Web Authorization Protocol
Internet-Draft
Intended status: Informational
Expires: 3 September 2026

S. Kumar
Grantex

2 March 2026

Delegated Agent Authorization Protocol (DAAP)
draft-mishra-oauth-agent-grants-01

Abstract

Artificial intelligence (AI) agents increasingly take autonomous actions -- submitting forms, initiating payments, and sending communications -- on behalf of human users across third-party services. This document defines the Delegated Agent Authorization Protocol (DAAP), an open, model-neutral, framework-agnostic protocol that specifies: cryptographic agent identity using Decentralized Identifiers (DIDs); a human-consent-based grant authorization flow modelled on OAuth 2.0; a signed JSON Web Token (JWT) grant token format with agent-specific claims; a revocation model with online verification; a hash-chained append-only audit trail; a policy engine for automated authorization decisions; a multi-agent delegation model with cascade revocation; budget controls for spending limits; real-time event streaming; a credential vault for secure secret storage; and external policy backend integration with OPA and Cedar. DAAP fills a gap unaddressed by existing OAuth 2.0 extensions: verifying that a specific human authorized a specific AI agent to perform a specific action, revoking that authorization in real time, and producing a tamper-evident record of what the agent did.

Status of This Memo

Table of Contents

1. Introduction

Deployed AI agents operate across arbitrary third-party services using credentials and permissions that belong to the human users they serve. Today, no interoperable standard exists for:

1. Verifying that an agent is who it claims to be

2. Confirming that a specific human authorized a specific agent to perform a specific action

3. Revoking that authorization in real time across all active tokens

4. Producing a tamper-evident record of agent activity

OAuth 2.0 [RFC6749] and its extensions address authorization for applications acting on behalf of users, but were not designed for the AI agent use case, which introduces distinct requirements:

* *Agent identity*: Unlike OAuth clients, AI agents are runtime entities that may be spawned dynamically and must carry a persistent cryptographic identity independent of the authorization server.

* *Multi-agent delegation*: An agent may spawn sub-agents, each of which requires a grant scoped to a subset of the parent's permissions, with the entire delegation tree revocable by the original principal.

* *Tamper-evident audit trail*: Regulated environments require a cryptographically linked record of agent activity that is verifiable without trust in the audit log operator.

* *Policy-driven automation*: High-volume agent deployments require automated authorization decisions (auto-approve, auto-deny) without per-request human interaction, while preserving the human principal's ability to revoke at any time.

* *Budget controls*: Agents acting in financial contexts require per-grant spending limits with atomic debit semantics and threshold alerts.

* *Real-time observability*: Operators require real-time event streams for monitoring agent activity, grant lifecycle events, and budget threshold crossings.

DAAP addresses these requirements as a layered extension to OAuth 2.0 concepts, reusing RFC-standard JWT and JWK primitives wherever possible.

1.1. Changes Since -00

This revision adds the following extensions:

* Budget Controls (Section 10): per-grant spending limits with the bdg JWT claim, atomic debit operations, and threshold alerting.

* Event Streaming (Section 11): SSE and WebSocket endpoints for real-time event delivery.

* Credential Vault (Section 12): encrypted per-user credential storage with token-to-credential exchange.

* External Policy Backends (Section 13): integration with OPA and Cedar as policy evaluation targets.

* Implementation Report (Appendix B): conformance results and SDK coverage.

* Updated Conformance Requirements (Section 9) with OPTIONAL extensions for Budget, Events, Vault, and Policy Backends.

1.2. Requirements Language

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.

1.3. Terminology

The following terms are used throughout this document:

Agent:
: An AI-powered software process that takes autonomous actions on behalf of a Principal. An Agent has a persistent cryptographic identity (DID) and must obtain an explicit grant from its Principal before acting on their behalf.

Principal:
: The human user who authorizes an Agent to act on their behalf. The Principal is the subject (sub) of any Grant Token issued by the authorization server.

Developer:
: The organization or individual who built and operates the Agent. The Developer authenticates to the authorization server using an API key.

Authorization Server:
: A server implementing this specification that issues Grant Tokens, maintains the grant registry, and provides the JWKS endpoint for offline verification.

Service:
: Any API or platform that receives requests from an Agent. Services MUST verify Grant Tokens before acting on agent requests.

Grant:
: A persistent record of permission given by a Principal to an Agent for a specific set of Scopes. A Grant is represented to the Agent as a Grant Token.

Grant Token:
: A signed JWT [RFC7519] representing a valid, non-revoked Grant. Grant Tokens are short-lived credentials carrying agent-specific claims defined in Section 5.

Scope:

A named permission string following the format resource:action[:constraint] as defined in Section 3.

DID:
A Decentralized Identifier [DID-CORE] -- the Agent's cryptographic identity. In DAAP, Agent DIDs take the form did:grantex:<agent_id>.

Policy:
A rule evaluated by the Policy Engine (Section 14) that automatically approves or denies an authorization request before the consent UI is shown to the Principal.

Anomaly:
A behavioral deviation from an agent's established activity baseline, detected by the runtime monitoring system defined in Section 15.

Budget Allocation:
A per-grant spending limit that constrains the total monetary value of actions an Agent may take under a single Grant.

2. Agent Identity

2.1. DID Format

Every Agent registered with a DAAP-compliant Authorization Server receives a Decentralized Identifier of the form:

did:grantex:<agent_id>

where <agent_id> is a ULID (Universally Unique Lexicographically Sortable Identifier) [ULID] prefixed with ag_.

