Web Authorization Protocol
Internet-Draft
Intended status: Standards Track
Expires: 1 November 2026

K. McGuinness
Independent

30 April 2026

OAuth Actor Profile for Delegation
draft-mcguinness-oauth-actor-profile-00

Abstract

OAuth deployments increasingly involve agents and workloads acting on behalf of human users across organizational boundaries. Existing specifications provide relevant building blocks (notably the act claim from RFC 8693 Token Exchange) but do not define a consistent profile for representing delegated actor relationships across JWT assertion grants (RFC 7523), JWT access tokens (RFC 9068), and Transaction Tokens, nor for classifying actor entity types or signaling support between authorization servers and resource servers. The result is inconsistent actor representation and actor-representation interoperability gaps that force deployments to rely on proprietary conventions.

This document defines the OAuth Actor Profile for Delegation. It specifies a common act claim structure extended with sub_profile for entity-type classification, processing rules for authorization servers and resource servers across the three token families and their Token Exchange inputs, and OAuth discovery metadata parameters for advertising actor-profile support. The profile applies uniformly across token types and integrates with existing sender-constraint mechanisms (DPoP, mTLS). It does not standardize the policies by which systems determine whether a given actor is permitted to act for a subject; those decisions remain deployment-specific.

About This Document

This note is to be removed before publishing as an RFC.

The latest revision of this draft can be found at https://mcguinness.github.io/draft-mcguinness-oauth-actor-profile/draft-mcguinness-oauth-actor-profile.html. Status information for this document may be found at https://datatracker.ietf.org/doc/draft-mcguinness-oauth-actor-profile/.

Discussion of this document takes place on the Web Authorization Protocol Working Group mailing list (mailto:oauth@ietf.org), which is archived at https://mailarchive.ietf.org/arch/browse/oauth/. Subscribe at https://www.ietf.org/mailman/listinfo/oauth/.

Source for this draft and an issue tracker can be found at https://github.com/mcguinness/draft-mcguinness-oauth-actor-profile.

Status of This Memo

Copyright Notice

Table of Contents

1. Introduction

When an agent acts on behalf of a user across trust domains, every system in the path needs to know who authorized the request and who is making it. Existing specifications provide relevant building blocks ([RFC8693] introduced the act claim for Token Exchange) but do not define a consistent, interoperable way to represent delegated actor relationships across JWT assertion grants, JWT access tokens, and Transaction Tokens, nor how common Token Exchange input credentials should feed that representation.

Several actor-representation interoperability gaps result from the absence of such a profile:

* *No standard entity classification.* The sub claim is routinely overloaded to represent heterogeneous entity types (end users, service accounts, AI agents, and workloads) without a standard classification mechanism, preventing deterministic cross-domain policy evaluation.

* *Inconsistent actor representation across token types.* Actor context, including key material associated with the acting party, has no standard representation that survives token transformation, as JWT assertion grants, JWT access tokens, and Transaction Tokens are specified in separate documents with differing claim conventions.

* *Implicit delegation via client identity.* Many deployments address actor representation through implicit delegation, inferring the acting party from the OAuth client identity (client_id or azp); this approach does not generalize to deployments where a single client registration fronts multiple agents, where requests pass through intermediary services, or where tokens cross organizational trust boundaries, because the client registration does not uniquely identify the runtime actor in those cases.

* *No discovery for actor-profile support.* Neither AS metadata [RFC8414] nor Protected Resource Metadata [RFC9728] define parameters for advertising actor-profile support, requiring bilateral out-of-band configuration that does not scale to environments where clients dynamically discover services across trust domains.

The design center of this document is delegation clarity. client_id identifies the OAuth client registration, sub identifies the authorizing principal, and act.sub identifies the actor exercising that authorization. These are distinct concepts. This profile makes the actor explicit in the token rather than leaving it to be inferred from client registration context, and it does so without redefining client identity or top-level subject semantics.

This document addresses that gap by specifying:

* A common actor profile structure that reuses act from [RFC8693] and adds sub_profile for entity-type classification.

* Processing rules for JWT assertion grants, JWT access tokens, and Transaction Tokens, covering Token Exchange inputs and outputs.

* A Token Exchange migration model that lets deployments move from bearer-style inputs to proof-of-possession outputs while preserving stable actor semantics.

* Resource-server guidance for evaluating delegated access using the (sub, outermost act.sub) pair.

* Integration with OAuth Entity Profiles and discovery metadata so actor classification and capability signaling can be used consistently across deployments.

* A stable extension model for companion profiles that add supplemental delegation data without altering core claim semantics.

The mechanisms are general-purpose and apply beyond AI agent scenarios. This document is a profile and extension of existing OAuth building blocks; unless stated otherwise, the requirements of [RFC8693], [RFC9068], [RFC9449], and [I-D.ietf-oauth-transaction-tokens] continue to apply.

1.1. Illustrative Use Case

Alice authorizes an AI agent to book a business trip on her behalf, and the request crosses from the enterprise identity provider's domain into an external booking domain and then into the booking provider's internal service mesh. The enterprise authorization server first issues a delegated credential that keeps Alice as sub and the agent as act; downstream issuers later transform that credential into an access token and, for the internal tool hop, a Transaction Token rebound to the booking tool as the new presenter. Across those steps, the subject remains Alice, the immediate actor changes only when a new presenter is explicitly established, and each trust-domain boundary re-issues the token under local control. The cross-domain example (Appendix B) provides the full end-to-end walkthrough.

1.2. Relationship to Related Work

OAuth Token Exchange ([RFC8693]): This document profiles the act claim from [RFC8693]. The actor object structure and processing rules supplement, and do not replace, the base Token Exchange requirements.

Identity Chaining ([I-D.ietf-oauth-identity-chaining]): Identity Chaining addresses cross-domain subject-identity propagation; this document addresses actor representation within those same tokens. The two are complementary and designed to be used together.

Identity Assertion JWT Authorization Grant ([I-D.ietf-oauth-identity-assertion-authz-grant]): ID-JAG defines how an IdP issues a JWT authorization grant via Token Exchange and how a downstream AS consumes it. ID-JAG permits actor_token inputs but leaves actor-delegation validation and resulting act claims to future profiles or extensions. This document defines one such profile: when an implementation uses ID-JAG with actor-profile claims, the Token Exchange and JWT assertion-grant processing rules in this document apply. See the cross-domain example (Appendix B) for an end-to-end walkthrough.

OAuth Entity Profiles ([I-D.mora-oauth-entity-profiles]): Defines sub_profile, client_profile, entity_profiles_supported, and the entity profile registry. This document consumes those mechanisms for actor classification and makes no independent registry requests.

Transaction Tokens ([I-D.ietf-oauth-transaction-tokens]): This document extends the Transaction Token claim model with actor-profile support and adds actor-profile-specific TTS processing rules. Base Transaction Token requirements continue to apply.

WIMSE Workload Identity ([I-D.ietf-wimse-workload-creds][I-D.ietf-wimse-wpt]): Defines workload credentials used to authenticate workloads at token endpoints. This document is mechanism-agnostic; the cross-domain example (Appendix B) illustrates a WIMSE-based TTS presenter binding.

2. Conventions and Definitions

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in BCP 14 [RFC2119] [RFC8174] when, and only when, they appear in all capitals, as shown here.

Unless otherwise specified, OAuth terms such as client, authorization server, resource server, access token, refresh token, grant, Transaction Token, and Transaction Token Service (TTS) are used as defined in [RFC6749], [RFC8693], [I-D.ietf-oauth-transaction-tokens], and related specifications.

The following terms are used in this document:

Actor: The party that is actively making a request. When delegation is present, the actor is distinct from the subject; the subject is the principal on whose behalf the actor is acting.

Subject: The principal whose authorization is being exercised. In a delegated token, the subject is the original authorizing party (e.g., an end-user or an upstream service), not the party making the immediate network request.

Delegation: The act by which a principal authorizes another party (the actor) to exercise a subset of the principal’s rights.

Cross-Domain Delegation: A delegation scenario in which the subject and actor are governed by different trust domains or different identifier namespaces under deployment policy. This document does not define a universal interoperable algorithm for classifying a particular token instance as cross-domain. Deployments typically make that determination from issuer context, actor identifier context, and the applicable trust framework or bilateral agreement. The token’s top-level iss alone is not sufficient to make that determination in all deployments.

Actor Authorization at the Resource Server: An authorization policy evaluation that considers both the subject and the actor, and the relationship between them, as policy inputs. Under this profile, the relevant actor is ordinarily the outermost actor.

Delegation Chain: The sequence of actors representing how authorization has flowed from the subject principal (sub) to the first actor (innermost act) through any intermediate parties to the immediate actor (outermost act.sub). The chain is conveyed structurally as the nested act claim.

Outermost Actor: The act object at the top level of the delegation chain (the one not nested inside any other act object). When a delegation chain of depth greater than one is present, the outermost actor identifies the immediate bearer of the token.

Local Policy: Deployment-specific rules, configurations, or decisions made by an individual AS, RS, or organization that are not specified by this document. Local policy MAY include delegation approval rules, scope-reduction algorithms, actor-identifier namespace mappings, and entity-profile acceptance criteria. When this document references local policy, the specific decision logic is intentionally not standardized.

Identifier Reconciliation: The process by which an AS or RS applies locally configured mapping rules to determine whether two identifiers from different claims or namespaces refer to the same underlying entity. Identifier reconciliation is performed by applying configured mapping rules; string similarity, shared naming patterns, or deployment intuition are not sufficient bases for reconciliation. When this document states that an implementation SHOULD perform identifier reconciliation, it SHOULD apply its configured mapping rules to determine whether the two identifiers are known to refer to the same entity. When no applicable mapping rule exists or reconciliation cannot be established, the identifiers MUST be treated as distinct.

Examples in this document are illustrative and focus on actor-profile-related claims and processing. They may omit unrelated claims, parameters, or validation steps required by the underlying specifications for a complete deployment.

This document uses dot-path notation to refer to nested claim values. For example, act.sub refers to the sub member of the act object, and act.act.sub refers to the sub member of the act object nested within the outer act object (the immediately prior actor in a depth-2 chain).

3. Actor Profile for Delegation

3.1. Overview

This profile specifies an extended form of the act claim defined in [RFC8693]. When an implementation elects to use this profile in a context where an actor is distinct from the subject, it MUST apply the profile as defined in this section. This document standardizes actor-profile claim structure, processing rules, and discovery metadata. It does not standardize delegation approval policy, trust framework decisions, or identifier-mapping logic; those remain deployment-specific. The absence of an explicit actor-carrying inbound credential MUST NOT be interpreted as meaning that the OAuth client automatically defines the delegated actor.

3.2. Profile Invariants

This document defines the following invariants:

* sub is the authorizing principal.

* The outermost act.sub is the immediate actor for the current token presentation.

* The canonical actor identifier conveyed by this profile is the (act.iss, act.sub) pair.

* act.iss identifies the issuer or namespace context in which act.sub is to be interpreted; implementations MUST NOT reinterpret it as the issuer of the current token, a credential-issuer claim, or a hop-provenance marker.

* Nested act objects are preserved prior-actor context unless a deployment explicitly applies additional local-policy processing to them; they are not independently authenticated by the current issuer merely because they appear in a re-issued token.

* client_id and azp are OAuth client identifiers, not actor identifiers.

* When the (act.iss, act.sub) pair identifies the same entity as the (iss, sub) pair of the token, no delegation is expressed and consumers MUST NOT infer a meaningful delegation relationship. String equality of act.sub and sub alone is not a sufficient test; the issuer or namespace context must also be considered.

3.3. Profile Scope

3.3.1. Representation and Policy

The interoperability defined by this profile operates at the representation and propagation layer, not at the authorization policy layer. Conformance means that issuers and consumers represent, preserve, validate, and interpret actor claims consistently: using the same claim structure, the same processing rules across token types, and the same discovery metadata. What this profile does not standardize is whether a specific actor is permitted to act on behalf of a specific subject for a specific scope; that authorization policy decision requires bilateral agreement, a trust framework, or deployment-specific configuration outside this document. This document does not close that policy gap; it makes the actor explicit and consistently represented in the token so that such agreements can be applied deterministically.

This document does not require every deployment to enforce authorization of the (sub, outermost act.sub) pair on every request, although such enforcement is RECOMMENDED for security-sensitive delegated access. Same-domain deployments will often satisfy the representation and propagation requirements with straightforward local configuration for actor identifier context and identifier mapping. Cross-domain deployments require explicit trust-framework or bilateral agreement decisions covering the permitted delegation relationships and therefore carry a higher interoperability burden beyond what this profile alone provides.

3.3.2. Token Format Scope

This profile defines actor-profile requirements for JWT assertion grants, JWT-formatted access tokens, and Transaction Tokens. Opaque access tokens are not conformant token formats under this profile. However, when an AS supports introspection, this document defines optional equivalent introspection response semantics for delegated opaque access tokens in Token Introspection (Section 8.4). Such support is an introspection-based compatibility path; it does not make the opaque token itself conformant to this profile. Use of opaque access tokens as subject_token or actor_token inputs to Token Exchange remains outside the interoperable scope of this document. An AS MAY translate introspection results for an opaque access token into equivalent local inputs for deployment-specific use, but that input-processing behavior is not interoperable behavior defined by this document.

3.3.3. Supported Token Types and Request Semantics

The actor profile defines processing rules for the following token types as issued outputs: JWT assertion grants (JWT Assertion Grants (Section 4)), JWT-formatted access tokens (JWT Access Tokens (Section 5)), and Transaction Tokens (Transaction Tokens (Section 7.1)). It also defines Token Exchange input processing for JWT assertion grants, JWT access tokens, OpenID Connect ID tokens, refresh tokens, and Transaction Tokens as subject_token, and for workload identity credentials, JWT client assertions, and JWT access tokens as actor_token. This document defines limited processing for the may_act claim as an optional delegation-authorization input, as described in may_act (Section 6.4); may_act is not itself a source of actor identity and is never propagated.

Subject to endpoint policy and the underlying token-exchange or grant mechanism, implementations MAY transform a supported input token type into a supported output token type for which this document defines the relevant issuance processing. This document normatively defines JWT assertion grant issuance (JWT Assertion Grant Output (Section 6.5.1)), JWT access token issuance via JWT Access Token Output (Section 6.5.2) after authorization-grant processing, Token Exchange processing, or Transaction Token Service processing, and Transaction Token issuance from inbound JWT assertion grants, JWT access tokens, or Transaction Tokens under Transaction Token Output Rules (Section 7.4). It does not require every implementation to support every possible cross-product of supported inputs and outputs. When an implementation supports a path defined by this document, actor profile information MUST be preserved and validated as specified for the resulting token type.

This document profiles token contents and the processing of those contents once present. It does not redefine the request semantics of [RFC8693], including the syntax or baseline meaning of subject_token, actor_token, resource, audience, or requested_token_type. When such inputs carry or imply actor information, this document defines only how that information is represented in issued tokens and how issuers and consumers process it.

For worked examples showing the actor profile in use in both same-domain service delegation and cross-domain delegation, see the service-to-service example (Appendix A) and the cross-domain example (Appendix B).

