

Google Zanzibar and Relation-Based Access Control (ReBAC) systems focus on resource-level permissions with global consistency ("user X can read document Y based on relationship graph").

\*Zanzibar/ReBAC Characteristics:\* - Resource-centric authorization - Graph-based relationship evaluation - Highly scalable, globally consistent - Focus: "Who can access what resource?"

\*AAP Focus:\* - Agent-centric authorization - Task context and operational constraints - Delegation chain tracking - Focus: "What can this agent do, under what conditions, for what purpose?"

\*Complementary Use:\* AAP and Zanzibar address different layers: -

\*AAP\*: Authorizes the agent to call APIs, binds actions to tasks, enforces operational limits - \*Zanzibar\*: Determines whether the agent (once authorized) can access specific resources

\*Integration Pattern:\*

1. Agent presents AAP token to API (capability: "document.read")
2. API validates AAP token (RS validation)
3. API calls Zanzibar to check if agent can read specific document
4. Both checks must pass for request to succeed

## 16.3. Cloud Identity and Access Management

Major cloud providers offer identity and temporary credential systems:

\*AWS Security Token Service (STS):\* - Cross-account delegation with session policies - Temporary credentials with limited lifetime - AssumeRole for privilege escalation/reduction - IAM policies define permissions

\*GCP Workload Identity:\* - Kubernetes ServiceAccount federation to GCP IAM - Workload identity binding for containerized applications - OAuth 2.0-based token exchange

**\*AAP Differences:** - **\*Vendor-neutral:** AAP is not tied to a specific cloud provider - **\*Agent-specific claims:** agent.model, task.purpose, oversight requirements not present in cloud IAM - **\*Explicit delegation tracking:** Cloud IAM tracks roles/sessions but not multi-hop agent delegation chains - **\*Task binding:** Cloud credentials are session-bound, not task-bound

**\*Integration:** AAP can be used as a policy enforcement layer above cloud IAM. For example: - AAP token authorizes agent for a task - Agent exchanges AAP token for cloud-specific credentials (AWS STS, GCP token) - Cloud IAM enforces resource-level permissions

#### 16.4. Service Mesh Authorization (Istio, Linkerd)

Service meshes provide network-level security with mutual TLS and Layer 7 policies:

**\*Service Mesh Capabilities:** - mTLS for service-to-service encryption - SPIFFE/SPIRE for workload identity - Authorization policies based on service identity - Network-level request routing and policy enforcement

**\*AAP Capabilities:** - Application-level agent identity with model and operator metadata - Task semantics and purpose binding - Business logic constraints (rate limits, domain allowlists, approval workflows) - Delegation chain for multi-agent workflows

**\*Complementary Layers:**

Layer	Technology	What It Provides
Network (L4/L7)	Service Mesh	Identity (SPIFFE), mTLS, service-to-service authz
Application	AAP	Agent identity, task binding, capability constraints, oversight

Table 15

**\*Integration:** - Use SPIFFE IDs in AAP agent.id claim - Service mesh enforces network policy ("can agent reach API?") - AAP enforces business logic ("can agent perform this action with these constraints?")

### 16.5. Capability-Based Security

AAP uses the term "capability" from classic capability-based security literature (Dennis & Van Horn 1966, Levy 1984):

**\*Classic Capability Systems:** - Unforgeable tokens (capabilities) that grant access to specific objects - Capabilities combine authority and designation - No ambient authority (no ACLs checked at access time) - Pure object-capability model (E language, seL4 microkernel)

**\*AAP Capabilities:** - Structured claims in signed JWTs representing permissions - Combine action designation (search.web) with constraints - Rely on centralized Authorization Server (not pure capability model) - Inherit unforgeability from JWT signature (not object references)

**\*Key Difference:** AAP is not a pure object-capability system. It uses centralized issuance (AS) and validation (RS) rather than distributed capability passing. However, AAP borrows the principle of **\*least authority\***: each capability explicitly states what is allowed and under what constraints, rather than checking ambient permissions.