Example:

did:grantex:ag_01HXYZ123abcDEF456ghi

2.2. Identity Document

The DID resolves to an identity document at the Authorization Server. The document MUST contain the following fields:

```
{
  "@context": "https://grantex.dev/v1/identity",
  "id": "did:grantex:ag_01HXYZ123abcDEF456ghi",
  "developer": "org_yourcompany",
  "name": "travel-booker",
  "description": "Books flights and hotels on behalf of users",
  "declaredScopes": ["calendar:read", "payments:initiate:max_500"],
  "status": "active",
  "createdAt": "2026-02-01T00:00:00Z",
  "verificationMethod": [{
    "id": "did:grantex:ag_01HXYZ123abcDEF456ghi#key-1",
    "type": "JsonWebKey2020",
    "publicKeyJwk": { "..." : "..." }
  }]
}
```

2.3. Key Management

Authorization Servers MUST adhere to the following key management requirements:

* Authorization Servers MUST use RS256 (RSASSA-PKCS1-v1_5 using SHA-256) [RFC7518] for signing Grant Tokens.

* Private signing keys MUST never be transmitted or stored outside the Authorization Server’s trust boundary.

* Public keys MUST be published at /.well-known/jwks.json as a JWK Set [RFC7517].

* Key rotation MUST be supported without changing the Agent’s DID or invalidating existing, unexpired Grant Tokens. Rotated keys MUST remain in the JWKS until all tokens signed with them have expired.

3. Scope Format and Registry

3.1. Format

Scopes are permission strings of the form:

resource:action[:constraint]

where:

* resource identifies the data or service being accessed (e.g., calendar, payments, email)

* action identifies the operation (e.g., read, write, send, initiate, delete)

* constraint is an optional limiting parameter (e.g., max_500 for a spending limit)

3.2. Standard Scope Registry

The following scopes constitute the normative standard registry. Implementations MUST support all standard scopes that are relevant to the resources they expose:

Scope	Description
calendar:read	Read calendar events
calendar:write	Create, modify, and delete calendar events
email:read	Read email messages
email:send	Send emails on the Principal’s behalf
email:delete	Delete email messages
files:read	Read files and documents
files:write	Create and modify files
payments:read	View payment history and balances
payments:initiate	Initiate payments of any amount
payments:initiate:max_N	Initiate payments up to N in the account’s base currency
profile:read	Read profile and identity information
contacts:read	Read address book and contacts

Table 1

3.3. Custom Scopes

Services MAY define custom scopes using reverse-domain notation per [RFC3986]:

```
com.stripe.charges:create:max_5000
io.github.issues:create
```

Custom scopes MUST use reverse-domain notation to avoid collisions with the standard registry.

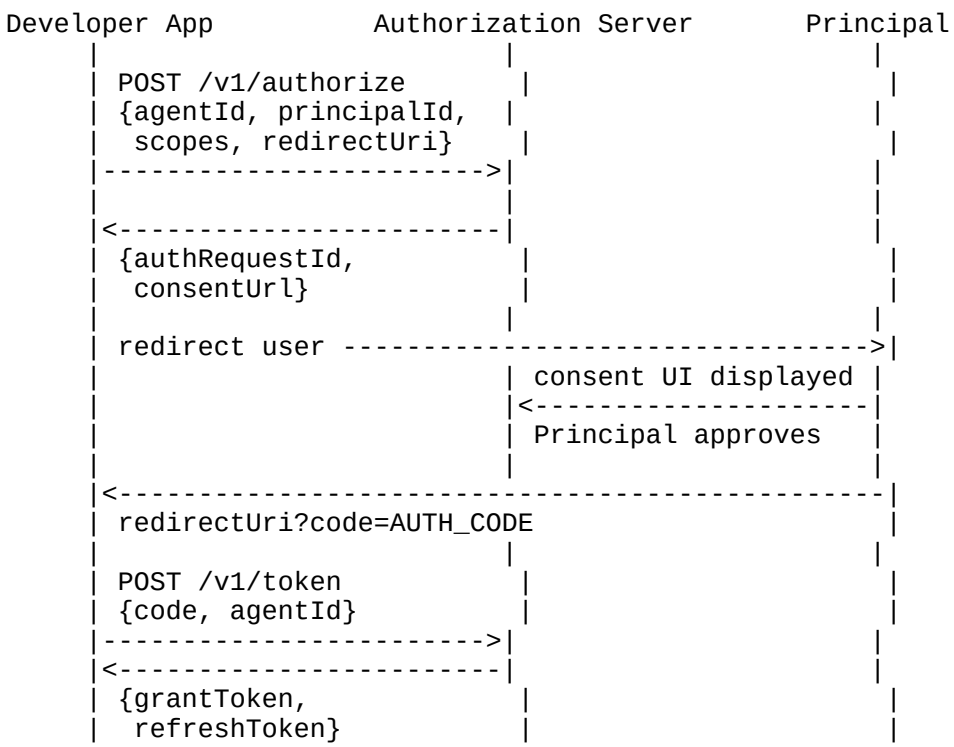
3.4. Scope Display Requirements

Authorization Servers MUST maintain a human-readable description for each scope in their registry. Consent UIs MUST display human-readable descriptions to Principals, never raw scope strings.

4. Grant Authorization Flow

4.1. Overview

The DAAP grant flow is modelled on the OAuth 2.0 Authorization Code flow [RFC6749] with the following adaptations: the client is always a Developer (identified by an API key), the resource owner is a Principal identified by the Developer’s internal user identifier, and the resulting token carries agent-specific claims.