3.4. Actor Object Structure

An actor object conforming to this profile is a JSON object that is the value of the act claim. In addition to the sub claim required by [RFC8693], a profile-conformant actor object MUST contain an iss claim and SHOULD contain a sub_profile claim. An act object that omits iss conforms to [RFC8693] but does not conform to this profile; handling of such objects is specified in Migration and Adoption (Section 12.1).

```
act-object = {
  "sub"           : StringOrURI,        ; REQUIRED
  "iss"           : StringOrURI,        ; REQUIRED
  ? "sub_profile" : JSON String,        ; RECOMMENDED
  * StringOrURI => any                  ; extension claims
}
```

sub: REQUIRED. The subject identifier of the actor, as defined in [RFC8693], Section 4.1. This value identifies the acting party. It is a StringOrURI as defined in [RFC7519].

iss: REQUIRED. Identifies the issuer or namespace context in which the actor identifier carried in act.sub is to be interpreted, playing the same role for act.sub that the JWT iss claim plays for the token sub: just as iss + sub form a globally unique principal identifier in a JWT (see [RFC9493]), act.iss + act.sub form a canonical actor identifier within the delegation chain. For URI, client, workload, or other deployment-specific identifiers, the value of act.iss MUST identify the context the issuer used when assigning or asserting that actor identifier. See "Cross-Domain Delegation" in Conventions and Definitions (Section 2). The value is a StringOrURI as defined in [RFC7519].

The canonical actor identifier conveyed by this profile is the (act.iss, act.sub) pair. Implementations MUST NOT interpret act.iss as the issuer of the current token, as a credential-issuer claim, or as a hop-provenance marker. The actor identifier context, the token issuer, and the credential issuer may be the same entity or different entities. In many deployments the value is an HTTPS URL, but other well-known identifier schemes (for example, a URN for workload identities) are also possible. Preserved inner act objects remain prior-actor context as described in Carry Prior-Actor Context (Section 3.6.2.4).

For example, a TTS might issue a Transaction Token with top-level iss equal to https://tts.travel-provider.example while setting act.iss for the booking tool to https://as.travel-provider.example, if local policy uses that AS’s identifier

namespace for booking tool identifiers. This is valid and expected: the TTS is the token issuer, while https://as.travel-provider.example is the actor identifier context for the booking tool identifier.

sub_profile: RECOMMENDED. A space-delimited list of entity profile values classifying the actor identified by act.sub, as defined in Section 4.2 of [I-D.mora-oauth-entity-profiles]. Values used within act objects MUST be registered with the "Actor Profile" usage location in the OAuth Entity Profiles registry (Section 14.1 of [I-D.mora-oauth-entity-profiles]) or be privately defined collision-resistant values.

If the acting entity fits more than one profile, multiple values MAY be included as a space-delimited string (e.g., "service ai_agent"). Interoperability requirements and implementation guidance for multi-value strings are defined in [I-D.mora-oauth-entity-profiles].

When sub_profile is absent from an act object, implementations MUST NOT assume a specific entity type for the actor; resource servers that enforce entity-type-based access control MUST treat an absent sub_profile as an unclassified actor and SHOULD apply the more restrictive policy applicable to unknown entity types.

The sub_profile claim MAY also appear as a top-level JWT claim outside any act object to classify the entity type of the token's sub; it applies exclusively to sub and does not affect sub_profile values within act objects. Issuers SHOULD include a top-level sub_profile when they can authoritatively classify the subject entity type.

Per-actor key provenance within the delegation chain is outside the scope of this profile. The current presenter's keying material is conveyed only by the token's top-level cnf claim, as described in Sender Constraint and Proof-of-Possession Validation (Section 3.7). Other members carried inside an act object, including any confirmation-style members defined by another profile, do not have standardized proof-of-possession semantics under this document unless another specification explicitly defines them.

The client_profile claim defined in [I-D.mora-oauth-entity-profiles] classifies the OAuth client and MUST NOT appear within an act object. Client classification belongs at the top level of the token. An AS or RS that encounters a client_profile member inside an act node MAY reject the token or ignore the offending member; it MUST NOT treat it as a valid actor classification.

When an act object contains extension members beyond those defined in this document, issuers and consumers MUST ignore unrecognized members unless another specification or local policy defines their meaning. An issuer that re-issues a validated delegation chain MAY preserve unrecognized extension members in inherited act objects under local policy. However, companion profiles that need independently verifiable provenance, per-hop receipts, or other chain-wide state SHOULD use the top-level extension pattern described in Companion Profiles and Extension Points (Section 11) rather than relying on inherited act-object extension members.

3.5. Delegation Chains

Delegation chains MUST be represented by nesting act objects as specified in [RFC8693], Section 4.1. In a nested structure, the outermost act object identifies the immediate actor; inner act objects represent prior actors in the chain, with the innermost representing the first actor authorized by the subject. This structure records delegated-actor history within the trust model of the issuer that conveys it; it does not, by itself, provide independent cryptographic provenance for each prior hop. This profile defines a single linear delegation chain per token; concurrent delegations to multiple independent actors are out of scope and yield separate tokens with their own chains.

This document uses the following terminology consistently:

* A *single-hop actor object* is an act object with no nested act; it represents delegation depth 1.

* An *inbound delegation chain* is the complete act structure received in an inbound token, whether depth 1 or greater.

* A *preserved delegation chain* is an inbound delegation chain that an issuer has validated and copied into a newly issued token without rewriting inherited actor entries.

* A *new outermost actor* is the actor object created by the current issuer to represent the newly identified outermost actor for the token it is issuing.

```
{
  "sub": "https://idp.enterprise.example/users/alice",
  "sub_profile": "user",
  "act": {
    "sub": "https://tools.example.com/booking-tool",
    "iss": "https://as.travel-provider.example",
    "sub_profile": "service",
    "act": {
      "sub": "https://agents.example.com/travel-assistant",
      "iss": "https://as.enterprise.example",
      "sub_profile": "ai_agent"
    }
  }
}
```

In this example the booking tool (outermost act) is the current outermost actor. The delegation chain originates with Alice (sub), who first authorized the travel assistant (nested act); the travel assistant then sub-delegated to the booking tool, which is now the immediate presenter. The chain carries prior-actor information to the extent that the current token issuer is trusted to have validated and faithfully propagated it.

Delegation depth is defined as the number of act objects in the chain, counting from the outermost. A token with a single act object and no nested act within it has depth 1; each additional level of nesting adds 1. Depth is counted on the resulting chain after any new outermost act is added, not on the inbound token.

Depth 1 is the minimum interoperable depth. Implementations intended for cross-domain multi-hop interoperability SHOULD support a local maximum of at least depth 4 (sufficient for the reference architecture user → orchestrator → agent → tool, with the tool as current presenter) and SHOULD document the configured maximum. An implementation that supports only depth 1 is conformant to this document but will not interoperate with multi-hop deployments. Same-domain deployments MAY enforce a shallower local maximum when that limit is sufficient for their architecture.

Implementations MUST define and enforce a local maximum delegation depth. Implementations that receive a token exceeding their configured local maximum MUST reject it with invalid_request. When a request would result in a chain exceeding that limit, the AS MUST reject with invalid_request; it MUST NOT silently truncate the chain.

A token represents delegation when the party exercising the token's authorization at runtime (the actor) is distinct from the token subject (sub) and the actor has been authorized by the subject to do so. The conditions that establish this are:

1. A validated actor_token identifying a distinct actor was present in the exchange request that produced this token.

2. An inbound subject_token from a trusted upstream issuer already carried an act chain, establishing that delegation was present before the current exchange.

3. The issuing AS has an independent delegation basis such as a pre-registered grant, explicit consent record, a may_act claim in a validated upstream token (see may_act (Section 6.4)), or a policy rule establishing that the current client or actor is acting as a distinct party on behalf of sub (see JWT Access Tokens (Section 5) for the outside-Token-Exchange case).

When a token represents delegation, the act claim MUST be present and MUST conform to Actor Object Structure (Section 3.4). When none of the above conditions holds, the token does not represent delegation and the act claim MUST be omitted. The AS MUST NOT include act solely because sub and the OAuth client identifier differ; the distinction between sub and client_id is expected and does not by itself constitute delegation under this profile.

3.6. Delegation Chain Validation and Construction

The following normative algorithm governs how an AS validates an inbound delegation chain and constructs the delegation chain in the issued token. It applies to all token types defined in this profile, on request paths where actor-profile conformance is required or claimed, either because one of the type-specific processing sections (Authorization Grant Processing (Section 4.2), Token Exchange Processing (Section 6), or Transaction Token Output Rules (Section 7.4)) mandates it or because local policy or advertised metadata requires profile conformance for the request path. Those type-specific sections reference this algorithm and add type-specific preconditions. Actor objects that do not conform to this profile (for example, RFC 8693 act objects that omit iss) MUST NOT be processed by this algorithm; Migration and Adoption (Section 12.1) specifies how such objects are handled. This algorithm governs claim handling only; it does not by itself create new delegation-authorization requirements beyond those imposed by the invoking processing section and local policy.

3.6.1. Terminology

* *Depth* of a delegation chain: the number of nested act objects counted from the outermost. A single act object with no nested act has depth 1; each additional nesting level adds 1. Depth is measured on the chain as it appears in the issued token, after any new outermost actor is added.

* *Security-relevant entry*: an inner act object that local policy uses as an additional input to issuance decisions (authorization, scope determination, or identity mapping). Use of such entries is deployment-specific and outside the baseline interoperable behavior of this profile.

* *Prior-actor context*: an inner act object preserved for audit or downstream authorization purposes without being used as a direct input to the current issuance decision.

3.6.2. Validation Steps

The AS applies the validation steps in the following order:

1. Validate the carrier token using Validate Carrier Token (Section 3.6.2.1).

2. Validate the outermost actor using Validate Outermost Actor (Section 3.6.2.2).

3. If inner actors are used as inputs to issuance decisions, validate those entries using Validate Inner Actors Used for Decisions (Section 3.6.2.3).

4. For inner actors preserved only as prior-actor context, apply Carry Prior-Actor Context (Section 3.6.2.4).

5. Enforce the configured maximum chain depth using Enforce Depth Limit (Section 3.6.2.5).

3.6.2.1. Validate Carrier Token

The AS MUST validate the token carrying the inbound delegation chain per the type-specific rules applicable to that token before extracting actor claims.

3.6.2.2. Validate Outermost Actor

For the outermost act object the AS MUST:

1. Verify that both act.sub and act.iss are present. If either is absent, reject with invalid_request.

2. Verify that the token issuer is trusted under local policy to assert the (act.iss, act.sub) actor identifier pair. This trust decision is outside the representation-layer interoperability defined by this profile (Representation and Policy (Section 3.3.1)); non-normative validation examples are in Trusting Actor Identifier Pairs (Section 12.2). This step does not interpret act.iss as the token issuer or as proof that prior hops were independently authenticated. If the token issuer is not trusted to assert the actor identifier pair, reject with invalid_grant. The JWT assertion-grant path adds token-type-specific requirements in Authorization Grant Processing (Section 4.2).

3. When the applicable processing path uses Extend Chain with New Actor (Section 3.6.3.1), the AS MUST evaluate whether that actor (act.sub) is authorized to exercise delegation on behalf of sub under local policy (for example, a pre-registered grant, explicit consent record, or policy rule covering the acting party). When the processing path uses Preserve Inbound Chain (Section 3.6.3.2) for an existing chain from a validated, trusted upstream issuer, delegation was evaluated upstream; the AS SHOULD evaluate the preserved relationship under local policy but is not required to do so for baseline interoperability. If the required delegation relationship is prohibited by policy or cannot be confirmed, reject with actor_unauthorized. The JWT assertion-grant path adds token-type-specific requirements in Authorization Grant Processing (Section 4.2).

3.6.2.3. Validate Inner Actors Used for Decisions

Interoperable processing under this profile is defined around sub and the outermost act.sub. If local policy additionally uses an inner act object as an input to issuance decisions, the AS SHOULD apply the same three sub-steps as Validate Outermost Actor (Section 3.6.2.2) for that entry, including delegation-confirmation only when the current processing path or local policy requires it for that entry. Such use of inner actors is deployment-specific.

3.6.2.4. Carry Prior-Actor Context

For inner act objects preserved solely as prior-actor context without being used for any issuance decision, the AS MAY rely on trust in the outer token issuer established by Validate Carrier Token (Section 3.6.2.1) rather than independently validating each hop. The AS MUST NOT treat preserved prior-actor context as independently authenticated; an inner act entry carried in a token is endorsed only by the outer token issuer's signature, not by independent verification at each prior hop.

3.6.2.5. Enforce Depth Limit

Compute the depth of the resulting chain, including any new outermost actor added by Extend Chain with New Actor (Section 3.6.3.1). If that depth exceeds the locally configured maximum (Delegation Chains (Section 3.5)), reject with invalid_request.

3.6.3. Construction Steps

The AS selects exactly one construction step in the following order:

1. If a new actor is identified for the issued token, use Extend Chain with New Actor (Section 3.6.3.1).

2. Otherwise, if a validated inbound delegation chain is present, use Preserve Inbound Chain (Section 3.6.3.2).

3. Otherwise, use Omit act (Section 3.6.3.3).

3.6.3.1. Extend Chain with New Actor

When a new actor is identified, the AS creates a new outermost act object and nests any validated inbound chain beneath it.

```
AddOutermostActor(inbound_chain, new_actor):
  outermost.sub         = new_actor.sub      // REQUIRED
  outermost.iss         = new_actor.iss      // REQUIRED: issuer or namespace context for
 new_actor.sub
  outermost.sub_profile = new_actor.sub_profile  // RECOMMENDED when known

  if inbound_chain is present:
    outermost.act = inbound_chain  // nest entire validated inbound chain
    verify depth(outermost) <= local_max_depth  // see Enforce Depth Limit

  return outermost
```

The act.iss value in the new outermost actor MUST be set by the issuing AS. The AS MUST NOT rewrite act.iss or any other field in inherited inner actor objects; those fields were set by upstream issuers at the time of authorship and are immutable. Dropping prior actors from the inbound chain to produce a shallower chain in the issued token is NOT permitted. If preserving the inbound chain would cause the resulting depth to exceed the local maximum, the AS MUST reject with invalid_request rather than silently truncate. The only exception is when the inbound token carries no act claim, in which case no prior chain exists and the issued token begins a new chain at depth 1. When the new actor identifies the same party as the inbound outermost act.sub (same act.sub and act.iss under the same issuer or namespace context), the AS MAY use Preserve Inbound Chain (Section 3.6.3.2) instead, preserving the existing chain unchanged rather than creating a redundant nested entry.

Detection of identifier reappearance deeper in the inbound chain (for example, the same actor appearing in both inner and outer positions of a longer chain) is not standardized by this profile. An AS MAY apply local policy to such cases; the chain-construction algorithm itself neither requires nor prohibits cycle detection.