### 16.6. OpenID Connect and Step-Up Authentication

OpenID Connect (OIDC) [OIDC] provides identity claims and authentication strength signaling:

**\*OIDC Capabilities:** - User identity claims (sub, email, name) - Authentication context (acr, amr claims) - Step-up authentication via acr\_values parameter - ID tokens separate from access tokens

**\*AAP Capabilities:** - Agent identity (non-human, autonomous software) - Task binding and operational constraints - Oversight requirements signal when human approval needed - Access tokens include both identity and authorization

**\*Overlap:** Both use JWT and OAuth 2.0 foundation.

**\*Integration:** - AAP oversight.requires\_human\_approval\_for can trigger OIDC step-up authentication - Resource Server can request user approval via OIDC interaction, using acr\_values for higher assurance - Human supervisor identified in oversight.supervisor can authenticate via OIDC

## 16.7. OAuth 2.0 Rich Authorization Requests (RAR)

Rich Authorization Requests extend OAuth to support complex authorization requirements beyond simple scopes.

**\*RAR Features:** - Structured authorization details in JSON - Request-specific authorization (e.g., payment with amount) - Detailed, fine-grained permissions

**\*AAP vs. RAR:** - **\*RAR:** Request-time authorization details sent by client - **\*AAP:** Token-time structured claims issued by AS, enforced by RS - **\*RAR:** Focuses on authorization request - **\*AAP:** Focuses on token structure and enforcement

**\*Complementary:** RAR can be used to request AAP capabilities. Client sends RAR with desired capabilities; AS issues AAP token with granted capabilities.

## 16.8. Summary: Where AAP Fits

AAP occupies a unique position in the authorization ecosystem:

System	AAP Relationship
OAuth Scopes	Extends with capabilities
Zanzibar/ReBAC	AAP: agent layer; Zanzibar: resource layer
Cloud IAM	Vendor-neutral; integrates with cloud IAM
Service Mesh	Adds app-level semantics (task, constraints)
OIDC	Agent identity; integrates for oversight
RAR	Defines token structure; RAR requests capabilities

Table 16

**\*AAP is designed for scenarios where:** - The client is an autonomous AI agent (not a human user) - Actions must be bound to explicit tasks with declared purposes - Authorization requires operational constraints (rate limits, domain restrictions, time windows) - Delegation chains must be tracked and auditable - Human oversight is required for high-risk actions - Standard OAuth scopes are insufficient for expressing agent policies

## 17. References

### 17.1. 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/rfc/rfc2119>>.
- [RFC5234] Crocker, D., Ed. and P. Overell, "Augmented BNF for Syntax Specifications: ABNF", STD 68, RFC 5234, DOI 10.17487/RFC5234, January 2008, <<https://www.rfc-editor.org/rfc/rfc5234>>.
- [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>>.
- [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>>.
- [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>>.
- [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/rfc/rfc8174>>.
- [RFC8693] Jones, M., Nadalin, A., Campbell, B., Ed., Bradley, J., and C. Mortimore, "OAuth 2.0 Token Exchange", RFC 8693, DOI 10.17487/RFC8693, January 2020, <<https://www.rfc-editor.org/rfc/rfc8693>>.
- [RFC8705] Campbell, B., Bradley, J., Sakimura, N., and T. Lodderstedt, "OAuth 2.0 Mutual-TLS Client Authentication and Certificate-Bound Access Tokens", RFC 8705, DOI 10.17487/RFC8705, February 2020, <<https://www.rfc-editor.org/rfc/rfc8705>>.
- [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>>.

### 17.2. Informative References

- [RFC7009] Lodderstedt, T., Ed., Dronia, S., and M. Scurtescu, "OAuth 2.0 Token Revocation", RFC 7009, DOI 10.17487/RFC7009, August 2013, <<https://www.rfc-editor.org/rfc/rfc7009>>.
- [RFC7517] Jones, M., "JSON Web Key (JWK)", RFC 7517, DOI 10.17487/RFC7517, May 2015, <<https://www.rfc-editor.org/rfc/rfc7517>>.
- [RFC7662] Richer, J., Ed., "OAuth 2.0 Token Introspection", RFC 7662, DOI 10.17487/RFC7662, October 2015, <<https://www.rfc-editor.org/rfc/rfc7662>>.
- [RFC7800] Jones, M., Bradley, J., and H. Tschofenig, "Proof-of-Possession Key Semantics for JSON Web Tokens (JWTs)", RFC 7800, DOI 10.17487/RFC7800, April 2016, <<https://www.rfc-editor.org/rfc/rfc7800>>.
- [RFC7942] Sheffer, Y. and A. Farrel, "Improving Awareness of Running Code: The Implementation Status Section", BCP 205, RFC 7942, DOI 10.17487/RFC7942, July 2016, <<https://www.rfc-editor.org/rfc/rfc7942>>.
- [OIDC] OpenID Foundation, "OpenID Connect Core 1.0", 2014, <[https://openid.net/specs/openid-connect-core-1\\_0.html](https://openid.net/specs/openid-connect-core-1_0.html)>.
- [SPIFFE] SPIFFE, "SPIFFE: Secure Production Identity Framework for Everyone", 2024, <<https://spiffe.io>>.
- [OpenTelemetry] CNCF, "OpenTelemetry", 2024, <<https://opentelemetry.io>>.