4.2. Authorization Request

The Developer initiates the flow by sending:

```
POST /v1/authorize
Authorization: Bearer <api_key>
Content-Type: application/json

{
  "agentId": "ag_01HXYZ123abc",
  "principalId": "user_abc123",
  "scopes": ["calendar:read", "payments:initiate:max_500"],
  "expiresIn": "24h",
  "redirectUri": "https://yourapp.com/auth/callback",
  "state": "<csrf_token>",
  "audience": "https://api.targetservice.com"
}
```

The audience field is OPTIONAL. When present, it MUST be embedded as the aud claim in the issued Grant Token. The state parameter is REQUIRED and MUST be validated by the Developer's callback handler to prevent CSRF attacks. The redirectUri MUST match a URI pre-registered for the Agent at the Authorization Server.

Authorization Servers MUST reject requests whose redirectUri does not exactly match a pre-registered value for the specified agentId.

Response 200 OK:

```
{
  "authRequestId": "areq_01HXYZ...",
  "consentUrl": "https://consent.example.com/authorize?req=eyJ...",
  "expiresAt": "2026-02-01T00:15:00Z"
}
```

4.3. Consent UI Requirements

Authorization Servers MUST render a consent UI to the Principal that displays all of the following before the Principal approves or denies:

1. The Agent's registered name and description
2. The Developer's registered organization name
3. The full list of requested scopes with human-readable descriptions
4. The token expiry period
5. A prominent deny/cancel action that is at least as visually prominent as the approve action

4.4. Token Exchange

After Principal approval, the Authorization Server calls redirectUri?code=AUTH_CODE&state=STATE.

```
POST /v1/token
Authorization: Bearer <api_key>
Content-Type: application/json

{
  "code": "AUTH_CODE",
  "agentId": "ag_01HXYZ123abc"
}
```

Response 200 OK:

```
{
  "grantToken": "eyJhbGciOiJSUzI1NiJ9...",
  "refreshToken": "ref_01HXYZ...",
  "grantId": "grnt_01HXYZ...",
  "scopes": ["calendar:read", "payments:initiate:max_500"],
  "expiresAt": "2026-02-02T00:00:00Z"
}
```

Refresh tokens are single-use. The Authorization Server MUST rotate the refresh token on every use. Refresh tokens MUST be invalidated when the underlying Grant is revoked.

5. Grant Token Format

5.1. JOSE Header

```
{
  "alg": "RS256",
  "typ": "JWT",
  "kid": "<key_id>"
}
```

The alg field MUST be RS256. Authorization Servers MUST NOT issue tokens with any other algorithm. Verifiers MUST explicitly reject tokens with any alg value other than RS256, including none and HS256.

5.2. JWT Claims

```
{
  "iss": "https://as.example.com",
  "sub": "user_abc123",
  "aud": "https://api.targetservice.com",
  "agt": "did:grantex:ag_01HXYZ123abc",
  "dev": "org_yourcompany",
  "grnt": "grnt_01HXYZ...",
  "scp": ["calendar:read", "payments:initiate:max_500"],
  "bdg": 5000,
  "iat": 1709000000,
  "exp": 1709086400,
  "jti": "tok_01HXYZ987xyz"
}
```

The following claims are defined by this specification:

Claim	Type	Required	Description
iss	string	REQUIRED	Authorization Server identifier URI
sub	string	REQUIRED	Principal identifier
aud	string	OPTIONAL	Intended audience (target service URI)
agt	string	REQUIRED	Agent DID
dev	string	REQUIRED	Developer organization identifier
grnt	string	REQUIRED	Grant identifier (used for revocation lookup)
scp	string[]	REQUIRED	Array of granted scope strings
bdg	number	OPTIONAL	Remaining budget amount (see Section 10)
iat	NumericDate	REQUIRED	Issued-at time
exp	NumericDate	REQUIRED	Expiration time
jti	string	REQUIRED	Unique token identifier (for replay prevention)

Table 2

5.3. Token Validation

Services receiving a Grant Token MUST verify all of the following:

1. The token signature is valid, verified using the JWK Set published at {iss}/.well-known/jwks.json, with the key identified by kid.

2. The alg header value is RS256. Tokens with any other alg MUST be rejected.

3. The exp claim has not passed (allowing for a reasonable clock skew of no more than 300 seconds).

4. If the service has a registered audience identifier, the aud claim matches that identifier.

5. The scp array contains all scopes required for the requested operation.

6. If the bdg claim is present and the operation has a cost, the bdg value is sufficient for the operation.

7. For high-stakes operations (see Section 5.4), the token has not been revoked via the online verification endpoint.

5.4. Token Lifetime Guidance

Use Case	Recommended Maximum TTL
High-stakes actions (payments:initiate, email:send, files:write)	1 hour
Standard agent tasks	8 hours
Long-running background agents	24 hours

Table 3

Implementations caching revocation state MUST NOT cache for longer than 300 seconds (5 minutes). Services processing high-stakes scopes (payments:initiate, email:send, files:write) SHOULD perform online verification for each token use.

6. Token Revocation

6.1. Revoke a Grant

```
DELETE /v1/grants/{grantId}
Authorization: Bearer <principal_token>
```

Effect: all active Grant Tokens issued under this Grant are immediately invalidated. The Grant record is marked revoked with a timestamp.

6.2. Revoke a Specific Token

```
POST /v1/tokens/revoke
Authorization: Bearer <api_key>
Content-Type: application/json

{
  "jti": "tok_01HXYZ987xyz"
}

Response: 204 No Content.
```

6.3. Online Verification

```
POST /v1/tokens/verify
Authorization: Bearer <api_key>
Content-Type: application/json

{
  "token": "eyJhbGciOiJSUzI1NiJ9..."
}

Response 200 OK:

{
  "valid": true,
  "grantId": "grnt_01HXYZ...",
  "scopes": ["calendar:read"],
  "principal": "user_abc123",
  "agent": "did:grantex:ag_01HXYZ123abc",
  "expiresAt": "2026-02-02T00:00:00Z"
}
```

6.4. JTI Replay Prevention

Authorization Servers MUST track all issued jti values for the lifetime of the corresponding token. If a jti value is presented for verification more than once within its validity window, the Authorization Server MUST return valid: false and SHOULD log an anomaly event.