3.6.3.2. Preserve Inbound Chain

When no new actor is identified and an inbound chain is present, the AS copies the validated inbound chain unchanged.

```
PreserveChain(inbound_chain):
  return inbound_chain  // copy without modification
```

The AS MUST copy the validated inbound chain exactly into the issued token. The AS MUST NOT add, remove, or rewrite any field in any actor object of a preserved chain.

3.6.3.3. Omit act

When no delegation is present and no actor information should appear in the issued token, the AS MUST NOT include an act claim. The AS MUST NOT silently drop an inbound act claim; if it cannot preserve or extend the chain, it MUST reject the request per Error Responses (Section 9).

3.7. Sender Constraint and Proof-of-Possession Validation

When sender-constrained tokens are used with a delegated token, the top-level cnf claim identifies the key or certificate of the immediate presenter of that token. When delegation is present, that immediate presenter is ordinarily the party identified by the outermost actor. The following normative rules govern proof-of-possession (PoP) validation for all token types and credential exchanges defined in this profile. Individual processing sections reference these rules and MUST apply them in addition to any type-specific requirements stated there.

For Token Exchange, this profile uses exactly two presenter-transition modes:

* *Presenter continuation*: the issued token keeps the presenter of the inbound subject_token. This mode is interoperable only when the inbound subject_token is PoP-capable and carries top-level cnf.

* *Presenter rebind*: the issued token installs a new presenter. This document defines presenter rebind only when a validated actor_token directly identifies the new presenter.

This profile does not define per-actor confirmation members within nested act objects. Stronger prior-hop key provenance, if needed, would require another profile layered on top of this one using the companion-profile extension pattern in Companion Profiles and Extension Points (Section 11).

DPoP nonce handling per [RFC9449], Section 8 applies unchanged to all DPoP-bound token requests under this profile; this profile does not modify DPoP nonce semantics.

3.7.1. Top-Level cnf Governs the Current Presenter

The top-level cnf claim of any token identifies the key or certificate of the current presenter. When delegation is present, that current presenter is the party identified by the outermost act claim: when DPoP ([RFC9449]) is used, the top-level cnf.jkt MUST identify that party’s key; when mTLS ([RFC8705]) is used, the top-level cnf.x5t#S256 MUST identify that party’s certificate. The AS or RS MUST validate proof of possession against the top-level cnf. Confirmation-style members that appear inside an act object due to another specification do not have standardized proof-of-possession semantics under this document.

3.7.2. Token Exchange Continuation

When Token Exchange runs in presenter-continuation mode, the subject_token itself supplies the current presenter's binding state. This mode applies only to PoP-capable subject_token inputs that carry top-level cnf.

* If the subject_token carries top-level cnf, the requester MUST prove possession for that binding using the mechanism applicable to the subject_token type and deployment.

* If the subject_token does not carry top-level cnf, this document does not define presenter continuation for that request.

3.7.3. Token Exchange Rebind

When Token Exchange runs in presenter-rebind mode, the request establishes a new presenter for the issued token. This document defines that presenter as established only by a validated actor_token that directly identifies it using its own top-level subject identity.

* The issuer MUST validate the actor_token and any accompanying proof required by that credential profile or deployment.

* The issuer MUST validate proof for the newly established presenter rather than requiring proof of possession for any prior subject_token binding solely because the inbound subject_token was sender-constrained.

* A delegated JWT access token or any other actor_token whose acting identity is available only through an embedded act claim does not qualify as a direct presenter credential for interoperable presenter rebind under this profile.

Rebind requires an actor_token whose own top-level sub names the new presenter. An actor holding only a delegated credential cannot rebind; intermediate actors that may become new presenters must possess a direct presenter credential (workload credential (Workload Credential Processing (Section 6.3.2.2)), RFC 7523 client assertion (JWT Client Assertion (Section 6.3.1)), or non-delegated JWT access token (JWT Access Token as actor_token (Section 6.3.3))).

3.7.4. Bearer-to-PoP Upgrade

When the inbound subject_token is a bearer credential or an identity-only credential and the request supplies a validated actor_token establishing a new presenter, the issuer MAY issue a sender-constrained output token bound to that new presenter. The absence of inbound top-level cnf creates no continuity obligation in this case.

4. JWT Assertion Grants

This section defines the actor-profile structure and authorization-grant processing rules for JWT assertion grants. JWT access-token structure, Token Exchange processing, and Transaction Token Service processing are defined in later sections.

4.1. Structure

Actor-profile requirements apply to any JWT used as an authorization grant under [RFC7521] and [RFC7523], independent of how or by which specification it was produced. One such profile is the Identity Assertion JWT Authorization Grant (ID-JAG, [I-D.ietf-oauth-identity-assertion-authz-grant]), a JWT bearer grant produced by Token Exchange. When the grant is an ID-JAG, this document acts as an actor-delegation profile layered on the base ID-JAG specification: ID-JAG defines the token-exchange and JWT-bearer flow, while this document defines how actor_token-derived actor identity and any resulting act claim are represented and processed.

Such a JWT MAY include an act claim conforming to the actor profile defined in Actor Profile for Delegation (Section 3). Use of this claim in JWT client authentication assertions is out of scope for this document because such assertions have different issuer and subject semantics. However, implementers should note that some deployments rely on the authenticated OAuth client itself as implicit evidence of the acting party. This document does not prohibit that input, but when delegation is to be expressed explicitly and propagated across token transformations, the acting party is represented by act rather than inferred solely from client authentication.

The following claims are defined for a JWT assertion grant that carries actor-profile delegation. Claims not listed here follow the requirements of [RFC7521] and [RFC7523].

iss (REQUIRED): Identifies the assertion issuer. MUST be authorized by local policy to assert the relationship between sub and act.sub.

sub (REQUIRED): The principal on whose behalf the grant is being made.

sub_profile (RECOMMENDED): Classifies the entity type of sub. MUST conform to the values defined in Actor Profile for Delegation (Section 3).

act (REQUIRED when delegation is asserted): The actor object identifying the entity exercising the subject's delegated rights. MUST conform to the actor object structure defined in Actor Profile for Delegation (Section 3). MUST include act.sub and act.iss.

cnf (REQUIRED when sender-constrained; otherwise OPTIONAL): When the JWT assertion grant is sender-constrained, the assertion MUST carry a top-level cnf claim identifying the binding: cnf.jkt per [RFC9449] when DPoP is used, or cnf.x5t#S256 per [RFC8705] when mTLS is used. When the assertion is not sender-constrained, top-level cnf is OPTIONAL unless required by another profile or local policy.

When the assertion or request context also identifies an OAuth client via client_id, azp, or an authenticated client credential, interoperable processing SHOULD use act.sub rather than treating that client identity as a substitute for it (see Client Identity and Delegation (Section 14.7) and Authorization Grant Processing (Section 4.2)).

Clients SHOULD use JWT assertion grants carrying actor-profile claims only when the AS's support for the actor-determination model has been confirmed via deployment documentation, prior agreement, or discovery. For ID-JAG specifically, that confirmation SHOULD include whether the AS supports the actor-delegation extension model defined by this document.

The following example shows an AS-issued assertion grant, which is the recommended pattern. The Enterprise IdP AS performed Token Exchange, authenticated the agent as the OAuth client, established the delegation relationship under local policy, and signed the assertion. act.iss equals the token iss here because the enterprise AS's issuer identifier is also the actor identifier context for the agent:

```
{
  "iss": "https://as.enterprise.example",
  "sub": "https://idp.enterprise.example/users/alice",
  "aud": "https://as.resource-domain.example/token",
  "jti": "a1b2c3d4-...",
  "exp": 1711820400,
  "iat": 1711816800,
  "sub_profile": "user",
  "cnf": { "jkt": "NzbLsXh8uDCcd7MNwrnNZpX0ak8ACQ" },
  "act": {
    "sub": "https://agents.enterprise.example/travel-assistant",
    "iss": "https://as.enterprise.example",
    "sub_profile": "ai_agent"
  }
}
```

The top-level sub_profile classifies the JWT's sub; sub_profile within act classifies the actor. The top-level cnf.jkt binds the assertion to the agent's DPoP key.

The AS-issued grant is the pattern defined by this document because the issuing AS independently authenticated the agent and validated the delegation relationship before signing. The receiving AS needs to trust the enterprise AS as an issuer and as an authority for asserting the actor identifier pair carried in the grant.

For AS-issued grants, the issuing AS MUST set act.iss to the issuer or namespace context in which act.sub is to be interpreted; for actors registered in the issuing AS's own namespace, this is often the AS's own issuer URI.

This document defines the following issuer patterns:

* an AS-issued delegated assertion where JWT iss is a trusted AS and act.sub identifies the actor (recommended),

* an assertion carrying a pre-existing nested act chain where the current JWT iss is a trusted AS carrying forward prior actor assertions.

A deployment MAY additionally accept a self-issued actor assertion when explicitly enabled by another specification or local policy, but that behavior is outside the scope of this document. Implementations MUST reject self-issued assertion grants by default; see Self-Issued Authorization Grants (Section 14.2) for the security controls any such deployment MUST independently establish.

4.2. Authorization Grant Processing

When an AS receives a JWT assertion grant containing an act claim:

1. The AS MUST validate the assertion per [RFC7523], including signature, iss, sub, aud, exp, and jti.

   * *Non-sender-constrained grants*: When neither a DPoP proof ([RFC9449]) nor an mTLS client certificate ([RFC8705]) is required at the token endpoint, the AS MUST reject any assertion whose jti has already been accepted within the assertion's validity window.

   * *Sender-constrained grants*: The AS SHOULD additionally apply jti replay prevention as defense-in-depth, consistent with [RFC7523].

2. The AS MUST verify that the JWT iss is trusted under local policy to assert delegation on behalf of the actor identified by act.sub.

   Note: Under this document the JWT iss is expected to be a trusted AS. Self-issued grants, where the acting entity is also the token issuer, are a deployment-specific extension outside the scope of this document; see Self-Issued Authorization Grants (Section 14.2).

3. The AS MUST verify that the JWT iss is trusted under local policy to assert the (act.iss, act.sub) actor identifier pair.

   * If act.iss is absent: reject with invalid_request (structural violation).

   * If the JWT iss is not trusted to assert the actor identifier pair: reject with invalid_grant.

   Note: See Validate Outermost Actor (Section 3.6.2.2) for the trust-validation framing and Trusting Actor Identifier Pairs (Section 12.2) for non-normative examples.

4. The AS MUST evaluate whether the identified actor is authorized to exercise delegation on behalf of sub. The required strength of that evaluation depends on how the outermost actor was introduced:

* *New actor*: When the request supplies an actor_token or self-issued assertion that introduces a new act.sub not carried by the inbound chain, the AS MUST confirm the delegation relationship under local policy (for example, a pre-registered grant, explicit consent record, or policy rule).

* *Preserved chain*: When the request preserves an existing chain from a validated, trusted upstream issuer, the issuer trust established in step 2 provides the baseline assurance; the AS SHOULD additionally evaluate under local policy but is not required to do so for baseline interoperability.

In either case:

* If the delegation relationship is prohibited by AS policy or cannot be confirmed: reject with actor_unauthorized.

5. If the inbound assertion’s act object contains a nested act claim (indicating that the asserted actor is itself a delegatee), the AS MUST handle the inner chain as follows:

* *Propagation decision*: The AS MUST determine whether to propagate the inner chain into the issued token. The AS SHOULD propagate it by preserving the nested structure, provided the total resulting chain depth does not exceed the limit in Delegation Chains (Section 3.5). If the AS does not accept pre-chained assertions, it MUST reject the request.

* *Entries used by the AS for issuance decisions*: Interoperable processing is defined around sub and the outermost act.sub. If local policy additionally uses an inner act object for authorization, scope determination, or another issuance decision, the AS MUST validate the act.sub and act.iss pair and MUST evaluate the delegation relationship before using that entry as a security input. Such use of inner act objects is deployment-specific rather than part of the baseline interoperable behavior of this profile.

* *Preserved prior-actor context*: For inner act objects the AS preserves only as prior-actor context, apply Carry Prior-Actor Context (Section 3.6.2.4). Downstream authorization interoperability is defined around sub and the outermost act.sub.

6. The AS MUST verify proof of possession according to the token-endpoint mechanism in use and the top-level cnf semantics in Sender Constraint and Proof-of-Possession Validation (Section 3.7).

* *DPoP*: When the inbound assertion grant is DPoP-bound, it MUST carry a top-level cnf.jkt; reject with invalid_request if absent. The AS MUST:

  - Verify the DPoP proof is valid per [RFC9449] with htm="POST" and htu equal to the AS token endpoint URI.

  - Verify the jkt in the DPoP proof exactly matches the cnf.jkt in the assertion.

  - Use the assertion’s cnf.jkt as set by the upstream issuer; MUST NOT substitute a locally registered key.

  - Reject with invalid_dpop_proof or invalid_grant if the proof is absent or invalid.

    Note: The ath claim is not applicable at the token endpoint and MUST NOT be required. See also [I-D.parecki-oauth-jwt-dpop-grant] for related work on DPoP-bound JWT grants.

* *mTLS*: When the inbound assertion grant is mTLS-bound, it MUST carry a top-level cnf.x5t#S256; reject with invalid_request if absent. The AS MUST:

  - Validate the client certificate presented at the token endpoint against cnf.x5t#S256.

  - Use the cnf.x5t#S256 value set by the upstream issuer; MUST NOT substitute a locally registered certificate.

  - Reject per [RFC8705] if the presented certificate does not match.

* When this JWT assertion grant is later used as a subject_token in Token Exchange, presenter continuation and presenter rebind are determined by Presenter Transition Model (Section 6.1) and Sender Constraint and Proof-of-Possession Validation (Section 3.7), not by nested act contents.

7. If the assertion or authenticated request context identifies an OAuth client separately from act.sub:

   * The AS MAY use that client identity as an additional authorization input.

* The AS SHOULD NOT infer that the client is authorized to act on behalf of the subject solely because the client initiated the request. Such inference is outside the interoperable behavior defined by this profile.

* When local policy maps the client identity to an actor identifier expected to match act.sub, the AS SHOULD perform identifier reconciliation before issuing a token. If reconciliation cannot be established, the AS MUST either treat the identifiers as distinct or reject the request according to local policy.

8. If the AS accepts the assertion, it MUST propagate the actor information into the issued token according to the rules for the output token type being issued. For JWT access tokens, see JWT Access Token Output (Section 6.5.2). For Transaction Tokens, see Transaction Token Output Rules (Section 7.4). When the output is another JWT assertion grant profile, the resulting assertion MUST preserve the validated actor information subject to local policy and the chain-depth limit in Delegation Chains (Section 3.5).

9. When constructing a new outermost act object using Extend Chain with New Actor (Section 3.6.3.1), the AS MAY enrich that object with sub_profile based on its own knowledge of the actor's entity type. The AS MAY also set or enrich the top-level sub_profile of the issued token based on its knowledge of sub. The AS MUST NOT add, modify, or remove any claim in preserved inner act objects; those entries are immutable under Preserve Inbound Chain (Section 3.6.3.2).