## Appendix A. Complete JSON Schema Reference

This appendix provides links to the complete JSON Schema definitions for AAP tokens. These schemas are normative and MUST be used for validation in conforming implementations.

### A.1. Schema Files

All schemas are available in the /schemas directory of the reference implementation repository:

- \* `*aap-token.schema.json*` - Root schema for complete AAP token payload
  - Validates all required standard JWT claims (iss, sub, aud, exp, iat)

- Validates all required AAP claims (agent, task, capabilities)
- Validates optional AAP claims (oversight, delegation, context, audit)
- \* `*aap-agent.schema.json*` - Schema for agent claim
  - Validates agent identity structure
  - Required fields: id, type, operator
  - Optional fields: name, version, model, runtime, certification
- \* `*aap-task.schema.json*` - Schema for task claim
  - Validates task binding structure
  - Required fields: id, purpose
  - Optional fields: created\_by, created\_at, expires\_at, priority, category
- \* `*aap-capabilities.schema.json*` - Schema for capabilities array
  - Validates capability structure
  - Each capability requires action field (validated against ABNF grammar via regex)
  - Optional constraints, resources, conditions fields
- \* `*aap-constraints.schema.json*` - Schema for constraint objects
  - Defines all standard constraint types with validation rules
  - Includes type definitions for rate limits, domain lists, time windows, etc.
- \* `*aap-oversight.schema.json*` - Schema for oversight claim
  - Validates human oversight requirements
  - Includes requires\_human\_approval\_for array, approval\_reference, etc.
- \* `*aap-delegation.schema.json*` - Schema for delegation claim
  - Validates delegation chain structure

- Required fields: depth, max\_depth
- Optional fields: chain, parent\_jti, privilege\_reduction
- \* `*aap-context.schema.json*` - Schema for context claim
  - Validates execution context information
  - Includes environment, location, runtime, session details
- \* `*aap-audit.schema.json*` - Schema for audit claim
  - Validates audit and logging requirements
  - Required field: trace\_id
  - Optional fields: log\_level, retention\_period, compliance\_framework

#### A.2. Using the Schemas

`*JSON Schema Version:` All schemas use JSON Schema Draft 2020-12.

`*Validation Example (Node.js with Ajv):`

```
const Ajv = require('ajv');
const addFormats = require('ajv-formats');

const ajv = new Ajv();
addFormats(ajv);

// Load all schemas
const schemas = {
  token: require('./schemas/aap-token.schema.json'),
  agent: require('./schemas/aap-agent.schema.json'),
  task: require('./schemas/aap-task.schema.json'),
};

// Add referenced schemas
ajv.addSchema(schemas.agent);
ajv.addSchema(schemas.task);

// Validate token
const validate = ajv.compile(schemas.token);
const valid = validate(decodedJWT);

if (!valid) {
  console.error('Validation errors:', validate.errors);
}

*Validation Example (Python with jsonschema):*
```

```
import jsonschema
import json

# Load schemas
with open('schemas/aap-token.schema.json') as f:
    token_schema = json.load(f)

# Create resolver for $ref
store = {}
for schema_file in ['aap-agent.schema.json', 'aap-task.schema.json']:
    with open(f'schemas/{schema_file}') as f:
        store[schema_file] = json.load(f)

resolver = jsonschema.RefResolver.from_schema(
    token_schema, store=store)

# Validate
try:
    jsonschema.validate(decoded_jwt, token_schema, resolver=resolver)
    print("Token is valid")
except jsonschema.ValidationError as e:
    print(f"Validation error: {e.message}")
```

### A.3. Schema Updates

When the AAP specification is updated: - Schema version will be incremented in the \$id field - Breaking changes will trigger new major version - Non-breaking additions (new optional fields) will increment minor version - Implementations SHOULD validate against the schema version matching the specification version

## Appendix B. Token Exchange Flow Example

This appendix provides a detailed example of OAuth 2.0 Token Exchange [RFC8693] for AAP delegation scenarios.