7. Audit Trail

7.1. Log Entry Schema

```
{
  "entryId": "alog_01HXYZ...",
  "agentId": "did:grantex:ag_01HXYZ123abc",
  "grantId": "grnt_01HXYZ...",
  "principalId": "user_abc123",
  "developerId": "org_yourcompany",
  "action": "payment.initiated",
  "status": "success",
  "metadata": {
    "amount": 420,
    "currency": "USD",
    "merchant": "Air India"
  },
  "timestamp": "2026-02-01T12:34:56.789Z",
  "hash": "sha256:abc123...",
  "prevHash": "sha256:xyz789..."
}
```

action values use the format resource.verb (e.g., payment.initiated, email.sent). status MUST be one of success, failure, or blocked.

7.2. Hash Chain

Each entry's hash is computed as a SHA-256 digest [RFC4648] over a canonical representation of the entry:

hash = SHA-256(canonical_json(entry_without_hash) || prevHash)

where canonical_json serializes all fields as a JSON object [RFC8259] with keys sorted alphabetically, and prevHash is the null string for the first entry in a chain. This construction makes any retrospective modification to a historical entry detectable, as it invalidates all subsequent hashes.

7.3. Audit Log Requirements

* Audit log entries MUST be append-only at the API level. No update or delete endpoints for audit entries are permitted.

* The Authorization Server MUST reject requests to modify or delete audit entries.

* The complete audit log for a Grant MUST remain accessible after the Grant is revoked, for a minimum retention period determined by the deployment's compliance requirements.

8. Multi-Agent Delegation

8.1. Delegation Token Claims

When Agent A spawns Agent B, B's Grant Token MUST carry delegation claims linking it to the original Principal's authorization:

```
{
  "sub": "user_abc123",
  "agt": "did:grantex:ag_B_456",
  "parentAgt": "did:grantex:ag_A_123",
  "parentGrnt": "grnt_parentXYZ",
  "scp": ["email:read"],
  "delegationDepth": 1
}
```

Additional delegation claims:

Claim	Type	Description
parentAgt	string	DID of the delegating (parent) Agent
parentGrnt	string	Grant ID of the parent Grant
delegationDepth	integer	Number of hops from the root Grant; 0 for root Grants

Table 4

8.2. Delegation Rules

* Sub-agent scopes MUST be a strict subset of the parent Grant's scp array. Authorization Servers MUST reject delegation requests whose requested scopes are not fully contained in the parent token's scp claim.

* delegationDepth MUST be incremented by exactly 1 at each hop.

* Implementations MUST enforce a developer-configurable delegation depth limit. The RECOMMENDED default limit is *3*. Implementations MUST enforce a hard cap of *10* regardless of developer configuration.

* The expiry of a delegated Grant Token MUST NOT exceed min(parent_token_exp, now + requested_expires_in).

8.3. Delegation Endpoint

```
POST /v1/grants/delegate
Authorization: Bearer <api_key>
Content-Type: application/json

{
  "parentGrantToken": "eyJhbGciOiJSUzI1NiJ9...",
  "subAgentId": "ag_01HXYZ_sub",
  "scopes": ["email:read"],
  "expiresIn": "1h"
}
```

The Authorization Server MUST:

1. Validate that the parent Grant has not been revoked.

2. Reject with 400 if any requested scope is not present in the parent token’s scp claim.

3. Reject with 400 if the resulting delegationDepth would exceed the configured limit.

4. Reject with 404 if subAgentId does not belong to the authenticated Developer.

5. Compute expiry as min(parent token exp, now + expiresIn).

Response 201 Created:

```
{
  "grantToken": "eyJhbGciOiJSUzI1NiJ9...",
  "grantId": "grnt_01HXYZ_sub",
  "scopes": ["email:read"],
  "expiresAt": "2026-02-01T01:00:00Z"
}
```

8.4. Cascade Revocation

Revoking a Grant via DELETE /v1/grants/:id MUST atomically revoke all descendant Grants -- that is, all Grants whose parent_grant_id traces back to the revoked Grant at any depth. Authorization Servers SHOULD implement this as a single recursive database transaction to eliminate any window during which descendant tokens remain valid.

9. Conformance Requirements

A conformant DAAP Authorization Server MUST expose the following endpoints:

Endpoint	Description
POST /v1/agents	Register an Agent
POST /v1/authorize	Initiate the grant authorization flow
POST /v1/token	Exchange authorization code for Grant Token
POST /v1/tokens/verify	Online token verification
POST /v1/tokens/revoke	Revoke a specific token by JTI
GET /v1/grants	List a Principal’s active Grants
GET /v1/grants/:id	Retrieve a single Grant
DELETE /v1/grants/:id	Revoke a Grant (cascades to all descendants)
POST /v1/grants/delegate	Issue a delegated sub-agent Grant
POST /v1/audit/log	Write an audit log entry
GET /v1/audit/entries	Query the audit log
GET /v1/audit/:id	Retrieve a single audit log entry
GET /.well-known/jwks.json	JWK Set for offline token verification
GET /health	Health check

Table 5

The following endpoints are OPTIONAL. Implementations that choose to support an optional extension MUST implement it as specified in this document:

* *Policy Engine*: POST /v1/policies, GET /v1/policies, GET /v1/policies/:id, PATCH /v1/policies/:id, DELETE /v1/policies/:id