5. JWT Access Tokens

This section defines the actor-profile structure of delegated JWT access tokens used by this document. Processing rules that lead to issuance of such tokens are defined in Token Exchange Processing (Section 6) and Transaction Token Service Processing (Section 7).

5.1. Structure

A delegated JWT access token is a JWT access token per [RFC9068] that carries an act claim conforming to the actor profile defined in Actor Profile for Delegation (Section 3). Claims not listed here follow [RFC9068] and any other applicable token profile.

The following claims are defined for a JWT access token that carries actor-profile delegation:

iss (REQUIRED): Identifies the access token issuer.

sub (REQUIRED): Identifies the principal on whose behalf the access token is issued.

sub_profile (RECOMMENDED): Classifies the entity type of sub. MUST conform to the values defined in Actor Profile for Delegation (Section 3).

act (REQUIRED when the token represents delegation per Delegation Chains (Section 3.5)): The actor object identifying the entity exercising the subject's delegated rights. MUST conform to the actor object structure defined in Actor Profile for Delegation (Section 3). MUST include act.sub and act.iss. When the token does not represent delegation, act MUST be omitted.

cnf (REQUIRED when sender-constrained; otherwise OPTIONAL): Binds the access token to the current presenter when a sender-constraining mechanism such as DPoP or mTLS is used.

client_id and azp (OPTIONAL): Identify the OAuth client when carried in the token. They do not identify the delegated actor and MUST NOT be used as a substitute for act.

Some deployments also carry an azp claim as an auxiliary client-identity signal, often as an OpenID Connect carry-over used by vendors in practice. When an issuer uses both azp and act.sub to represent the same acting party, it SHOULD perform identifier reconciliation between them or else treat them as distinct identifiers under local policy. See Migrating from Implicit to Explicit Delegation (Section 12.1.2) and Client Identity and Delegation (Section 14.7) for migration, reconciliation, and the normative client-identity rules.

The following example shows a JWT access token with actor profile claims:

```
{
  "iss": "https://as.resource-domain.example",
  "sub": "https://idp.enterprise.example/users/alice",
  "client_id": "travel-assistant-client-id",
  "azp": "https://agents.example.com/travel-assistant",
  "aud": "https://api.resource-domain.example",
  "jti": "xyz987",
  "exp": 1711820400,
  "iat": 1711816800,
  "scope": "travel:book",
  "sub_profile": "user",
  "cnf": {
    "jkt": "NzbLsXh8uDCcd7MNwrnNZpX0ak8ACQ"
  },
  "act": {
    "sub": "https://agents.example.com/travel-assistant",
    "iss": "https://as.enterprise.example",
    "sub_profile": "ai_agent"
  }
}
```

In this single-hop case the top-level bearer key identifies the same party as the outermost actor because the actor is the bearer. The differing client_id, azp, and act.sub values in this example are intentional: they illustrate a deployment where client-facing identifiers and actor identifiers are distinct and must be reconciled, if at all, only through trusted local mapping rules. In multi-hop chains each actor remains a distinct identity in the chain; see the cross-domain example (Appendix B).

5.2. Delegated Token Issuance

When an AS issues a JWT access token outside Token Exchange and the token represents delegation through an independent delegation basis (Delegation Chains (Section 3.5)), it MUST establish that basis for the (sub, actor) relationship before including act. Examples include a pre-registered delegation grant, an explicit consent record, or a policy rule covering the acting party or a class of acting parties.

A client registration MAY satisfy this requirement only when it uniquely identifies a single distinct acting entity and the AS can derive that actor’s identifier from the registration alone (for example, a dedicated registration for a specific agent or service). A client registration that fronts multiple distinct acting entities (for example, an agent orchestration platform or shared-client deployment) does NOT satisfy this requirement, because client_id alone does not identify the runtime actor in those cases.

When the AS determines the actor from authenticated client context, local delegation policy, or other deployment-specific inputs rather than from an explicit actor-carrying artifact, that is an operational allowance rather than an interoperable actor-proof mechanism defined by this document. The interoperability defined here applies to the issued token and its processing, not to the upstream method by which the AS determined the actor.

The authorization code flow is within scope of the preceding rules. When a resource owner interactively authorizes an agent via /authorize, the AS MAY include act in the issued access token if it has an independent delegation basis establishing that the OAuth client is acting as a distinct actor on behalf of the resource owner. The actor identity MUST be derived from that delegation basis. If no such basis exists, the AS MUST NOT include act in the access token.

This document does not define an authorization request parameter, authorization endpoint interaction, or token request parameter for selecting or proving the actor in the authorization code flow. When actor-profile claims are included in access tokens issued from an authorization code grant, the AS determines the actor from local state established during authorization, client registration, consent, enterprise policy, or another deployment-specific mechanism. Such issuance is interoperable only at the issued-token representation and resource-server processing layer.

6. Token Exchange Processing

Each credential type presented as subject_token or actor_token in a Token Exchange request is processed per the applicable rules in this section. These rules explain how subject or actor identity is introduced into the actor-profile model; they do not redefine the core act semantics in Actor Profile for Delegation (Section 3).

This section covers RFC 8693 Token Exchange input processing and the output-token issuance rules for JWT assertion grants and JWT access tokens. JWT assertion-grant structure is defined in JWT Assertion Grant Structure (Section 4.1). JWT access-token structure is defined in JWT Access Token Structure (Section 5.1). Transaction Token issuance is defined separately in Transaction Token Service Processing (Section 7).

Authorization-server error handling for requests processed in this section is defined in Error Responses (Section 9).

This profile defines three JWT-based actor_token credential types. JWT access tokens use actor_token_type=urn:ietf:params:oauth:token-type:access_token (JWT Access Token as actor_token (Section 6.3.3)).

RFC 7523 client assertions (JWT Client Assertion (Section 6.3.1)) and workload identity credentials (Workload Credential Processing (Section 6.3.2.2)) both use actor_token_type=urn:ietf:params:oauth:token-type:jwt and are distinguished as follows.

A JWT is identified as the RFC 7523 client-assertion profile when it is presented as client_assertion with client_assertion_type=urn:ietf:params:oauth:client-assertion-type:jwt-bearer AND its sub equals the client_id of the authenticating client. A JWT presented as client_assertion whose sub does not equal client_id is not a conformant RFC 7523 assertion and MUST NOT be identified as the RFC 7523 client-assertion profile solely on the basis of the client_assertion parameter; the AS MUST instead apply workload identity credential processing if the credential otherwise matches that profile, or reject with invalid_grant if no profile matches. If the AS cannot identify exactly one supported actor-credential profile for the actor_token, it MUST reject the request with invalid_grant. When client_assertion and actor_token are different JWTs, the AS MUST process them independently per the rules of their respective types; the disambiguation rule above applies only to the actor_token.

6.1. Presenter Transition Model

For PoP migration, this profile distinguishes two semantic classes of subject_token input based on whether they carry inbound act state and presenter-continuity information:

Input type	Carries inbound act state	Carries top-level cnf	Presenter continuity
ID token	No	No	Not available
Refresh token	No	No	Not available
JWT assertion grant	Yes (if present)	Yes (if present)	Available
JWT access token	Yes (if present)	Yes (if present)	Available
Transaction Token	Yes (if present)	Yes (if present)	Available

Table 1

Identity-only inputs (ID tokens, refresh tokens) establish sub and MAY establish supporting subject state such as sub_profile or an authorization ceiling. They do not establish inbound act state or presenter continuity, and do not by themselves justify carrying act into the issued token.

Token-state inputs (JWT assertion grants, JWT access tokens, Transaction Tokens) establish sub and MAY establish sub_profile, inbound act chain state, and current-presenter binding through top-level cnf. They are the only subject_token inputs from which this document defines interoperable delegation-chain preservation and presenter continuation.

Token Exchange under this profile runs in exactly one of two presenter-transition modes:

* *Presenter continuation*: no validated actor_token establishing a new presenter is supplied. The issued token keeps the presenter of a PoP-capable token-state subject_token.

* *Presenter rebind*: a validated actor_token establishes a new presenter for the issued token. When the output token is sender-constrained, its top-level cnf is bound to that new presenter.

This document defines presenter rebind only when a validated actor_token establishes the new presenter. Other ways an implementation might authenticate or install a new presenter are deployment-specific and outside the scope of this document.

When the inbound subject_token is bearer or identity-only, presenter continuation is unavailable, but presenter rebind remains available. This bearer-to-PoP upgrade path lets a deployment exchange an existing bearer-style input for a sender-constrained output token without changing the actor semantics defined by this profile.

A rebind-capable actor_token under this profile is a *direct presenter credential*: its own top-level subject identity names the new presenter, and the request proves possession according to that credential profile when sender-constrained output is being established.

For legacy bearer-to-PoP migration, the recommended interoperable pattern is to present the existing bearer-style or identity-only credential as subject_token, present a direct presenter credential as actor_token, and issue a sender-constrained output token in presenter-rebind mode. Deployments that need to preserve an inbound delegation chain while changing presenters SHOULD use the delegated credential as subject_token and a separate direct presenter credential as actor_token.

JWT assertion grants are not suitable for use as actor_token in Token Exchange. Their sub identifies the subject of delegation rather than the acting party. Requests that need to establish an agent, workload, or client as the actor SHOULD use one of the actor credential types defined in this section instead.

6.2. Subject Tokens

This section is organized by the two semantic subject_token classes defined in Presenter Transition Model (Section 6.1). The class-level text defines the common actor-profile consequences. The individual token sections then define only the validation and extraction rules specific to each token type. SAML 1.1 and SAML 2.0 assertions are not supported subject_token inputs under this profile; while [RFC8693] defines token type URNs for SAML assertions, actor-profile extraction from XML-based SAML credentials is outside scope.

6.2.1. Token-State Subject Tokens

JWT assertion grants, JWT access tokens, and Transaction Tokens are token-state subject_token inputs. For these inputs, this document defines the following common model:

* the validated token establishes the inbound sub;

* sub_profile, if present and trusted, becomes inbound supporting subject state;

* act, if present, becomes inbound delegation-chain state for JWT Access Token Output (Section 6.5.2) or Transaction Token Output Rules (Section 7.4);

* top-level cnf, if present, makes the input eligible for presenter continuation under Sender Constraint and Proof-of-Possession Validation (Section 3.7);

* if top-level cnf is absent, the token still MAY be used in presenter-rebind mode for bearer-to-PoP upgrade.

6.2.1.1. JWT Assertion Grant

When a Token Exchange request ([RFC8693]) presents a JWT assertion grant as the subject_token, the AS MUST apply the inbound validation rules of Authorization Grant Processing (Section 4.2) to validate the inbound token. Assertion-grant output construction from that section does not apply; propagation and scope reduction are governed by the rules below and by JWT Access Token Output (Section 6.5.2).

A JWT assertion grant used as subject_token is a token-state input for Presenter Transition Model (Section 6.1). If it carries top-level cnf, presenter continuation requires proof for that binding per Sender Constraint and Proof-of-Possession Validation (Section 3.7). If it does not carry top-level cnf, it still MAY be used as a bearer subject token in presenter-rebind mode for bearer-to-PoP upgrade when a validated actor_token establishes the new presenter.

The AS MUST apply scope reduction per [RFC8693], Section 4 and MUST then apply the propagation rules in JWT Access Token Output (Section 6.5.2).

6.2.1.2. JWT Access Token

When a Token Exchange request ([RFC8693]) presents a JWT access token as the subject_token (subject_token_type=urn:ietf:params:oauth:token-type:access_token), the AS MUST apply the following steps. Use of an opaque access token as the subject_token is outside the interoperable scope of this profile (see Profile Scope (Section 3.3)).

1. The AS MUST validate the inbound JWT access token per [RFC9068]: signature, iss, sub, exp, nbf, and jti. Because a JWT access token used as subject_token was issued for a resource server, its

aud will not ordinarily include the Token Exchange AS's token endpoint; the AS MUST NOT reject the inbound token solely because its aud does not include the AS's token endpoint URI.

2. The AS MUST verify that the inbound token's iss is trusted under local policy to assert the delegation chain it carries. If not, the AS MUST reject the request with invalid_grant.

3. If the inbound token carries top-level cnf, it is a PoP-capable token-state input for Presenter Transition Model (Section 6.1), and the AS MUST apply the continuation or rebind rules in Sender Constraint and Proof-of-Possession Validation (Section 3.7). If the inbound token carries no top-level cnf, it is a bearer subject_token for PoP purposes and MAY still be used in presenter-rebind mode for bearer-to-PoP upgrade.

4. The AS MUST extract sub, sub_profile (if present), and act (if present) from the validated token as the inbound delegation state for JWT Access Token Output (Section 6.5.2).

5. The AS MUST apply scope reduction per [RFC8693], Section 4. The effective scope of the issued token MUST NOT exceed the inbound token's effective scope.

After completing these steps, the AS MUST apply the propagation rules in JWT Access Token Output (Section 6.5.2).

6.2.1.3. Transaction Token

When a Token Exchange request ([RFC8693]) presents a Transaction Token as the subject_token (subject_token_type=urn:ietf:params:oauth:token-type:txn_token) to a regular AS (not a TTS), the AS MUST apply the following steps.

1. The AS MUST validate the Transaction Token per [I-D.ietf-oauth-transaction-tokens] (signature, aud, exp, iat, and issuer identity). When the token carries an act claim, the top-level iss claim MUST be present and the AS MUST validate it as the Transaction Token issuer identifier; if iss is absent, the AS MUST reject the request with invalid_request. When the token carries no act claim and iss is absent, the AS MUST determine the issuer using the trust-domain and issuer-identification rules of [I-D.ietf-oauth-transaction-tokens] and local deployment configuration. If validation fails or the issuer cannot be established, the AS MUST reject the request with invalid_grant.

2. The AS MUST verify that the Transaction Token issuer identified in step 1 is trusted under local policy. If not, the AS MUST reject the request with invalid_grant.

3. If the inbound Transaction Token carries a top-level presenter-binding claim such as cnf, it is a PoP-capable token-state input for Presenter Transition Model (Section 6.1), and the AS MUST apply the continuation or rebind rules in Sender Constraint and Proof-of-Possession Validation (Section 3.7) using the proof mechanism defined by [I-D.ietf-oauth-transaction-tokens] and the applicable deployment profile. If the inbound Transaction Token carries no top-level presenter binding, it MAY still be used in presenter-rebind mode for bearer-to-PoP upgrade.

4. The AS MUST extract sub, sub_profile (if present), and act (if present) from the validated Transaction Token as the inbound delegation state for JWT Access Token Output (Section 6.5.2).

5. The req_wl claim identifies the workload that requested the Transaction Token from the TTS and provides informational context about the transaction origin. The AS MAY use req_wl for audit or local policy decisions but MUST NOT carry it forward into the issued JWT access token.

6. The AS MUST apply the propagation rules in JWT Access Token Output (Section 6.5.2). For a Transaction Token used as subject_token, this document defines only the actor-profile consequences of the inbound sub, act, req_wl, and presenter-binding state. Transaction Token field semantics and any transaction-specific scope handling remain defined by [I-D.ietf-oauth-transaction-tokens], [RFC8693], and local policy.