### B.1. Scenario

A research agent (Agent A) needs to delegate web scraping capability to a specialized tool (Tool B):

- \* \*Original agent:\* agent-researcher-01 (depth=0)
- \* \*Delegated tool:\* tool-web-scraper (depth=1)
- \* \*Capability reduction:\* Remove cms.create\_draft, keep only search.web with tighter constraints

\* \*Lifetime reduction:\* Original token: 3600s, derived token: 1800s

## B.2. Step 1: Agent Obtains Original Token

\*Request to Authorization Server:\*

```
POST /token HTTP/1.1
Host: as.example.com
Content-Type: application/x-www-form-urlencoded
```

```
grant_type=client_credentials
&client_id=agent-researcher-01
&client_secret=[SECRET]
&scope=aap:research
```

\*Authorization Server Response:\*

```
{
  "access_token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9...",
  "token_type": "Bearer",
  "expires_in": 3600,
  "scope": "aap:research"
}
```

\*Decoded Token Payload (Original):\*

```
{
  "iss": "https://as.example.com",
  "sub": "agent-researcher-01",
  "aud": "https://api.example.com",
  "exp": 1704067200,
  "iat": 1704063600,
  "jti": "token-original-123",
  "agent": {
    "id": "agent-researcher-01",
    "type": "llm-autonomous",
    "operator": "org:acme-corp"
  },
  "task": {
    "id": "task-123",
    "purpose": "research_climate_data"
  },
  "capabilities": [
    {
      "action": "search.web",
      "constraints": {
        "domains_allowed": ["example.org", "trusted.example"],
        "max_requests_per_hour": 100
      }
    },
    {
      "action": "cms.create_draft"
    }
  ],
  "delegation": {
    "depth": 0,
    "max_depth": 2,
    "chain": ["agent-researcher-01"]
  }
}
```

### B.3. Step 2: Agent Exchanges Token for Tool-Specific Token

\*Token Exchange Request:\*

```
POST /token HTTP/1.1
Host: as.example.com
Content-Type: application/x-www-form-urlencoded
```

```
grant_type=urn:ietf:params:oauth:grant-type:token-exchange
&subject_token=eyJhbGciOiJIJFZlIiwiaXN5cCI6IkpXVCJ9...
&subject_token_type=urn:ietf:params:oauth:token-type:access_token
&resource=https://tool-scraper.example.com
&scope=aap:research.scraping
&requested_token_type=urn:ietf:params:oauth:token-type:access_token
```

\*Key Parameters:\* - grant\_type: Token Exchange grant type -  
subject\_token: The original AAP token (from Step 1) - resource:  
Intended audience for derived token (Tool B) - scope: Reduced scope  
for delegation

\*Authorization Server Processing:\*

1. Validate subject\_token (signature, expiration, issuer)
2. Extract delegation depth from subject token: depth = 0
3. Check if delegation allowed: depth < max\_depth (0 < 2) PASS
4. Determine reduced capabilities:
  - \* Keep only search.web (remove cms.create\_draft - not needed by scraper)
  - \* Tighten constraints: max\_requests\_per\_hour: 100 -> 50
  - \* Narrow domains\_allowed: Keep only example.org (remove trusted.example)
5. Reduce token lifetime: 3600s -> 1800s (50% reduction for delegated token)
6. Increment delegation depth: 0 -> 1
7. Append to delegation chain: ["agent-researcher-01"] -> ["agent-researcher-01", "tool-web-scraper"]
8. Record parent\_jti: token-original-123
9. Generate new jti: token-delegated-456
10. Sign and issue derived token

\*Authorization Server Response:\*

```
{
"access_token": "eyJhbGciOiJIJFZlNiJ9...",
"issued_token_type": "urn:ietf:params:oauth:token-type:access_token",
"token_type": "Bearer",
"expires_in": 1800,
"scope": "aap:research.scraping"
}
```

\*Decoded Token Payload (Derived):\*

```
{
  "iss": "https://as.example.com",
  "sub": "agent-researcher-01",
  "aud": "https://tool-scraper.example.com",
  "exp": 1704065400,
  "iat": 1704063600,
  "jti": "token-delegated-456",
  "agent": {
    "id": "agent-researcher-01",
    "type": "llm-autonomous",
    "operator": "org:acme-corp"
  },
  "task": {
    "id": "task-123",
    "purpose": "research_climate_data"
  },
  "capabilities": [
    {
      "action": "search.web",
      "constraints": {
        "domains_allowed": ["example.org"],
        "max_requests_per_hour": 50
      }
    }
  ],
  "delegation": {
    "depth": 1,
    "max_depth": 2,
    "chain": [
      "agent-researcher-01",
      "tool-web-scraper"
    ],
    "parent_jti": "token-original-123",
    "privilege_reduction": {
      "capabilities_removed": ["cms.create_draft"],
      "constraints_added": [],
      "lifetime_reduced_by": 1800
    }
  }
}
```