* *Webhooks*: POST /v1/webhooks, GET /v1/webhooks, DELETE /v1/webhooks/:id
* *Anomaly Detection*: POST /v1/anomalies/detect, GET /v1/anomalies, PATCH /v1/anomalies/:id/acknowledge
* *Enterprise SCIM 2.0*: /scim/v2/ endpoints as defined in RFC 7642/7643/7644
* *SSO (OIDC)*: POST /v1/sso/config, GET /sso/login, GET /sso/callback
* *Budget Controls*: POST /v1/budget/allocate, POST /v1/budget/debit, GET /v1/budget/balance/:grantId, GET /v1/budget/transactions/:grantId (see Section 10)
* *Event Streaming*: GET /v1/events/stream (SSE), GET /v1/events/ws (WebSocket) (see Section 11)
* *Credential Vault*: POST /v1/vault/credentials, GET /v1/vault/credentials, DELETE /v1/vault/credentials/:id, POST /v1/vault/exchange (see Section 12)
* *External Policy Backends*: OPA (POST /v1/data/grantex/authz) and Cedar (POST /v1/is_authorized) integration (see Section 13)

10. Budget Controls

10.1. Purpose

Budget Controls provide per-grant spending limits that constrain the total monetary value of actions an Agent may perform under a single Grant. This extension is critical for agents operating in financial contexts (e.g., payments:initiate) where unconstrained spending could cause irreversible harm.

10.2. Budget Allocation

A Developer allocates a budget to a Grant by sending:

```
POST /v1/budget/allocate
Authorization: Bearer <api_key>
Content-Type: application/json

{
  "grantId": "grnt_01HXYZ...",
  "amount": 10000,
  "currency": "USD"
}

Response 201 Created:

{
  "id": "bdgt_01HXYZ...",
  "grantId": "grnt_01HXYZ...",
  "initialBudget": 10000,
  "remainingBudget": 10000,
  "currency": "USD",
  "createdAt": "2026-02-01T00:00:00Z"
}
```

A Grant MUST NOT have more than one active budget allocation. Authorization Servers MUST reject allocation requests for Grants that already have an active allocation.

10.3. Budget Debit

Services debit from a Grant’s budget using an atomic operation:

```
POST /v1/budget/debit
Authorization: Bearer <api_key>
Content-Type: application/json

{
  "grantId": "grnt_01HXYZ...",
  "amount": 250,
  "description": "Flight booking - DEL to BOM",
  "metadata": { "merchant": "Air India" }
}

Response 200 OK:

{
  "remaining": 9750,
  "transactionId": "btxn_01HXYZ..."
}
```

Authorization Servers MUST implement budget debit as an atomic operation (e.g., UPDATE ... WHERE remaining >= amount). If the remaining budget is insufficient, the Authorization Server MUST respond with 402 Payment Required and the error code INSUFFICIENT_BUDGET.

10.4. The bdg JWT Claim

When a Grant has an active budget allocation, the Authorization Server SHOULD include the bdg claim in issued Grant Tokens. The value MUST be the remaining budget amount at the time of token issuance.

Services receiving a Grant Token with a bdg claim MAY use it for local pre-flight budget checks. However, the bdg claim is advisory -- the atomic debit endpoint remains the authoritative mechanism for budget enforcement.

10.5. Threshold Alerts

Authorization Servers implementing Budget Controls MUST emit events when budget utilization crosses predefined thresholds:

Threshold	Event Type	Description
50% consumed	budget.threshold	Warning: half of the budget has been consumed
80% consumed	budget.threshold	Alert: budget is running low
100% consumed	budget.exhausted	Budget fully consumed; subsequent debits will fail

Table 6

These events MUST be delivered via the configured webhook endpoints and SHOULD be delivered via the event streaming endpoints (Section 11) if the extension is supported.

10.6. Budget Balance and Transaction History

```
GET /v1/budget/balance/{grantId}
Authorization: Bearer <api_key>
```

Returns the current BudgetAllocation object.

```
GET /v1/budget/transactions/{grantId}
Authorization: Bearer <api_key>
```

Returns a paginated list of all debit transactions for the specified Grant's budget.

11. Event Streaming

11.1. Purpose

Event Streaming provides real-time delivery of authorization lifecycle events to connected clients. This extension complements webhooks by offering a persistent connection model suitable for dashboards, monitoring systems, and real-time alerting.

11.2. Event Types

Event Type	Description
grant.created	A new Grant has been issued
grant.revoked	A Grant has been revoked
token.issued	A new Grant Token has been issued
budget.threshold	A budget threshold has been crossed
budget.exhausted	A budget has been fully consumed

Table 7

11.3. SSE Endpoint

```
GET /v1/events/stream
Authorization: Bearer <api_key>
Accept: text/event-stream
```

The Authorization Server MUST implement Server-Sent Events [SSE] delivery. Events are formatted as:

```
event: grant.created
data: {"grantId":"grnt_01HXYZ...","agentId":"ag_01HXYZ...","timestamp":"2026-02-01T00:00:
00Z"}

event: budget.threshold
data: {"grantId":"grnt_01HXYZ...","threshold":80,"remaining":2000,"timestamp":"2026-02-01
T01:00:00Z"}
```

11.4. WebSocket Endpoint

```
GET /v1/events/ws
Upgrade: websocket
Connection: Upgrade
Authorization: Bearer <api_key>
```

WebSocket connections receive the same event payloads as SSE, serialized as JSON messages. The Authorization Server MUST send periodic ping frames (RECOMMENDED interval: 30 seconds) to detect stale connections.

11.5. Connection Limits

Authorization Servers MUST enforce a maximum number of concurrent event streaming connections per Developer. The RECOMMENDED limit is *5* concurrent connections. When the limit is exceeded, the Authorization Server MUST reject new connections with 429 Too Many Requests.

11.6. Event Delivery Guarantees

Event Streaming provides at-most-once delivery semantics. For guaranteed delivery, Developers SHOULD use webhooks, which provide at-least-once delivery with persistent retry.