6.2.2. Identity-Only Subject Tokens

ID tokens and refresh tokens are identity-only subject_token inputs. For these inputs, this document defines the following common model:

* the validated input establishes sub;

* sub_profile, if available from the validated input or trusted state, becomes supporting subject state;

* the input does not establish inbound act chain state for propagation;

* the input does not establish presenter continuity;

* any act in the issued token therefore comes from a validated actor_token or another independent delegation basis under local policy;

* any sender-constrained issued token is therefore issued in presenter-rebind mode, with the new presenter established in the current exchange.

6.2.2.1. OpenID Connect ID Token

6.2.2.1.1. Overview

An OpenID Connect ID token [OpenID.Core] is a JWT issued by an OpenID Provider (OP) that asserts the authenticated identity of a user. Its sub identifies the user, its aud identifies the relying party (the OAuth client) to which it was issued, and it may carry azp as an authorized-party/client identifier per [OpenID.Core]. Under this profile, aud and azp remain client-identity inputs; they do not establish actor identity. ID tokens carry no act claim and no delegation chain; they establish the user’s identity only.

In this profile, ID tokens serve as subject_token in Token Exchange requests to introduce a user identity into a delegation chain. The acting party is established by the actor_token. This is the foundational pattern for cross-domain identity chaining: the user’s OIDC identity flows forward as sub in the issued token while the actor identity becomes act.sub.

6.2.2.1.2. Processing

When a Token Exchange request ([RFC8693]) presents an ID token as the subject_token (subject_token_type=urn:ietf:params:oauth:token-type:id_token), the AS MUST apply the following steps.

1. The AS MUST validate the ID token per [OpenID.Core] and local policy before using it as actor-profile input. Any checks on aud or azp remain OpenID Connect and client-identity checks; they do not by themselves establish the delegated actor under this profile.

2. The AS MUST use the validated ID token’s sub as the subject identity for the issued token, subject to the same-subject preservation rule in JWT Access Token Output (Section 6.5.2).

3. The AS SHOULD set sub_profile to user in the issued token if it can authoritatively classify the ID token’s sub as a human user identity and no conflicting subject classification is available under local policy.

4. The ID token is an identity-only subject_token for Presenter Transition Model (Section 6.1). It does not establish actor identity or presenter continuity. If an actor_token is present, the AS MUST process it per its type-specific rules and MUST use the derived actor identity as act.sub. If the issued token is sender-constrained, that actor_token also establishes the new presenter for presenter-rebind mode. If no actor_token or independent delegation basis is present, the AS MUST NOT include act in the issued token.

5. The AS MUST apply the propagation rules in JWT Access Token Output (Section 6.5.2) to determine the remaining claims in the issued token. Because an ID token carries no inbound act chain and no OAuth scope ceiling, delegation-chain construction and scope determination come from the actor_token (if any), [RFC8693], and local policy rather than from the ID token itself.

6.2.2.2. Refresh Token

6.2.2.2.1. Overview

A refresh token is a long-lived credential issued by an AS to a specific OAuth client that authorizes that client to obtain new access tokens without repeating end-user authentication. Refresh tokens are opaque by design; they do not carry claims such as sub or act and are validated through the issuing AS's own token store rather than by signature verification.

Refresh tokens MAY be presented as subject_token in a Token Exchange request to introduce a user's existing authorization into a delegation chain; for example, when a client authorized to use a user's refresh token needs a delegated access token for a specific resource. This use is intended for same-AS deployments or deployments with a trusted back-channel to the issuing AS. A refresh token MUST NOT be treated as a portable cross-domain delegation artifact. Because refresh tokens carry no identity claims directly, the actor MUST be established by an actor_token in the same request or by an independent delegation basis under local policy.

Refresh tokens MUST NOT be used as actor_token. They carry no actor identity, are opaque credentials bound to a user authorization grant rather than to a workload or client identity, and cannot supply the sub value required to construct act.sub.

This document does not standardize client-binding policy, cross-client presentation policy, or cross-AS acceptance policy for refresh tokens used as subject_token. Those behaviors remain deployment-specific. This document defines refresh-token subject_token

processing only for deployments where the AS can validate the refresh token's authorization state directly or through a trusted back-channel. Cross-AS refresh-token presentation without such validation is outside the interoperable scope of this profile.

6.2.2.2.2. Processing

When a Token Exchange request ([RFC8693]) presents a refresh token as the subject_token (subject_token_type=urn:ietf:params:oauth:token-type:refresh_token), the AS MUST apply the following steps.

1. The AS MUST validate the refresh token through its own token store or a trusted back-channel with the issuing AS. Signature-based validation does not apply. If the AS did not issue the refresh token, it MUST NOT accept the token unless it validates the token and obtains the associated subject, client binding, authorized scope, and revocation state through a trusted back-channel with the issuing AS. If the token is expired, revoked, or otherwise invalid, the AS MUST reject the request with invalid_grant.

2. The AS MUST verify that the requesting client or authenticated presenter is authorized to use the refresh token under the refresh token's client-binding, sender-constraint, rotation, and cross-client presentation policy. If the requester is not authorized to use the refresh token, the AS MUST reject the request with invalid_grant.

3. The AS MUST extract the subject identity and authorized scope associated with the refresh token from its token store or other trusted refresh-token state. The sub of the user associated with the refresh token becomes sub in the issued token. The AS SHOULD set sub_profile in the issued token if it can authoritatively classify the subject entity type.

4. Refresh tokens are identity-only subject_token inputs for Presenter Transition Model (Section 6.1); they do not establish presenter continuity or actor identity. A refresh token alone does not establish delegation. The acting party MUST be established by an actor_token or an independent delegation basis under local policy. If neither is present, the AS MUST NOT include act in the issued token. If an actor_token is present, the AS MUST process it per its type-specific rules and MUST use the actor identity derived by those rules as the outermost actor in the issued token. If the issued token is sender-constrained, that actor_token also establishes the new presenter for presenter-rebind mode.

5. The effective scope of the issued token MUST be a subset of the scope authorized by the refresh token. The AS MUST apply scope reduction per [RFC8693], Section 4 against that ceiling.

After completing these checks, the AS MUST apply the propagation rules in JWT Access Token Output (Section 6.5.2) to determine the remaining claims in the issued token.

This document does not standardize whether an AS issues refresh tokens in response to delegated JWT assertion grant requests, how such refresh tokens are revoked, or whether later use of such refresh tokens requires re-presentation of upstream delegation artifacts. Those decisions remain deployment-specific.

6.3. Actor Tokens

An actor_token of any type defined in this profile MUST identify the acting party in its top-level sub. If the actor_token carries an act claim, indicating that the credential itself represents a delegation chain rather than a direct identity assertion, the AS MUST reject the request with invalid_grant. This uniform rule preserves the identity-extraction invariant that the actor_token's top-level sub is the new outermost actor. Type-specific sections below refine validation and presenter-rebind handling.

Interoperable sub-delegation under this profile therefore requires every actor that may become a new presenter to possess a direct presenter credential naming itself in sub. Deployments anticipating multi-hop chains SHOULD provision intermediate actors accordingly.

6.3.1. JWT Client Assertion

6.3.1.1. Overview

A JWT client assertion per [RFC7523] may be presented as actor_token (actor_token_type=urn:ietf:params:oauth:token-type:jwt) to establish an OAuth client's own identity as the acting party. Per [RFC7523], the assertion has iss = sub = client_id and is signed with the client's private key. Under Presenter Transition Model (Section 6.1), it is a direct presenter credential. Two usage patterns arise:

* The same JWT is presented as both client_assertion and actor_token in a single request, making the authenticated client identity explicit in the issued token's act chain.

* The client authenticates by another method (e.g., client_secret, mTLS) and presents a separate JWT client assertion as actor_token to name that same client as the actor.

To establish a principal distinct from the OAuth client_id as the actor, the request MUST use a different actor credential type, such as a workload identity credential (Workload Credential Processing (Section 6.3.2.2)), whose sub names that distinct principal. A client assertion conforming to [RFC7523] cannot name a subordinate identity.

6.3.1.2. Processing

When a Token Exchange request includes an actor_token that is a JWT client assertion, the AS MUST apply the following steps.

1. The AS MUST validate the actor_token per [RFC7523]. If the same JWT is also used as client_assertion for client authentication in the same request and this shared validation fails, the AS MUST reject the request with invalid_client; otherwise, the AS MUST reject the request with invalid_grant.

2. The AS MUST verify that the actor_token’s iss is a client registered with the AS, that sub equals that client’s client_id, and that local policy permits that client’s assertion to be used as an actor credential. If not, the AS MUST reject the request with invalid_grant.

3. The actor_token’s sub identifies the immediate acting party. The AS MUST use it as act.sub in the outermost act of the issued token and MUST set act.iss to the issuer or namespace context in which that actor identifier is to be interpreted.

4. When the actor_token is the same JWT presented as client_assertion for client authentication in the same request, the AS MAY derive the actor identity from the already-authenticated client context rather than re-validating the actor_token separately, provided the result is an identical act.sub value. Actor-profile-specific policy failures that occur after successful client authentication are still invalid_grant, not invalid_client.

5. When the request uses this client assertion to establish a sender-constrained output token in presenter-rebind mode, the AS MUST validate any proof required by the selected proof mechanism for the new presenter per Sender Constraint and Proof-of-Possession Validation (Section 3.7).

6. When a subject_token is also present and carries an act chain: the actor_token's sub takes precedence as the new outermost actor identity. Divergence (the actor_token's identified principal and the inbound outermost act.sub refer to different entities under their respective issuer or namespace contexts, with no trusted local mapping establishing equivalence) is the normal case in presenter-rebind flows and is permitted by default. Local policy MAY restrict divergence for specific request paths where the actor_token is expected only to confirm an existing actor rather than introduce a new one; when such a restriction applies and the identities diverge, the AS MUST reject with invalid_grant. Any prior act chain in the actor_token itself MUST NOT be automatically merged with the subject_token's chain; the AS MUST omit it from the issued token unless local policy defines a single unambiguous ordering, in which case the AS MAY preserve it subject to the chain-depth limit in Delegation Chains (Section 3.5).

6.3.2. Workload Identity Credential

6.3.2.1. Overview

A workload identity credential is a JWT that asserts the identity of a software workload (such as a microservice, AI agent, or batch job). Unlike JWT assertion grants (JWT Assertion Grant Structure (Section 4.1)), whose sub identifies the principal on whose behalf the grant is made, a workload credential has the workload's own identifier in sub, making it the natural credential type for establishing an agent or service as the acting party. Under Presenter Transition Model (Section 6.1), it is a direct presenter credential. WIMSE workload credentials are defined in [I-D.ietf-wimse-workload-creds]; profile disambiguation is in Token Exchange Processing (Section 6).

The recommended pattern for agentic Token Exchange is:

* subject_token: a JWT access token or JWT assertion grant carrying the user's sub and the delegation chain (act)

* actor_token: a workload identity credential whose sub is the agent or service identity

* Output: a JWT access token with the user as sub and the workload as the outermost act.sub

6.3.2.2. Processing

When a Token Exchange request ([RFC8693]) includes an actor_token that is a workload identity credential, the AS MUST apply the following steps.

1. The AS MUST validate the workload identity credential per its type specification. For WIMSE workload identity credentials ([I-D.ietf-wimse-workload-creds]), validation follows the rules defined in that specification. If validation fails, the AS MUST reject the request with invalid_grant.

2. The AS MUST verify that the workload credential's issuer (iss) is trusted under local policy to assert the workload's identity. If not, the AS MUST reject the request with invalid_grant.

3. The workload credential's sub identifies the immediate acting party. The AS MUST use it as act.sub in the outermost act of the issued token and MUST set act.iss to the issuer or namespace context in which that actor identifier is to be interpreted.

4. When the request uses this workload credential to establish a sender-constrained output token in presenter-rebind mode, the AS MUST validate the proof required by the workload-credential profile: a WIMSE Workload Proof Token (WPT, [I-D.ietf-wimse-wpt]) per its specification, or a DPoP proof ([RFC9449]) over the token endpoint URI when the credential carries cnf.jkt. If the required proof is absent or invalid, the AS MUST reject the request with invalid_grant.

5. When a subject_token is also present and carries an act chain:

   * The workload credential's sub takes precedence as the new outermost actor identity.

   * Divergence from the inbound outermost act.sub (the two identities refer to different entities under their respective issuer or namespace contexts, with no trusted local mapping establishing equivalence) is permitted by default, as this is the normal case in presenter-rebind flows. Local policy MAY restrict divergence for paths where the actor_token is expected only to confirm an existing actor; when such a restriction applies and the identities diverge, the AS MUST reject with invalid_grant.

   * Any prior act chain in the actor_token itself MUST NOT be automatically merged with the subject_token's chain; the AS MUST omit it from the issued token unless local policy defines

a single unambiguous ordering, in which case the AS MAY preserve it subject to the chain-depth limit in Delegation Chains (Section 3.5).

6.3.3. JWT Access Token

6.3.3.1. Overview

A non-delegated JWT access token may be presented as actor_token to establish a service or workload as the acting party; its top-level sub identifies the acting party and satisfies the direct-presenter-credential requirement in Presenter Transition Model (Section 6.1). A delegated JWT access token (one carrying act) does not satisfy that requirement; see Actor Tokens (Section 6.3) for the sub-delegation pattern.

6.3.3.2. Processing

When a Token Exchange request includes an actor_token that is a JWT access token (actor_token_type=urn:ietf:params:oauth:token-type:access_token), the AS MUST apply the following steps. Use of an opaque access token as the actor_token is outside the interoperable scope of this profile (see Profile Scope (Section 3.3)).

1. The AS MUST validate the actor_token per [RFC9068]. If validation fails, the AS MUST reject the request with invalid_grant.

2. The AS MUST verify that the actor_token’s iss is trusted under local policy to assert the acting party’s identity in the top-level sub. If trust cannot be established, the AS MUST reject the request with invalid_grant.

3. Apply the uniform actor_token rule in Actor Tokens (Section 6.3): if the actor_token carries act, the AS MUST reject the request with invalid_grant.

4. The actor_token’s top-level sub identifies the immediate acting party. The AS MUST use it as act.sub in the outermost act of the issued token and MUST set act.iss to the issuer or namespace context in which that actor identifier is to be interpreted.

5. When the request uses this JWT access token to establish a sender-constrained output token in presenter-rebind mode and the token carries top-level cnf, the AS MUST validate proof for that binding per Sender Constraint and Proof-of-Possession Validation (Section 3.7).

6. When a subject_token is also present and carries an act chain, the outermost actor identity derived from this actor_token takes precedence as the new outermost actor. Divergence from the inbound outermost act.sub (the two identities refer to different entities under their respective issuer or namespace contexts, with no trusted local mapping establishing equivalence) is permitted by default. Local policy MAY restrict divergence for paths where the actor_token is expected only to confirm an existing actor; when such a restriction applies and the identities diverge, the AS MUST reject with invalid_grant.