## B.4. Key Changes in Derived Token

Field	Change
aud	Changed to tool audience
exp	Lifetime reduced 50%
jti	New unique ID
capabilities	Removed cms.create_draft
domains_allowed	Narrowed to example.org only
max_requests_per_hour	Reduced from 100 to 50
delegation.depth	Incremented from 0 to 1
delegation.chain	Tool appended
delegation.parent_jti	Parent link added

Table 17

## B.5. Privilege Reduction Summary

The Authorization Server applied the principle of *\*least privilege\** by:

1. *\*Capability Removal:*\* Tool doesn't need cms.create\_draft (CMS access) for scraping task
2. *\*Constraint Tightening:*\* Reduced rate limit from 100 to 50 requests/hour
3. *\*Domain Narrowing:*\* Removed trusted.example from allowed domains (tool only accesses example.org)
4. *\*Lifetime Reduction:*\* 1800s instead of 3600s (shorter validity window reduces risk)

These reductions ensure that if the tool is compromised or malicious, damage is limited to the specific delegated capabilities and constraints.

## B.6. Further Delegation (Depth=2)

If Tool B needs to delegate to another tool (e.g., HTML parser), it can repeat the Token Exchange process:

- \* Tool B exchanges token-delegated-456 for new token with depth=2
- \* Further capability reduction (e.g., read-only access to specific URLs)
- \* Shorter lifetime (e.g., 900s)
- \* Chain becomes: ["agent-researcher-01", "tool-web-scraper", "tool-html-parser"]
- \* max\_depth=2 prevents delegation beyond this point

## Appendix C. AAP-Specific Error Codes

This appendix defines error codes specific to AAP authorization failures, extending the OAuth 2.0 error code registry.

### C.1. Error Code Reference

- aap\_invalid\_capability (403): No matching capability for requested action. Example: Agent has search.web but requests cms.publish.
- aap\_constraint\_violation (429 or 403): Capability constraint violated. Example: 51st request when max\_requests\_per\_hour is 50.
- aap\_task\_mismatch (403): Request inconsistent with task purpose. Example: Token for "research" task used for data deletion.
- aap\_approval\_required (403): Action requires human approval. Example: cms.publish listed in oversight.requires\_human\_approval\_for.
- aap\_excessive\_delegation (403): Delegation depth exceeded. Example: Token with depth=3 when max\_depth=2.
- aap\_invalid\_context (403): Context restriction violated. Example: Request outside time\_window or from blocked region.
- aap\_domain\_not\_allowed (403): Target domain not in allowlist. Example: Request to malicious.example when domains\_allowed is ["example.org"].
- aap\_agent\_not\_recognized (403): Agent identity not recognized by

policy. Example: Unknown agent.id value.

aap\_invalid\_delegation\_chain (403): Delegation chain validation failed. Example: Chain length does not match depth, or parent token is invalid.

aap\_capability\_expired (403): Time-based capability expired.  
Example: Request after time\_window.end.

## C.2. Error Response Format

AAP error responses SHOULD follow OAuth 2.0 error response format [RFC6749] Section 5.2 with AAP-specific error codes:

```
{
  "error": "aap_constraint_violation",
  "error_description": "The request violates capability constraints",
  "error_uri": "https://aap.example/errors#constraint-violation"
}
```

**\*Privacy Consideration:** Error descriptions SHOULD be generic and MUST NOT leak constraint values, agent details, or policy information. Detailed errors SHOULD be logged server-side only.

## C.3. Error Code Usage Examples

**\*Example 1: Rate Limit Exceeded\***

HTTP/1.1 429 Too Many Requests  
Content-Type: application/json  
Retry-After: 3600

```
{
  "error": "aap_constraint_violation",
  "error_description": "Rate limit exceeded for this capability"
}
```

**\*Example 2: Domain Not Allowed\***

HTTP/1.1 403 Forbidden  
Content-Type: application/json

```
{
  "error": "aap_domain_not_allowed",
  "error_description": "Domain not in allowed list"
}
```

**\*Example 3: Approval Required\***

HTTP/1.1 403 Forbidden

Content-Type: application/json

```
{
  "error": "aap_approval_required",
  "error_description": "This action requires human approval",
  "approval_reference": "https://approve.example.com/task-123"
}
```

#### C.4. Error Handling Guidance for Clients

When receiving AAP error codes, agents SHOULD:

1. `*aap_constraint_violation` with 429:\* Respect Retry-After header; back off requests
2. `*aap_approval_required`:\* Present `approval_reference` to operator for human approval
3. `*aap_invalid_capability`:\* Do not retry; request new token with correct capabilities
4. `*aap_excessive_delegation`:\* Do not attempt further delegation; use current token directly
5. `*aap_domain_not_allowed`:\* Validate domains before requests to avoid repeated errors

Agents MUST NOT attempt to bypass errors by modifying tokens (signature validation will fail).