12. Credential Vault

12.1. Purpose

The Credential Vault provides encrypted per-user credential storage, enabling a token-to-credential exchange pattern where an Agent presents a valid Grant Token and receives the associated service credentials in return. This eliminates the need for Agents to store long-lived secrets directly.

12.2. Credential Storage

```
POST /v1/vault/credentials
Authorization: Bearer <api_key>
Content-Type: application/json

{
  "principalId": "user_abc123",
  "service": "stripe",
  "credentials": {
    "apiKey": "sk_live_...",
    "webhookSecret": "whsec_..."
  },
  "scopes": ["payments:initiate"]
}
```

Credentials MUST be encrypted at rest using AES-256-GCM or an equivalent authenticated encryption algorithm. The encryption key MUST be derived from a key management system (KMS) that is separate from the database storing the ciphertext.

12.3. Credential Exchange

An Agent exchanges a valid Grant Token for the associated credentials:

```
POST /v1/vault/exchange
Authorization: Bearer <grant_token>
Content-Type: application/json

{
  "service": "stripe"
}
```

The Authorization Server MUST verify the Grant Token before returning credentials. The response MUST only include credentials whose required scopes are a subset of the Grant Token’s scp claim.

12.4. Credential Lifecycle

* Credentials MUST be automatically deleted when the associated Principal revokes all Grants for the associated Agent.

* Developers MAY delete credentials at any time via DELETE /v1/vault/credentials/:id.

* The Authorization Server MUST log all credential access events in the audit trail.

13. External Policy Backends

13.1. Purpose

While the built-in Policy Engine (Section 14) is sufficient for simple allow/deny rules, production deployments often require integration with dedicated policy decision points (PDPs) that support richer policy languages. This extension defines the integration pattern for two external policy backends: Open Policy Agent (OPA) [OPA] and Cedar [CEDAR].

13.2. Policy Evaluation Context

Authorization Servers MUST construct the following evaluation context and send it to the configured external backend:

```
{
  "subject": {
    "type": "agent",
    "id": "did:grantex:ag_01HXYZ123abc",
    "developer": "org_yourcompany"
  },
  "resource": {
    "type": "grant",
    "scopes": ["calendar:read", "payments:initiate:max_500"]
  },
  "action": {
    "name": "authorize"
  },
  "context": {
    "principalId": "user_abc123",
    "timestamp": "2026-02-01T12:00:00Z",
    "ipAddress": "203.0.113.42"
  }
}
```

This evaluation context is intentionally aligned with the OpenID AuthZEN [AUTHZEN] subject/resource/action/context model.

13.3. OPA Integration

Authorization Servers supporting OPA MUST send a POST request to the configured OPA endpoint:

```
POST {OPA_URL}/v1/data/grantex/authz
Content-Type: application/json

{
  "input": { ... evaluation context ... }
}
```

The OPA response MUST contain:

```
{
  "result": {
    "allow": true
  }
}
```

If result.allow is false, the Authorization Server MUST deny the authorization request. OPA policies are written in Rego.

13.4. Cedar Integration

Authorization Servers supporting Cedar MUST send a POST request to the configured Cedar endpoint:

```
POST {CEDAR_URL}/v1/is_authorized
Content-Type: application/json

{
  "principal": "Agent::\"did:grantex:ag_01HXYZ123abc\"",
  "action": "Action::\"authorize\"",
  "resource": "Grant::\"grnt_01HXYZ...\"",
  "context": { ... }
}
```

The Cedar response MUST contain:

```
{
  "decision": "Allow"
}
```

If decision is "Deny", the Authorization Server MUST deny the authorization request.

13.5. Timeout and Fallback

Authorization Servers MUST enforce a timeout on external policy backend requests. The RECOMMENDED timeout is *5 seconds*. If the backend does not respond within the timeout, the Authorization Server MUST apply a configurable fallback policy:

Fallback Mode	Behavior
deny (default)	Deny the authorization request
allow	Allow the authorization request (use with caution)
builtin	Fall back to the built-in Policy Engine

Table 8

The fallback mode MUST be configurable by the Developer. The default MUST be deny (fail-closed).

14. Policy Engine

14.1. Purpose

The Policy Engine evaluates developer-defined rules against each authorization request before the consent UI is displayed. Policies enable developers to auto-approve routine low-risk requests and auto-deny requests that violate organizational constraints.

14.2. Effects

Effect	Description
auto_approve	Grant Token issued immediately without showing the consent UI
auto_deny	Authorization request rejected immediately with 403 Forbidden

Table 9

14.3. Condition Fields

Field	Type	Description
scopes	string[]	Matches when the requested scopes are a subset of this list

principalId	string	Matches a specific Principal identifier
agentId	string	Matches a specific Agent identifier
timeWindow	object	Time constraint: { "startHour": N, "endHour": N, "days": [1..7] } where days are ISO weekday integers (1=Monday, 7=Sunday)

Table 10

14.4. Evaluation Order

Policy evaluation MUST follow this order:

1. If an external policy backend (Section 13) is configured, the evaluation context is sent to the backend first. If the backend returns a decision, that decision is final.

2. auto_deny rules are evaluated first. The first matching deny rule wins and the request is rejected immediately.

3. auto_approve rules are evaluated next. The first matching allow rule causes the Grant Token to be issued.

4. If no rule matches, the consent UI is displayed to the Principal.

This ordering ensures that restrictive policies cannot be bypassed by a conflicting allow rule.

15. Anomaly Detection

15.1. Purpose

The anomaly detection system monitors Agent behavior at runtime against each Agent’s established activity baseline. It identifies behavioral deviations and surfaces them to Developers for review.