6.4. may_act

The may_act claim ([RFC8693], Section 4.4) pre-authorizes a specific party to act on behalf of the subject in a subsequent Token Exchange. This document defines limited use of may_act as a delegation-authorization input when actor identity is established by other means: when present in a validated subject_token, it MAY satisfy the delegation-authorization check in step 3 of Validate Outermost Actor (Section 3.6.2.2) without a separately pre-registered grant. In no case is may_act itself the source of actor identity, and it MUST NOT be propagated into any output token.

Two pre-conditions apply regardless of how the Token Exchange request is structured:

1. The subject_token issuer is trusted under local policy to assert may_act on behalf of the subject.

2. The canonical may_act identifier matches the derived actor identity under Identifier Reconciliation (Conventions and Definitions (Section 2)).

Because [RFC8693] Section 4.4 defines may_act with only a sub member and no iss, the issuer or namespace context for may_act.sub is taken to be subject_token.iss; the canonical may_act identifier is therefore (subject_token.iss, may_act.sub). The AS MUST apply configured mapping rules and MUST NOT infer equivalence from naming similarity alone.

How actor identity is derived and how the identifier reconciliation check is performed depends on what the request supplies:

Request includes	Actor identity source	Identifier Reconciliation
actor_token	Derived from actor_token per type-specific rules, yielding (act.iss, act.sub)	Reconcile (subject_token.iss, may_act.sub) against (act.iss, act.sub)
No actor_token	Authenticated confidential client credential	Reconcile (subject_token.iss, may_act.sub) against the canonical client identifier

Table 2

actor_token present: actor identity is derived from the actor_token per the applicable type-specific rules in this document, yielding the (act.iss, act.sub) pair. The AS applies Identifier Reconciliation to determine whether (subject_token.iss, may_act.sub) refers to the same entity as (act.iss, act.sub). may_act MUST NOT override the actor identity derived from the actor_token.

No actor_token present: the requesting client MUST be a confidential client that has authenticated in the request; public clients MUST NOT use this path. Actor identity is derived from the authenticated client credential. The AS applies Identifier Reconciliation to determine whether (subject_token.iss, may_act.sub) refers to the same entity as the authenticated client, and sets act.sub to the canonical identifier of the authenticated client. The AS MUST set act.iss to the issuer or namespace context that locally registered the authenticated client (typically the AS's own issuer URI when the client is registered directly with the AS). This path enables the pre-authorization use case of [RFC8693], where the subject_token pre-authorizes a specific client and that client presents it directly without a separate actor_token.

When may_act is absent or the conditions above are not met, the AS MUST satisfy the delegation-authorization check through another recognized basis (pre-registered grant, consent record, or applicable policy rule). The AS MUST NOT treat the mere presence of may_act as authorization for any actor other than the one whose canonical identity matches it.

6.5. Output Token Rules

6.5.1. JWT Assertion Grant Output

When requested_token_type in a Token Exchange request ([RFC8693]) designates a JWT assertion grant as the output, whether by using urn:ietf:params:oauth:token-type:jwt or a more specific JWT assertion-grant token type defined by another specification (for example, an ID-JAG token type), the issued JWT MUST satisfy the structural requirements of JWT Assertion Grant Structure (Section 4.1). A JWT assertion-grant token type URI is compatible with this section when it is defined as a profile of urn:ietf:params:oauth:token-type:jwt; for example, urn:ietf:params:oauth:token-type:id-jag as defined by [I-D.ietf-oauth-identity-assertion-authz-grant] is one such compatible type. This section defines the actor-profile requirements for the resulting JWT assertion grant regardless of which compatible JWT assertion-grant output token type was requested. The delegation chain MUST be constructed per JWT Access Token Output (Section 6.5.2). The aud MUST be set to the downstream token endpoint (from the resource parameter or deployment configuration), and the assertion MUST be signed by the issuing AS. The AS MUST NOT issue a JWT assertion grant as Token Exchange output unless it is configured to do so and can establish the delegation relationship under local policy.

6.5.2. JWT Access Token Output

The issued JWT access token MUST satisfy the structural requirements in JWT Access Token Structure (Section 5.1). This section is the common output step for Token Exchange paths in this section that produce a JWT access token. It is reached after completing the applicable class-level and type-specific subject_token processing defined above, the applicable actor_token processing, or Transaction Token Service processing.

After completing the applicable subject_token processing in this section, whether for an identity-only input (ID Token Processing (Section 6.2.2.1.2), Refresh Token Processing (Section 6.2.2.2.2)) or a token-state input (JWT Assertion Grant as subject_token (Section 6.2.1.1), JWT Access Token as subject_token (Section 6.2.1.2), Transaction Token as subject_token (Section 6.2.1.3)), or after Transaction Token Service processing (Transaction Token Output Rules (Section 7.4)), the AS MUST apply the following rules when issuing a JWT access token. These rules govern the delegation chain, subject, and scope in the issued token regardless of inbound credential type.

When both a subject_token carrying an inbound act chain and an actor_token are present, the validated actor_token determines the new outermost actor identity in the issued token. Any preserved inbound act chain from the subject_token is nested beneath that new outermost act; the validation and conflict rules are defined in the applicable actor_token section.

When the issued token is sender-constrained, the issuer MUST bind the output token's top-level cnf according to Presenter Transition Model (Section 6.1):

* In presenter-continuation mode, the output token remains bound to the continued presenter of the validated PoP-capable subject_token.

* In presenter-rebind mode, the output token is bound to the new presenter established by the validated actor_token.

* Bearer-to-PoP upgrade is therefore performed by issuing a sender-constrained output token in presenter-rebind mode even when the inbound subject_token carried no top-level cnf.

If a Token Exchange request explicitly seeks a delegated output, for example by supplying an actor_token or by presenting a subject_token that already carries act, and the AS cannot validate the actor information, it MUST reject the request with invalid_grant. If the AS can validate the actor information but cannot establish or confirm the required delegation basis, or if local policy prohibits the relationship, it MUST reject the request with actor_unauthorized. The AS MUST NOT issue a non-delegated JWT access token in place of the requested delegated output.

1. When the issued access token represents delegation per Delegation Chains (Section 3.5), the AS MUST include an act claim. The AS MUST NOT silently drop actor information. If the inbound credential carries no act, no validated actor_token is present, and no independent delegation basis exists, the AS MUST omit act.

2. The AS MUST preserve sub to refer to the same underlying subject as the inbound token. If the AS uses a different subject-identifier namespace, it MAY change the sub value only to re-express that same subject in the new namespace under a trusted local mapping. The AS MUST NOT replace sub with an identifier for a different subject. Subject-namespace translation requirements and relying-party consequences are described in Subject Namespace Translation (Section 14.9).

3. The AS MUST construct the act claim using the construction decision order in Delegation Chain Validation and Construction (Section 3.6). In summary:

   * first, when a new actor is identified, create a new outermost act object and nest any inbound chain beneath it (Extend Chain with New Actor (Section 3.6.3.1));

   * otherwise, when an inbound chain is present, copy that chain unchanged (Preserve Inbound Chain (Section 3.6.3.2));

   * otherwise, omit act from the issued token (Omit act (Section 3.6.3.3)).

   The act.iss obligation applies only to an act object the AS creates for the current actor. Inherited act objects MUST NOT be rewritten.

   When the actor identity was derived from an actor_token parameter rather than from an act claim in the subject_token, the outermost act in the issued token is AS-asserted based on the validated actor_token; it is not chain-propagated from the subject_token. Consumers MUST NOT infer that the actor_token-derived act was present in or endorsed by the subject_token issuer.

4. If actor information that would be preserved or added to the issued token cannot be validated, or nesting would exceed the chain-depth limit in Delegation Chains (Section 3.5), the AS MUST reject the request. The AS MUST return invalid_request when the chain-depth limit would be exceeded and invalid_grant when actor information fails validation; these error codes apply equally to Token Exchange requests producing JWT access tokens. Note: a missing required structural claim (act.sub or act.iss) in the inbound token is a structural violation that produces invalid_request, not a validation failure that produces invalid_grant. The AS MUST NOT issue a token that partially preserves the delegation chain.

5. The AS SHOULD include sub_profile in the issued token’s top-level claims if it can authoritatively classify the token’s sub entity type.

6. The AS MUST apply scope reduction per [RFC8693], Section 4. If the reduction required by [RFC8693] yields an empty scope, the AS MUST reject the request with invalid_scope.

   When the AS additionally applies actor-based scope restriction based on the (sub, act.sub) pair or the actor’s sub_profile:

* The AS MUST include the effective scope value in the token response so that clients can detect any reduction from the requested scope.

* If actor-based restriction yields an empty scope because the actor is categorically unauthorized for any of the remaining scope (for example, the actor's sub_profile is not permitted for any of the requested scope values), the AS MUST reject with actor_unauthorized.

* If actor-based restriction yields an empty scope for a reason unrelated to the actor's categorical authorization (for example, because the actor's scope ceiling simply does not include the requested values), the AS MUST reject with invalid_scope.

* The scope value in the token response MUST reflect the effective granted scope after all reductions, including any actor-based reduction. This is consistent with the existing requirement in [RFC8693], Section 4 that scope in the response indicate the actual authorized scope. Silent divergence between the requested scope and the issued token's scope is not permitted.

7. If the inbound credential carries client_id, azp, or both, the AS MAY preserve those values per the output token profile or local policy. If preserved:

   * They MUST continue to identify the OAuth client and MUST NOT be rewritten to represent delegation state that belongs in act.

   * If preserving them would create ambiguity about the delegated actor relationship, the AS SHOULD omit them.

   See Client Identity and Delegation (Section 14.7) for the normative rules governing client identity and actor identity.

8. Clients SHOULD use the resource parameter ([RFC8707]) when requesting delegated tokens to restrict the issued token's aud claim to the intended resource server. The AS MUST honor resource-indicator constraints in delegated token requests per [RFC8693], Section 4.2. Audience-scoped delegation limits the blast radius of a compromised token by preventing its presentation to resource servers other than the one it was issued for. This profile does not define a new mechanism for audience restriction; it applies the existing resource parameter and aud claim semantics to the delegated-token context.

7. Transaction Token Service Processing

This section defines the actor-profile claim structure for Transaction Tokens and the rules a Transaction Token Service (TTS) applies when it validates supported subject_token inputs, authenticates the new presenter, and issues a delegated Transaction Token.

TTS error handling for requests processed in this section is defined in Error Responses (Section 9).

7.1. Transaction Tokens

Transaction Tokens [I-D.ietf-oauth-transaction-tokens] are short-lived JWTs that capture the workload identity and request context for a series of related service calls within a single business transaction. They are issued by a Transaction Token Service (TTS), which is a specialized authorization server.

Transaction Token claims are defined in [I-D.ietf-oauth-transaction-tokens]. This profile modifies or adds the following claims:

iss (OPTIONAL in [I-D.ietf-oauth-transaction-tokens]; REQUIRED by this profile when carrying act): Issuer of the Transaction Token. MUST be present when the Transaction Token carries an act claim, because recipients need to identify the token issuer before deciding whether to trust the actor identifier pairs carried in the chain. SHOULD be present when the Transaction Token crosses trust-domain boundaries, even in non-delegated cases, as recipients can require it to validate issuer identity or chain-of-trust per local policy. MAY be omitted only when no delegation (act) is present, all tokens are scoped to a single Trust Domain, and all recipients have out-of-band knowledge of the issuer. When iss is omitted, recipients MUST rely on the trust-domain and issuer-identification rules of [I-D.ietf-oauth-transaction-tokens] and local deployment configuration rather than the generic outer-token iss processing rules in this document.

req_wl: This claim provides TTS-level workload context and is not a substitute for act.sub; see Actor Claim in Transaction Tokens (Section 7.1.1).

act (REQUIRED when delegated; OPTIONAL otherwise): Represents the current acting party and any prior delegation steps, conforming to Actor Object Structure (Section 3.4). See Actor Claim in Transaction Tokens (Section 7.1.1) for delegation semantics and the relationship between act.sub and req_wl.

7.1.1. Actor Claim in Transaction Tokens

A Transaction Token is delegated for purposes of this document when it carries an explicit actor credential (either an actor_token in the exchange request or an inbound token that already contains an act chain) establishing that the requesting workload is acting on behalf of the identified sub rather than in its own right. In a delegated Transaction Token, the act claim conforming to this profile MUST be included to represent the current acting party and any prior delegation steps, as specified in Transaction Token Output Rules (Section 7.4). Because a delegated Transaction Token carries act, it MUST also carry a top-level iss claim per Transaction Tokens (Section 7.1). In non-delegated Transaction Tokens (those issued for a workload acting under its own independent grant, where no explicit actor credential was presented), act is OPTIONAL. The TTS MUST NOT infer delegation solely from the fact that sub and req_wl identify different entities; the presence of an explicit actor credential is required.

req_wl identifies the workload that requested the token from the TTS. act.sub identifies the immediate acting party in the subject identifier namespace used by this profile. The authoritative actor identifier for authorization decisions under this document is the outermost act.sub; req_wl is supporting workload context.

Claim semantics under this profile:

* sub: identifies the original initiator. When a Transaction Token is exchanged for a replacement, the new token MUST continue to refer to the same underlying subject. The issuer MAY change sub only to re-express that same subject in a different identifier namespace.

* act.sub (outermost): identifies the immediate acting party. When a TTS sets both req_wl and the new outermost act.sub in a single token issuance (presenter-rebind mode), it MUST ensure they identify the same entity under local policy. When a TTS preserves req_wl from an inbound token, the TTS SHOULD perform identifier reconciliation between req_wl and the outermost act.sub. When a recipient relies on both and cannot reconcile them under local policy, the recipient MUST reject the token.

* Inner act objects: identify prior presenters in the delegation path. act.sub_profile at each level classifies the entity type of that presenter.

The following example shows a Transaction Token after two hops:

```
{
  "iss": "https://tts.enterprise.example",
  "sub": "https://idp.enterprise.example/users/alice",
  "sub_profile": "user",
  "scope": "inventory:check",
  "req_wl": "https://tools.example.com/booking-tool",
  "aud": "https://api.travel-provider.example",
  "txn": "550e8400-e29b-41d4-a716-446655440000",
  "exp": 1711816900,
  "iat": 1711816800,
  "tctx": {
    "action": "check-availability"
  },
  "rctx": {
    "req_ip": "203.0.113.42"
  },
  "cnf": {
    "jkt": "0ZcOCORZNYy9ZhHiZN..."
  },
  "act": {
    "sub": "https://tools.example.com/booking-tool",
    "iss": "https://as.travel-provider.example",
    "sub_profile": "service",
    "act": {
      "sub": "https://agents.example.com/travel-assistant",
      "iss": "https://as.enterprise.example",
      "sub_profile": "ai_agent"
    }
  }
}
```

In this example the booking tool is the current presenter. It is identified by req_wl as the workload that requested the token and by the outermost act.sub in the actor profile’s subject namespace, and it has its own top-level cnf.jkt. The travel assistant appears as a nested act object, showing that it was the prior delegated actor between Alice and the booking tool.