#### Appendix D. Conformance Checklist

This appendix provides implementation checklists for Authorization Servers and Resource Servers to verify AAP conformance.

##### D.1. Authorization Server Conformance Checklist

An AAP-conformant Authorization Server MUST implement the following:

`*Token Issuance`:

- \* Supports OAuth 2.0 Client Credentials Grant [RFC6749] Section 4.4
- \* Issues JWTs with all required AAP claims: agent, task, capabilities
- \* Validates capabilities against operator policy before issuance

- \* Supports at least one proof-of-possession mechanism (DPoP [RFC9449] or mTLS [RFC8705])
- \* Signs tokens with ES256 or RS256 (not HS256)
- \* Issues tokens with unique jti (JWT ID) for each token
- \*Delegation (Token Exchange):\*
- \* Supports OAuth 2.0 Token Exchange [RFC8693]
- \* Implements privilege reduction on delegation (capability subset, constraint tightening, lifetime reduction)
- \* Increments delegation.depth on each Token Exchange
- \* Appends tool/agent ID to delegation.chain
- \* Validates parent token (via parent\_jti) is not expired or revoked before issuing derived token
- \* Enforces delegation.depth < delegation.max\_depth before issuance
- \* MUST NOT issue token if resulting depth exceeds max\_depth
- \*Claim Validation:\*
- \* Validates capabilities array is non-empty
- \* Validates action fields conform to ABNF grammar (Section 5.5)
- \* Validates constraint types against standard definitions (Section 5.6)
- \* Applies operator policy to reduce requested capabilities to authorized subset
- \*Key Management:\*
- \* Stores private signing keys in HSM or equivalent secure storage (production)
- \* Rotates signing keys every 90 days (or per organization policy)
- \* Publishes public keys via JWKS endpoint [RFC7517]
- \* Supports multiple concurrent signing keys (for rotation overlap)

**\*Revocation:\***

- \* Provides token revocation endpoint [RFC7009] or introspection [RFC7662]
- \* Distributes revocation events to Resource Servers within 30 seconds
- \* Optionally supports token family revocation (revoke parent + descendants)

**\*Audit and Logging:\***

- \* Logs all token issuance events with agent.id, task.id, capabilities, timestamp
- \* Logs Token Exchange events with parent-child JTI linkage
- \* Supports trace ID correlation (audit.trace\_id from request)
- \* Provides tamper-evident audit logs (cryptographic chaining or append-only storage)

**D.2. Resource Server Conformance Checklist**

An AAP-conformant Resource Server MUST implement the following:

**\*Standard Token Validation:\***

- \* Validates token signature using AS public key (via JWKS)
- \* Validates token expiration (exp claim) with acceptable clock skew (<=5 minutes)
- \* Validates audience (aud claim) matches Resource Server identifier
- \* Validates issuer (iss claim) is trusted Authorization Server
- \* Checks token revocation status (if revocation mechanism in place)

**\*Proof-of-Possession Validation:\***

- \* Validates DPoP proof if cnf.jkt present in token
- \* Validates mTLS client certificate if cnf.x5t#S256 present in token
- \* Rejects bearer tokens if proof-of-possession is required by policy

**\*Agent Identity Validation:\***

- \* Validates agent claim is present and well-formed
- \* Validates agent.id is recognized or allowed by local policy
- \* Validates agent.runtime.attested if required by policy
- \* Rejects tokens with agents on deny list

**\*Task Binding Validation:\***

- \* Validates task claim is present
- \* Validates requested action is consistent with task.purpose
- \* Rejects requests that clearly fall outside declared purpose
- \* Enforces time window if task.expires\_at is present

**\*Capability Enforcement:\***

- \* Matches requested operation to capability.action entry
- \* Uses exact string matching for action names (case-sensitive)
- \* Denies request if no matching capability found (returns aap\_invalid\_capability)
- \* Enforces ALL constraints in matching capability (AND semantics)