15.2. Non-Blocking Requirement

Anomaly detection MUST NOT block token issuance. Detection operates asynchronously as an advisory layer. Authorization Servers MUST NOT delay Grant Token responses pending anomaly analysis.

15.3. Anomaly Types

Type	Description
unusual_scope_access	Agent requested scopes outside its established pattern
high_frequency	Token issuance rate significantly exceeds the agent’s baseline
off_hours_activity	Activity detected outside the Principal’s normal active hours
new_principal	Agent is requesting access for a previously unserved Principal
cascade_delegation	Delegation chain depth approaching or exceeding configured limits

Table 11

15.4. Severity Levels

Anomaly severity MUST be one of: low, medium, high, critical.

16. Security Considerations

16.1. Algorithm Restrictions

All Grant Tokens MUST be signed with RS256 (RSASSA-PKCS1-v1_5 with SHA-256). Symmetric signing algorithms, including HS256, are NOT PERMITTED. All verifiers MUST explicitly reject tokens presenting alg: none or any symmetric algorithm, regardless of library defaults. This prevents algorithm confusion attacks as described in [RFC8725].

RSA key moduli MUST be at least 2048 bits. Authorization Servers generating or importing signing keys MUST enforce this minimum.

16.2. Token Replay Prevention

Every issued Grant Token carries a unique jti claim. Authorization Servers providing online verification MUST track issued jti values and reject any verification request presenting a jti that has already been used, for the full lifetime of the token.

16.3. CSRF and Redirect URI Security

The state parameter in the authorization request MUST be a cryptographically random, unpredictable value generated per-request. Developer callback handlers MUST validate the returned state value against the value sent in the original request.

Redirect URIs MUST be pre-registered by the Developer for each Agent. Authorization Servers MUST perform exact-match comparison of the redirectUri in each authorization request against the pre-registered set. Prefix matching and wildcard matching are NOT PERMITTED.

16.4. Scope Reduction for Delegation

Delegated Grant Tokens MUST carry a scope set that is a strict subset of the parent Grant's scope set. Authorization Servers MUST enforce this at token issuance time; it MUST NOT be enforced only at verification time.

16.5. Revocation Propagation

Authorization Servers MUST propagate grant revocation to all descendant grants atomically. The maximum allowable latency between a revocation request and the invalidation of all descendant tokens via the online verification endpoint is implementation-defined, but implementations SHOULD target sub-second propagation.

Implementations caching revocation state MUST NOT cache for longer than 300 seconds.

16.6. Consent UI Integrity

Consent UIs MUST display the agent name, developer name, all requested scopes with human-readable descriptions, token expiry, and a prominent deny/cancel action. This information MUST be sourced from the Authorization Server's registry, not from the authorization request itself, to prevent a malicious Developer from displaying misleading scope descriptions.

16.7. Audit Log Integrity

The hash-chain construction defined in Section 7 ensures that any modification to a historical audit entry is detectable. Implementations MUST store audit log entries in an append-only manner and MUST expose no API for modification or deletion of audit entries. Audit log export implementations SHOULD verify the hash chain before serving exports.

16.8. Enterprise Identity Security

SSO callback handlers MUST validate both the state (CSRF protection) and nonce (replay protection) parameters before establishing a session. ID tokens received in the SSO callback MUST be cryptographically verified against the identity provider's JWKS endpoint before any claims are trusted.

SCIM provisioning endpoints MUST authenticate via a credential (SCIM Bearer token) that is entirely separate from the Developer API key infrastructure. Compromise of a Developer API key MUST NOT grant access to SCIM provisioning endpoints, and vice versa.

16.9. Budget Security

Budget debit operations MUST be implemented as atomic database operations (e.g., UPDATE ... SET remaining = remaining - amount WHERE remaining >= amount). Non-atomic implementations risk race conditions that could allow spending beyond the allocated budget.

The bdg JWT claim is advisory and MUST NOT be used as the sole mechanism for budget enforcement. Services MUST use the atomic debit endpoint for authoritative budget checks.

16.10. Credential Vault Security

Credentials stored in the Vault MUST be encrypted at rest using authenticated encryption (AES-256-GCM or equivalent). Encryption keys MUST be managed by a dedicated KMS and MUST NOT be stored alongside the ciphertext.

All credential access events (store, retrieve, exchange, delete) MUST be logged in the audit trail with the credential identifier but NOT the credential value.

17. IANA Considerations

17.1. JWT Claims Registration

This document requests registration of the following claims in the IANA "JSON Web Token Claims" registry established by [RFC7519]:

agt:
Claim Name: agt

Claim Description: Agent Decentralized Identifier

Change Controller: IETF

Specification Document: This document, Section 5

dev:
Claim Name: dev

Claim Description: Developer organization identifier

Change Controller: IETF

Specification Document: This document, Section 5

grnt:
Claim Name: grnt

Claim Description: Grant identifier for revocation lookup

Change Controller: IETF

Specification Document: This document, Section 5

scp:
Claim Name: scp

Claim Description: Array of granted authorization scope strings

Change Controller: IETF

Specification Document: This document, Section 5

bdg:
Claim Name: bdg

Claim Description: Remaining budget amount for the Grant

Change Controller: IETF

Specification Document: This document, Section 10

parentAgt:
Claim Name: parentAgt

Claim Description: DID of the delegating parent Agent

Change Controller: IETF

Specification Document: This document, Section 8

parentGrnt:
Claim Name: parentGrnt

Claim Description: Grant identifier of the parent Grant

Change Controller: IETF

Specification Document: This document, Section 8

delegationDepth:
Claim Name: delegationDepth

Claim Description: Number of delegation hops from the root Grant

Change Controller: IETF

Specification Document: This document, Section 8

17.2. Well-Known URI Registration

No new Well-Known URIs are defined by this specification. Implementations use the existing /.well-known/jwks.json path established by [RFC8414].