7.2. Presenter Authentication and Transition

The TTS applies the same two presenter-transition modes defined in Presenter Transition Model (Section 6.1), but only for token-state subject_token inputs:

* *Presenter continuation*: the authenticated requester is the same current presenter as the inbound token. This mode is available only when the inbound token carries a top-level presenter binding and the TTS validates proof for that binding under

[I-D.ietf-oauth-transaction-tokens] and the applicable deployment profile. When the inbound token carries act, the authenticated requester MUST correspond to the outermost (act.iss, act.sub) pair. In this mode the TTS preserves the inbound act chain unchanged and MUST NOT add a new outermost act.

* *Presenter rebind*: a validated actor_token direct presenter credential establishes a different current presenter for the issued Transaction Token. In this mode the TTS creates a new outermost act for that presenter and nests any inbound act chain beneath it.

A bearer token-state subject_token that carries no presenter binding is not eligible for presenter continuation, but it MAY still be upgraded to a sender-constrained Transaction Token in presenter-rebind mode when a validated actor_token establishes the new presenter. This document does not define TTS processing for identity-only subject_token inputs such as ID tokens or refresh tokens.

7.3. Supported Subject Tokens

The TTS accepts only token-state subject_token inputs. Identity-only inputs such as ID tokens and refresh tokens do not carry the inbound delegation-chain or presenter state needed for Transaction Token Output Rules (Section 7.4) and are therefore outside the scope of the TTS profile defined here.

7.3.1. JWT Assertion Grant

When the TTS receives a JWT assertion grant as subject_token, it MUST apply the validation, presenter-continuity, and scope-reduction rules in JWT Assertion Grant as subject_token (Section 6.2.1.1). Instead of applying JWT Access Token Output (Section 6.5.2), the TTS uses the resulting subject identity, optional sub_profile, inbound act chain, proof-of-possession state, and effective scope ceiling as the inbound delegation state for Transaction Token Output Rules (Section 7.4).

7.3.2. JWT Access Token

When the TTS receives a JWT access token as subject_token, it MUST apply the validation and extraction rules in JWT Access Token as subject_token (Section 6.2.1.2). Instead of applying JWT Access Token Output (Section 6.5.2), the TTS uses the resulting inbound delegation state as input to Transaction Token Output Rules (Section 7.4).

7.3.3. Transaction Token

When the TTS receives a Transaction Token as subject_token, it MUST apply the validation and extraction rules in Transaction Token as subject_token (Section 6.2.1.3). Instead of applying JWT Access Token Output (Section 6.5.2), the TTS uses the extracted sub, optional sub_profile, inbound act chain, req_wl, and presenter-binding state as input to Transaction Token Output Rules (Section 7.4).

7.4. Transaction Token Output Rules

TTS processing follows the generic delegation chain algorithm defined in Delegation Chain Validation and Construction (Section 3.6) and the output rules in JWT Access Token Output (Section 6.5.2). The steps below apply those rules to the Transaction Token context and define the TTS-specific adaptations: req_wl handling, the TTS presenter-authentication model, Transaction Token claim set requirements, and the treatment of iss in Transaction Tokens. Where a step below is parallel to JWT access token processing, the TTS-specific detail is noted explicitly.

When a TTS receives a token-exchange request to issue or refresh a Transaction Token from an inbound JWT assertion grant, JWT access token, or Transaction Token that carries actor-profile claims, it MUST apply the following rules:

1. The TTS MUST preserve sub from the inbound token to refer to the same underlying subject.

   * The TTS MAY re-express sub in a different identifier namespace only when a trusted local mapping establishes that both identifiers refer to the same underlying subject (for example, when crossing trust-domain boundaries in a federation scenario).

   * The TTS MUST NOT replace sub with an identifier for a different subject.

     Note: Subject-namespace translation requirements and relying-party consequences are described in Subject Namespace Translation (Section 14.9).

2. The req_wl field and any Transaction Token fields other than actor-profile claims remain governed by [I-D.ietf-oauth-transaction-tokens] and local policy. Under this profile, req_wl is supporting workload context and MUST NOT be treated as a substitute for the outermost act.sub.

3. The TTS MUST compute the depth of the resulting act chain after applying step 6. If that resulting chain would exceed the limit in Delegation Chains (Section 3.5), the TTS MUST reject the request with invalid_request.

4. The TTS MUST validate the inbound token and establish issuer trust before preserving or extending any act chain. For the outermost act object in the inbound chain, the TTS MUST:

   * Verify that the inbound token issuer is trusted under local policy to assert the (act.iss, act.sub) actor identifier pair, using local trust mechanisms equivalent to Validate Outermost Actor (Section 3.6.2.2).

   * Evaluate the delegation relationship per Validate Outermost Actor (Section 3.6.2.2), applying the same creation-vs-preservation distinction: MUST evaluate when installing a new outermost actor in presenter-rebind mode; SHOULD evaluate under local policy when preserving an existing chain from a trusted upstream issuer in presenter-continuation mode.

   For inner act objects in the inbound chain:

   * *Security-relevant use*: If local policy uses an inner entry as an input to access control or scope decisions, the TTS SHOULD apply the same validation as for the outermost entry. If the TTS cannot validate it to the required assurance level, it SHOULD reject with invalid_grant.

   * *Prior-actor context only*: If an inner entry is preserved solely for audit purposes without driving any security decision, apply Carry Prior-Actor Context (Section 3.6.2.4).

5. The TTS MUST determine whether the request is presenter continuation or presenter rebind:

   * *Presenter continuation*: The TTS MUST authenticate the requester as the same current presenter as the inbound token. When the inbound token carries act, the authenticated requester MUST correspond to the outermost (act.iss, act.sub) pair or the TTS MUST reject the request with invalid_grant. If the required actor relationship is prohibited by local policy, absent, or cannot be confirmed from the current inputs and policy, the TTS MUST reject the request with actor_unauthorized.

* *Presenter rebind*: The TTS MUST validate a direct presenter actor_token for the new presenter. Before creating a new outermost act object, the TTS MUST evaluate whether the newly authenticated presenter is authorized under local policy to act on behalf of sub for the requested transaction. If the required actor relationship is prohibited by local policy, absent, or cannot be confirmed from the current inputs and policy, the TTS MUST reject the request with actor_unauthorized.

If the current inputs satisfy neither presenter-continuation nor presenter-rebind requirements, the TTS MUST reject the request with invalid_grant.

6. When the issued Transaction Token carries delegated actor information, the TTS MUST include a top-level iss claim identifying itself as the Transaction Token issuer, and it MUST construct the act claim using Delegation Chain Validation and Construction (Section 3.6). In summary:

   * in presenter-continuation mode, preserve the inbound chain unchanged (Preserve Inbound Chain (Section 3.6.3.2));

   * in presenter-rebind mode, create a new outermost act object for the new presenter and nest any inbound chain beneath it (Extend Chain with New Actor (Section 3.6.3.1)).

   For a new outermost actor, the TTS MUST set act.sub to the new presenter’s identifier, MUST set act.iss to the issuer or namespace context for that identifier, and SHOULD set act.sub_profile when known. Inherited act objects MUST NOT be rewritten.

7. When the issued Transaction Token includes a top-level presenter-binding claim such as cnf, that binding applies to the current presenter. The underlying presenter-authentication and proof mechanism is defined by [I-D.ietf-oauth-transaction-tokens] and any applicable deployment profile, not by this document.

8. Transaction Token fields other than actor-profile claims, including scope, tctx, and rctx, are defined by [I-D.ietf-oauth-transaction-tokens] and local policy. This document does not standardize their issuance semantics.

These same preservation rules apply regardless of whether the inbound credential is a JWT assertion grant, a JWT access token, or a Transaction Token, provided that the TTS supports issuing a Transaction Token from that input. This document does not define direct Transaction Token issuance from ID tokens or refresh tokens.

This document does not define TTS-specific processing for may_act. A deployment MAY use may_act under local policy or another specification as a transaction-authorization hint, but that behavior is outside the scope of this document and MUST NOT replace validation of the inbound credential, presenter authentication, or the rules in this document governing whether a new outermost act is created.

8. Resource Server Processing

This section brings together the normative rules for delegated-token consumers at the resource server and the actor authorization model they apply, whether the RS evaluates a JWT directly or relies on introspection.

8.1. Actor Authorization

When a token contains both sub and an act claim, a resource server has two independent principals available for authorization policy:

* *Subject principal* (sub): the party whose authorization is being exercised. This principal typically has a relationship with the resource (e.g., an account, a role, a permission).

* *Actor principal* (act.sub): the party that is making the immediate request. This principal may be in a different organizational domain and trust level from the subject.

For Transaction Tokens, the primary policy pair remains (sub, act.sub). The req_wl claim provides workload context from the TTS and is not a replacement for act.sub. Nested act objects provide prior-actor context for audit or other deployment-specific processing; this document does not standardize their authorization use.

Actor authorization is conditional under this profile. When an RS accepts a token as satisfying a delegated-access requirement, it MUST NOT ignore the act claim and authorize the request solely as if the token were non-delegated. The RS SHOULD evaluate the (sub, outermost act.sub) pair according to local policy. Resource servers that receive delegated tokens SHOULD define and document their actor authorization policy. The following steps describe one approach for resource servers that choose to enforce actor authorization policy:

1. *Advertise delegated-token requirements*: An RS that wants to signal that delegated requests are expected to carry actor-profile information SHOULD set actor_profile_required: true (Protected Resource Metadata (Section 10.2)). An RS MAY still apply actor authorization without advertising it, but clients MUST NOT rely on that behavior.

2. *Evaluate subject authorization*: Determine whether sub has been granted the requested scope or permission, using the same mechanisms applied to non-delegated tokens.

3. *Evaluate actor authorization*: Determine whether the (sub, outermost act.sub) pair is permitted for the requested operation. This evaluation MAY be performed against:

   * a registered delegation policy for the (subject, actor) pair,

   * the actor’s sub_profile (e.g., only AI agents from a trusted domain are permitted to act as delegatees),

   * the token’s scope claim.

   For Transaction Tokens, the RS SHOULD evaluate req_wl as supporting context. If the RS relies on both req_wl and act.sub to identify the current presenter and cannot reconcile them under local policy, it MUST reject the request.

4. *Evaluate combined policy*: Apply resource-specific actor authorization policies (e.g., requiring both principals to have agreed to terms of service).

5. If the RS requires actor authorization but cannot complete it, it MUST reject the request.

When a deployment applies multi-principal authorization under local policy, the outermost act.sub remains the baseline interoperable actor identifier, while nested act objects are additional local-policy inputs only. Failure semantics, ordering, and weighting for those nested actors are deployment-specific. Clients MUST NOT assume that nested actors will be used for authorization unless deployment-specific agreements say otherwise.

8.2. JWT Access Token Processing

Upon receiving a JWT access token on a request path where delegated-token processing may apply, a resource server MUST validate and process that token according to its local delegated-token policy. For RS processing purposes, a JWT access token is evaluated as delegated when it carries an act claim conforming to this profile; the RS MUST NOT infer delegation from azp, client_id, or other client-identity claims alone. The authorization-policy model for delegated tokens is defined in Resource Server Processing (Section 8). Protected resource metadata can advertise delegated-token expectations to clients, but the RS's enforcement decision remains local to the resource server.

When the resource server evaluates a JWT access token as a delegated token under local policy, it MUST:

1. Validate the token's signature, iss, aud, and temporal claims per [RFC9068]. If local policy for the request path requires actor-profile conformance for delegated requests (including when the resource advertises actor_profile_required: true per Protected Resource Metadata (Section 10.2)), any token the RS evaluates as delegated MUST carry act, and each act object the RS relies on MUST include act.iss; if either is missing, the RS MUST reject the request with HTTP 401 invalid_token. The actor_profile_required: true signal does not require non-delegated tokens for the same resource to carry act.

2. If the token carries a top-level cnf.jkt, validate the accompanying DPoP proof per [RFC9449], Section 7. If a DPoP proof is present but the token does not carry cnf.jkt, the RS MUST treat the token as a bearer token; the RS MUST NOT infer a confirmation binding from the DPoP proof key.

3. Extract the sub and the outermost act.sub as the two principals relevant for authorization policy.

4. If the token carries client_id, azp, or both, treat those as client-identity inputs only. The RS SHOULD use act.sub rather than client_id or azp as the actor identifier when act is present. When local policy expects both to identify the same acting party, the RS SHOULD perform identifier reconciliation; if reconciliation cannot be established, the RS MUST either treat them as distinct identifiers or reject the request according to local policy. See Client Identity and Delegation (Section 14.7).

5. Apply actor authorization per Actor Authorization (Section 8.1) when required by local policy or when the token is accepted as satisfying a delegated-access requirement for the request path. Resource servers that do not require actor authorization SHOULD still evaluate the actor as part of authorization, audit, or trust decisions.

6. The RS MAY traverse inner act objects for audit, policy refinement, or trust decisions; such use is deployment-specific. Inner act objects are prior-actor context as described in Carry Prior-Actor Context (Section 3.6.2.4), and interoperable authorization behavior is defined around sub and the outermost act.sub.

7. If any of the above steps fail, return an appropriate error response. The HTTP authentication scheme used in the WWW-Authenticate challenge follows the token's binding mechanism: Bearer per [RFC6750], Section 3.1 for bearer or mTLS-bound ([RFC8705]) tokens, or DPoP per [RFC9449], Section 7.1 for DPoP-bound tokens.

   * If signature, iss, aud, or temporal validation fails: HTTP 401 with error="invalid_token".

   * If DPoP proof validation for cnf.jkt fails: HTTP 401 per [RFC9449], Section 7.

   * If the token is structurally valid but the actor fails an authorization evaluation required by local policy, including actor authorization when required: HTTP 403 with error="actor_unauthorized". The error attribute supports extension values registered in the OAuth Extensions Error Registry; actor_unauthorized is registered by this document (OAuth Error Registry (Section 16.4)). The RS MUST NOT use error="insufficient_scope" to signal actor authorization failures; that error code indicates the token's scope is insufficient for the operation and implies the client should retry with broader scope, which does not describe an actor identity or delegation policy failure.

   * The RS MUST NOT include actor-specific rejection details in error responses exposed to clients outside the trust domain.

8.3. Transaction Token Processing

Upon receiving a Transaction Token on a request path where delegated-token processing may apply, a resource server MUST validate and process that token according to [I-D.ietf-oauth-transaction-tokens], any applicable deployment profile, and the actor-profile rules in this document.

When the resource server evaluates a Transaction Token as a delegated token under local policy, it MUST:

1. Validate the Transaction Token per [I-D.ietf-oauth-transaction-tokens] and local deployment profile (signature, aud, temporal claims, and issuer identification). When the token carries an act claim, the top-level iss claim MUST be present and the RS MUST validate it as the Transaction Token issuer identifier. When the token carries no act claim and iss is absent, the RS MUST determine the issuer using the trust-domain and issuer-identification rules of [I-D.ietf-oauth-transaction-tokens] and local deployment configuration. If local policy for the request path requires actor-profile conformance for delegated requests (including when the resource advertises actor_profile_required: true per Protected Resource Metadata (Section 10.2)), any Transaction Token the RS evaluates as delegated MUST carry act, and each act object the RS relies on MUST include act.iss; if either is missing, the RS MUST reject the request. The actor_profile_required: true signal does not require non-delegated Transaction Tokens for the same resource to carry act.