**\*Constraint Enforcement:\***

- \* Enforces max\_requests\_per\_hour (fixed window, resets at minute 0)
- \* Enforces max\_requests\_per\_minute (sliding 60-second window)
- \* Enforces domains\_allowed (DNS suffix matching, rightmost)
- \* Enforces domains\_blocked (takes precedence over allowlist)
- \* Enforces time\_window (inclusive start, exclusive end)
- \* Enforces max\_depth for delegation
- \* Uses distributed rate limiting for multi-instance deployments

**\*Oversight Enforcement:\***

- \* Checks if action in oversight.requires\_human\_approval\_for
- \* Returns HTTP 403 with aap\_approval\_required if approval needed
- \* Includes approval\_reference in error response
- \*Delegation Validation:\*
- \* Validates delegation.depth <= delegation.max\_depth
- \* Validates delegation.chain length equals depth + 1
- \* Validates delegation depth against capability-specific max\_depth constraints
- \*Error Handling:\*
- \* Returns privacy-preserving error messages (no constraint values, agent details)
- \* Returns appropriate HTTP status codes (401, 403, 429)
- \* Returns AAP-specific error codes (Appendix C)
- \* Logs detailed error information server-side (not in response)
- \*Audit and Logging:\*
- \* Logs all authorized requests with agent.id, task.id, action, outcome
- \* Propagates trace ID from audit.trace\_id to downstream services
- \* Logs authorization failures (with reason, not returned to client)
- \* Supports tamper-evident audit logs

#### D.3. Optional Features (RECOMMENDED)

- \*Authorization Server:\*
- \* Supports multiple token lifetimes based on risk level
- \* Supports dynamic rate limit adjustment
- \* Implements behavioral anomaly detection
- \* Provides policy approval workflow (draft -> review -> active)

- \* Supports claim filtering on Token Exchange (privacy)
- \*Resource Server:\*
- \* Implements behavioral anomaly detection for agent requests
- \* Provides real-time monitoring dashboards
- \* Supports audit log anonymization after retention period
- \* Implements re-validation before executing high-risk actions (TOCTOU mitigation)

#### D.4. Testing Conformance

Organizations SHOULD test conformance using: - \*Test Vectors:\* Validate against AAP test vectors (valid and invalid tokens) - \*Interoperability Tests:\* Verify interoperability with reference implementations - \*Security Audits:\* Third-party security review of AS and RS implementations - \*Compliance Scans:\* Automated conformance checking tools

### Appendix E. Implementation Examples

#### E.1. Example Policy Configuration

\*Operator Policy (JSON format):\*

```
{
  "policy_id": "policy-research-agents-v1",
  "policy_version": "1.0",
  "applies_to": {
    "agent_type": "llm-autonomous",
    "operator": "org:acme-corp"
  },
  "allowed_capabilities": [
    {
      "action": "search.web",
      "default_constraints": {
        "domains_allowed": ["example.org", "trusted.example"],
        "max_requests_per_hour": 100,
        "max_requests_per_minute": 10
      }
    },
    {
      "action": "cms.create_draft",
      "default_constraints": {
        "max_requests_per_hour": 20
      }
    },
    {
      "action": "cms.publish",
      "requires_oversight": true
    }
  ],
  "global_constraints": {
    "token_lifetime": 3600,
    "max_delegation_depth": 2,
    "require_pop": true
  },
  "oversight": {
    "level": "approval",
    "requires_human_approval_for": [
      "cms.publish", "data.delete"
    ],
    "approval_reference": "https://approve.example.com/agents"
  },
  "audit": {
    "log_level": "full",
    "retention_period_days": 90,
    "compliance_framework": ["SOC2", "GDPR"]
  }
}
```