18. References

18.1. 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>.

[RFC3986] Berners-Lee, T., Fielding, R., and L. Masinter, "Uniform Resource Identifier (URI): Generic Syntax", STD 66, RFC 3986, DOI 10.17487/RFC3986, January 2005, <https://www.rfc-editor.org/rfc/rfc3986>.

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

[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>.

[RFC7517] Jones, M., "JSON Web Key (JWK)", RFC 7517, DOI 10.17487/RFC7517, May 2015, <https://www.rfc-editor.org/rfc/rfc7517>.

[RFC7518] Jones, M., "JSON Web Algorithms (JWA)", RFC 7518, DOI 10.17487/RFC7518, May 2015, <https://www.rfc-editor.org/rfc/rfc7518>.

[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>.

[RFC8259] Bray, T., Ed., "The JavaScript Object Notation (JSON) Data Interchange Format", STD 90, RFC 8259, DOI 10.17487/RFC8259, December 2017, <https://www.rfc-editor.org/rfc/rfc8259>.

[RFC8414] Jones, M., Sakimura, N., and J. Bradley, "OAuth 2.0 Authorization Server Metadata", RFC 8414, DOI 10.17487/RFC8414, June 2018, <https://www.rfc-editor.org/rfc/rfc8414>.

18.2. Informative References

[AUTHZEN] OpenID Foundation, "OpenID AuthZEN Authorization API", 2024, <https://openid.net/specs/openid-authzen-authorization-api-1_0.html>.

[CEDAR] Amazon Web Services, "Cedar Policy Language", 2024, <https://www.cedarpolicy.com/>.

[DID-CORE] W3C, "Decentralized Identifiers (DIDs) v1.0", 19 July 2022, <https://www.w3.org/TR/did-core/>.

[OPA] Styra, "Open Policy Agent", 2024, <https://www.openpolicyagent.org/docs/latest/>.

[RFC7662] Richer, J., Ed., "OAuth 2.0 Token Introspection", RFC 7662, DOI 10.17487/RFC7662, October 2015, <https://www.rfc-editor.org/rfc/rfc7662>.

[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>.

[RFC8725] Sheffer, Y., Hardt, D., and M. Jones, "JSON Web Token Best Current Practices", BCP 225, RFC 8725, DOI 10.17487/RFC8725, February 2020, <https://www.rfc-editor.org/rfc/rfc8725>.

[SSE] WHATWG, "Server-Sent Events", 2024, <https://html.spec.whatwg.org/multipage/server-sent-events.html>.

[ULID] Feerasta, A., "Universally Unique Lexicographically Sortable Identifier", 2016, <https://github.com/ulid/spec>.

Appendix A. Comparison with OAuth 2.0 Extensions

DAAP shares OAuth 2.0's fundamental grant model but differs in the following respects:

versus RFC 6749 (OAuth 2.0): OAuth 2.0 defines a general-purpose delegated authorization framework. DAAP specializes this for AI agents by: adding cryptographic agent identity (DID); defining agent-specific JWT claims (agt, dev, grnt, scp, bdg); mandating RS256 exclusively; and adding the delegation, audit, policy, anomaly detection, budget controls, event streaming, credential vault, and external policy backend subsystems.

versus RFC 8693 (Token Exchange): Token Exchange [RFC8693] enables a client to exchange one token for another, including impersonation and delegation use cases. DAAP's delegation model serves a narrower purpose -- chaining AI agent sub-authorizations back to a human principal -- and adds depth-limiting and cascade revocation semantics not present in RFC 8693.

versus RFC 7662 (Token Introspection): Token Introspection [RFC7662] defines an endpoint for resource servers to query token metadata. DAAP's /v1/tokens/verify endpoint serves a similar purpose but returns DAAP-specific fields (agent, principal, scopes) and is used by agent-side SDKs rather than resource servers.

Appendix B. Implementation Report

This appendix documents the conformance status of the reference implementation and SDK coverage as of March 2026. A detailed implementation report is available at docs/ietf-draft/implementation-report.md in the Grantex repository.

B.1. Reference Authorization Server

The Grantex authorization server (Fastify + PostgreSQL + Redis) implements all REQUIRED endpoints and the following OPTIONAL extensions:

Extension	Status	Notes
Policy Engine	Implemented	Built-in + OPA + Cedar backends
Webhooks	Implemented	Persistent retry with exponential backoff
Anomaly Detection	Implemented	5 anomaly types, async worker
SCIM 2.0	Implemented	Full RFC 7643 compliance
SSO (OIDC)	Implemented	Authorization Code + PKCE
Budget Controls	Implemented	Atomic debit, threshold alerts
Event Streaming	Implemented	SSE + WebSocket
Credential Vault	Implemented	AES-256-GCM encryption
External Policy Backends	Implemented	OPA + Cedar with timeout/fallback

Table 12

~362 automated tests pass across all features.

B.2. SDK Coverage

SDK	Language	Version	Core	Budget	Events	Vault
@grantex/sdk	TypeScript	0.2.0	Full	Full	Full	Full
grantex	Python	0.2.0	Full	Full	Full	Full
grantex-go	Go	0.1.2	Full	Full	Full	Full

Table 13

All SDKs pass the @grantex/conformance test suite.

Acknowledgements

The authors thank the members of the IETF OAuth Working Group for prior art in delegated authorization, and the W3C Decentralized Identifier Working Group for the DID specification that DAAP builds upon for agent identity.

Author’s Address

Sanjeev Kumar
Grantex
Email: mishra.sanjeev@gmail.com
URI: https://grantex.dev