## E.2. Example Token Validation Code (Pseudocode)

```
def validate_aap_token(token, request):
    # 1. Standard OAuth validation
    if not verify_signature(token, AS_PUBLIC_KEY):
        raise InvalidSignature()

    if token.exp < now():
        raise TokenExpired()

    if token.aud != RESOURCE_SERVER_ID:
        raise InvalidAudience()

    # 2. Proof-of-possession (if required)
    if REQUIRE_POP:
        if 'cnf' not in token:
            raise ProofOfPossessionRequired()
        validate_pop(token.cnf, request)

    # 3. Agent identity
    if token.agent.id not in ALLOWED_AGENTS:
        raise AgentNotRecognized()

    # 4. Task binding
    if not is_consistent(request.action, token.task.purpose):
        raise TaskMismatch()

    # 5. Capability matching
    matching_cap = find_capability(
        token.capabilities, request.action)
    if not matching_cap:
        raise NoMatchingCapability()

    # 6. Constraint enforcement
    enforce_constraints(matching_cap.constraints, request, token)

    # 7. Oversight
    approvals = token.oversight.requires_human_approval_for
    if request.action in approvals:
        ref = token.oversight.approval_reference
        raise ApprovalRequired(ref)

    # 8. Delegation depth
    depth = token.delegation.depth
    if depth > token.delegation.max_depth:
        raise ExcessiveDelegation()

    # 9. Audit logging
    log_authorized_request(
        token.agent.id, token.task.id,
```

```
        request.action)

    return AUTHORIZED

def enforce_constraints(constraints, request, token):
    # Rate limiting
    if 'max_requests_per_hour' in constraints:
        hourly = constraints.max_requests_per_hour
        if get_hourly_count(token.jti) >= hourly:
            raise RateLimitExceeded()
        increment_hourly_count(token.jti)

    # Domain allowlist
    if 'domains_allowed' in constraints:
        domain = extract_domain(request.target_url)
        allowed = constraints.domains_allowed
        if not domain_matches_allowlist(domain, allowed):
            raise DomainNotAllowed()

    # Time window
    if 'time_window' in constraints:
        tw = constraints.time_window
        if not (tw.start <= now() < tw.end):
            raise OutsideTimeWindow()

    # Additional constraints...
```

## Appendix F. Test Vectors

This appendix provides canonical test vectors for validating AAP implementations. A complete test vector suite is maintained separately (see Section 15.2). The examples below are normative and illustrate key validation scenarios.

### F.1. Valid Token -- Basic Research Agent

The following JWT payload represents a valid AAP token for a research agent with web search capability:

```
{
  "iss": "https://as.example.com",
  "sub": "agent-researcher-01",
  "aud": "https://api.example.com",
  "exp": 1735689600,
  "iat": 1735686000,
  "jti": "tv-valid-basic-001",
  "agent": {
    "id": "agent-researcher-01",
    "type": "llm-autonomous",
    "operator": "org:acme-corp"
  },
  "task": {
    "id": "task-research-001",
    "purpose": "research"
  },
  "capabilities": [
    {
      "action": "search.web",
      "constraints": {
        "domains_allowed": ["example.org", "trusted.example"],
        "max_requests_per_hour": 100
      }
    }
  ],
  "delegation": {
    "depth": 0,
    "max_depth": 2,
    "chain": ["agent-researcher-01"]
  }
}
```

\*Expected Results:\* - Request to search.web targeting example.org:  
AUTHORIZED - Request to search.web targeting malicious.example:  
FORBIDDEN (aap\_domain\_not\_allowed) - Request to cms.publish:  
FORBIDDEN (aap\_invalid\_capability)

## F.2. Invalid Token -- Excessive Delegation

```
{
  "iss": "https://as.example.com",
  "sub": "agent-researcher-01",
  "aud": "https://api.example.com",
  "exp": 1735689600,
  "iat": 1735686000,
  "jti": "tv-invalid-delegation-001",
  "agent": {
    "id": "agent-researcher-01",
    "type": "llm-autonomous",
    "operator": "org:acme-corp"
  },
  "task": {
    "id": "task-001",
    "purpose": "research"
  },
  "capabilities": [
    {
      "action": "search.web"
    }
  ],
  "delegation": {
    "depth": 4,
    "max_depth": 3,
    "chain": ["agent-01", "tool-a", "tool-b", "tool-c", "tool-d"],
    "parent_jti": "parent-token-id"
  }
}
```

\*Expected Result:\* REJECTED -- aap\_excessive\_delegation (HTTP 403).  
The delegation.depth (4) exceeds delegation.max\_depth (3).

### F.3. Edge Case -- Clock Skew Tolerance

Given a token with exp: 1735686000 and a clock skew tolerance of 300 seconds (5 minutes):

Validation Time	Seconds Past exp	Expected Result	Reason
1735686000	0	REJECTED	Token is expired at exp (exclusive boundary)
1735686240	240	ACCEPTED	Within 5-minute tolerance
1735686300	300	ACCEPTED	At boundary of tolerance (inclusive)
1735686301	301	REJECTED	Beyond tolerance

Table 18

## Author's Address

Angel Cruz  
Independent  
Email: bullgram@gmail.com