2. When the token carries a top-level presenter-binding claim such as cnf, validate the accompanying proof according to [I-D.ietf-oauth-transaction-tokens] and the applicable deployment profile. The top-level presenter binding applies to the current presenter only.

3. Extract sub and the outermost act.sub as the two principals relevant for authorization policy. If req_wl is present, treat it as supporting workload context only. The RS MUST NOT treat req_wl as a substitute for act.sub. When local policy expects req_wl and the outermost act.sub to identify the same party, the RS SHOULD perform identifier reconciliation; if reconciliation cannot be established, the RS MUST either treat them as distinct identifiers or reject the request according to local policy.

4. Apply actor authorization per Actor Authorization (Section 8.1) when required by local policy or when the token is accepted as satisfying a delegated-access requirement for the request path.

Resource servers that do not require actor authorization SHOULD still evaluate the actor as part of authorization, audit, or trust decisions.

5. Optionally traverse inner act objects to audit the full delegation chain. If the RS relies on inner act objects for audit, policy refinement, or trust decisions, it MUST do so only under the prior-actor context rules in Carry Prior-Actor Context (Section 3.6.2.4).

6. If any of the above steps fail, the RS MUST reject the request according to the Transaction Token mechanism in use and local deployment profile. Validation or presenter-proof failures are token-validation failures; actor-policy failures required by local policy are authorization failures. The RS MUST NOT include actor-specific rejection details in error responses exposed outside the trust domain.

8.4. Token Introspection

When token introspection ([RFC7662]) is used for delegated tokens, an AS MUST expose actor-profile information needed for equivalent RS processing. For an active delegated token whose authorization context includes actor-profile claims, the introspection response MUST include:

* active: REQUIRED. MUST be true for an active token.

* sub: REQUIRED. The subject of the delegated token, as defined in [RFC7662].

* act: REQUIRED. The actor object conforming to Actor Object Structure (Section 3.4), including act.sub, act.iss, and any nested act chain, structured identically to the JWT form defined in this document.

* sub_profile: REQUIRED when the token’s authorization context includes a top-level sub_profile; otherwise SHOULD be included when the AS can authoritatively classify the subject entity type.

* scope: REQUIRED. The effective scope of the token.

* iss: REQUIRED when the AS has a stable issuer identifier.

* chain_complete: OPTIONAL. When absent, the RS SHOULD treat the chain as complete unless local policy or deployment context indicates otherwise.

An AS MUST NOT omit actor-profile claims that are part of the delegated token’s authorization context, because doing so would misrepresent the token’s delegation status to the introspecting RS. When a delegated token carries a nested act chain, the AS MUST include the complete nested act structure unless local privacy policy requires omitting specific inner chain entries.

When local privacy policy requires omitting specific inner chain entries, the AS MAY return a filtered act chain but MUST include "chain_complete": false. The AS SHOULD NOT filter inner chain entries when it has knowledge (for example, from actor_profile_required metadata (Protected Resource Metadata (Section 10.2)), deployment configuration, or bilateral agreement) that the RS uses one or more inner chain entries as inputs to authorization, scope, or other security decisions. In those cases the AS SHOULD either return the full chain or reject the introspection request.

The RS handles chain_complete: false as follows:

* *Security-relevant use*: When local policy uses any inner act entry as an input to an authorization, scope determination, or other security decision, the RS MUST reject the request upon receiving chain_complete: false. The RS cannot safely make that security decision on an incomplete chain, and a filtered chain is indistinguishable from a chain that was truncated or manipulated before reaching the introspection endpoint.

* *Audit-only use*: When local policy uses inner act entries solely for logging, audit, or informational purposes, and not as inputs to any security decision, the RS MAY accept a filtered chain and record the partial information, provided it notes the incompleteness. The outermost act.sub and sub, which are unaffected by inner-chain filtering, remain available for authorization.

In either case, the RS MUST NOT treat the partial chain as a faithful representation of the complete delegation history. A companion profile that defines additional introspection parameters aligned to the visible delegation chain MUST define how those parameters behave when the visible act chain is filtered, consistent with Companion Profiles and Extension Points (Section 11).

Opaque access tokens are not conformant token formats under this profile. When an AS supports delegated opaque access tokens through introspection, it MUST return the equivalent actor-profile fields listed above for active delegated tokens. This compatibility path does not make the opaque token itself conformant to this profile, and support for it MUST NOT be inferred solely from actor_profile_required metadata.

An introspecting RS MUST apply the same delegated-token processing to the introspection response claims that it would apply to equivalent locally validated JWT claims, including actor authorization when required by local policy. An RS that relies on introspection rather than local JWT validation MUST treat a missing act claim as an inconsistency whenever local policy, protected resource metadata, or other token context indicates that the token is delegated or that actor-profile conformance is required. In such cases, the RS MUST reject the token. Only when no such requirement or indication exists MAY the RS treat an active introspection response without act as representing a non-delegated token.

Introspection endpoints for delegated tokens SHOULD be advertised via the introspection_endpoint parameter in AS metadata ([RFC8414]). When revocation is integrated, the introspection response for a revoked delegated token MUST return "active": false and MUST NOT include act or sub_profile claims.

Resource servers that cache introspection responses for delegated tokens SHOULD use short cache lifetimes appropriate to the deployment’s revocation requirements. When an introspection response carries "chain_complete": false, an RS that uses inner act entries for security decisions SHOULD NOT cache the response, because a subsequent request relying on the cached partial chain cannot distinguish privacy filtering from chain truncation or manipulation.

9. Error Responses

When an AS or TTS rejects a request under this profile for reasons related to actor-profile processing, it MUST return an OAuth error response per [RFC6749], Section 5.2 and [RFC8693], Section 2.2. These error codes do not override invalid_client when a request fails client authentication per [RFC6749] or [RFC7523].

The following table summarizes actor-profile error handling by using three existing OAuth error codes and defining one new error code, actor_unauthorized. The "AS" and "TTS" columns note context-specific triggers where the two server types differ; otherwise the trigger condition applies to both.

Error Code	General Trigger	AS-specific detail	TTS-specific detail
invalid_request	act claim structure syntactically invalid; delegation chain depth exceeds limit; required claim (act.sub or act.iss) absent	DPoP-bound JWT assertion grant omits required top-level cnf.jkt; delegated Transaction Token used as subject_token omits required top-level iss	Delegated inbound Transaction Token omits required top-level iss
invalid_grant	Inbound token or assertion fails signature or claims validation; issuer not trusted to assert the delegation relationship	Proof-of-possession check cannot be confirmed	sub identity cannot be preserved per step 1 of Transaction Token Output Rules (Section 7.4); inbound actor information fails validation per step 4
invalid_scope	Scope reduction under [RFC8693] or local policy leaves no effective scope for reasons unrelated to actor authorization	Requested scope is not available after ordinary Token Exchange scope reduction	Requested transaction scope is not available under TTS policy
actor_unauthorized	Actor-policy failure: the	Includes policy failures for a	Includes the newly

	(sub, act.sub) pair fails a policy check, the actor's sub_profile is not in the accepted set, the required delegation relationship cannot be confirmed from current inputs, or local policy prohibits the actor relationship	newly introduced actor or a preserved delegation chain evaluated by the AS	authenticated presenter the TTS would install as the new outermost actor

Table 3

The error_description field SHOULD be included and SHOULD describe which aspect of actor-profile processing failed, to the extent permitted by the server's security and privacy policy. Some actor_unauthorized failures are recoverable by using a different actor credential, actor type, or delegation grant; others are definitive local-policy prohibitions.

When actor_unauthorized is returned from a token endpoint (AS or TTS), the client SHOULD consult entity_profiles_supported.actor in AS metadata (Authorization Server Metadata (Section 10.1)) to determine whether the actor's sub_profile is accepted, then inspect error_description for missing-grant indications. If neither resolves the failure, deployment documentation governs whether any remediation path exists. When returned from a resource server, the token has already been issued; the client SHOULD obtain a new token via a different actor credential or grant path.

Example:

```
{
  "error": "actor_unauthorized",
  "error_description": "act.sub_profile not accepted for payments:create"
}
```

10. Metadata and Discovery

This section defines a minimal set of metadata parameters that, together with the entity_profiles_supported.actor array defined in [I-D.mora-oauth-entity-profiles] and the authorization_grant_profiles_supported parameter defined by [I-D.ietf-oauth-identity-assertion-authz-grant], allow authorization servers and resource servers to advertise actor-profile support with reduced out-of-band coordination. Authorization grant profile support uses authorization_grant_profiles_supported; Token Exchange role-specific capabilities are grouped under actor_profile_token_exchange; entity type enumeration uses [I-D.mora-oauth-entity-profiles] metadata.

These parameters are coarse-grained capability indicators only; they do not provide a complete description of every supported grant path or guarantee successful token issuance. Deployment documentation, prior agreement, or companion profiles can still be needed for details outside the parameters defined here. Companion profiles MAY define additional metadata, consistent with Companion Profiles and Extension Points (Section 11).

10.1. Authorization Server Metadata

The following parameters are defined for use in the AS metadata document ([RFC8414]):

authorization_grant_profiles_supported: OPTIONAL. A JSON array of authorization grant profile identifiers, as defined by [I-D.ietf-oauth-identity-assertion-authz-grant]. An AS that supports processing JWT authorization grants carrying actor-profile claims under this document SHOULD include urn:ietf:params:oauth:grant-profile:actor-profile in this array. This value covers all JWT authorization-grant paths defined by this document (JWT Assertion Grants (Section 4)), not a single specific grant type variant; it signals that the AS implements the actor-profile JWT authorization-grant processing rules as a whole. It does not indicate that any particular issuer, subject, actor, client, audience, resource, scope, or authorization request will be accepted. An AS that includes urn:ietf:params:oauth:grant-profile:actor-profile in authorization_grant_profiles_supported MUST also include urn:ietf:params:oauth:grant-type:jwt-bearer in grant_types_supported.

actor_profile_token_exchange: OPTIONAL. A JSON object advertising coarse Token Exchange capabilities for requests in which actor-profile processing can apply. When absent, the AS makes no claim about Token Exchange support for this actor profile. This

parameter applies only to Token Exchange; it does not describe authorization code grants or JWT bearer authorization grants. The object members defined by this document are:

* subject_token_types_supported: OPTIONAL. A JSON array of token-type URI strings indicating the subject_token_type values the AS accepts for Token Exchange requests in which actor-profile processing can apply. Values defined by this document are:

  - urn:ietf:params:oauth:token-type:jwt: JWT assertion grants (JWT Assertion Grants (Section 4))

  - urn:ietf:params:oauth:token-type:access_token: JWT access tokens (JWT Access Tokens (Section 5))

  - urn:ietf:params:oauth:token-type:id_token: OpenID Connect ID tokens (OpenID Connect ID Token (Section 6.2.2.1))

  - urn:ietf:params:oauth:token-type:refresh_token: refresh tokens (Refresh Token (Section 6.2.2.2))

  - urn:ietf:params:oauth:token-type:txn_token: Transaction Tokens (Transaction Tokens (Section 7.1))

* actor_token_types_supported: OPTIONAL. A JSON array of token-type URI strings indicating the actor_token_type values the AS accepts for Token Exchange requests in which actor-profile processing can apply. Values defined by this document are:

  - urn:ietf:params:oauth:token-type:jwt: JWT client assertions (JWT Client Assertion (Section 6.3.1)) and workload identity credentials (Workload Credential Processing (Section 6.3.2.2))

  - urn:ietf:params:oauth:token-type:access_token: JWT access tokens (JWT Access Token as actor_token (Section 6.3.3))

* requested_token_types_supported: OPTIONAL. A JSON array of token-type URI strings indicating the requested_token_type values the AS accepts for Token Exchange requests in which actor-profile processing can apply. Values defined by this document are:

  - urn:ietf:params:oauth:token-type:access_token: JWT access tokens (JWT Access Tokens (Section 5))

- urn:ietf:params:oauth:token-type:jwt: JWT assertion grants (JWT Assertion Grants (Section 4))

- urn:ietf:params:oauth:token-type:txn_token: Transaction Tokens (Transaction Tokens (Section 7.1))

Each member of actor_profile_token_exchange is a coarse capability signal only. The presence of a token type in one member does not guarantee that it can be combined with every token type in another member, every resource, every scope, every presenter-binding mechanism, or every profile-specific JWT variant. For example, this document uses urn:ietf:params:oauth:token-type:jwt for both RFC 7523 client assertions and workload identity credentials as actor_token inputs, with profile disambiguation performed during request processing (Token Exchange Processing (Section 6)).

The entity profile types the AS accepts for actors are advertised via the entity_profiles_supported.actor array defined in [I-D.mora-oauth-entity-profiles], not via a separate metadata parameter. DPoP support is advertised via dpop_signing_alg_values_supported per [RFC9449].

Example AS metadata fragment:

```
{
  "issuer": "https://as.enterprise.example",
  "token_endpoint": "https://as.enterprise.example/token",
  "grant_types_supported": [
    "urn:ietf:params:oauth:grant-type:token-exchange",
    "urn:ietf:params:oauth:grant-type:jwt-bearer"
  ],
  "dpop_signing_alg_values_supported": ["ES256", "RS256"],
  "authorization_grant_profiles_supported": [
    "urn:ietf:params:oauth:grant-profile:actor-profile"
  ],
  "actor_profile_token_exchange": {
    "subject_token_types_supported": [
      "urn:ietf:params:oauth:token-type:id_token",
      "urn:ietf:params:oauth:token-type:jwt",
      "urn:ietf:params:oauth:token-type:access_token",
      "urn:ietf:params:oauth:token-type:txn_token"
    ],
    "actor_token_types_supported": [
      "urn:ietf:params:oauth:token-type:jwt",
      "urn:ietf:params:oauth:token-type:access_token"
    ],
    "requested_token_types_supported": [
      "urn:ietf:params:oauth:token-type:access_token",
      "urn:ietf:params:oauth:token-type:jwt",
      "urn:ietf:params:oauth:token-type:txn_token"
    ]
  },
  "entity_profiles_supported": {
    "client": ["service", "ai_agent"],
    "subject": ["user", "service", "ai_agent"],
    "actor":   ["user", "service", "ai_agent"]
  }
}
```

10.2. Protected Resource Metadata

One new parameter is defined for use in Protected Resource Metadata ([RFC9728]):

actor_profile_required: OPTIONAL. A boolean. When true, the RS indicates that requests intending to exercise delegated authority for this resource are expected to provide actor-profile information conforming to this document’s semantics. For JWT access tokens and Transaction Tokens, this ordinarily means a delegated token carrying an act claim conforming to this profile. In deployments that use opaque access tokens, equivalent actor-profile information MAY instead be conveyed to the RS via the

introspection compatibility path in Token Introspection (Section 8.4), but the opaque token itself is not conformant to this profile. When false or absent, actor-profile conformance is not advertised by metadata.

Clients SHOULD treat actor_profile_required: true as a strong indication that delegated access will require either a conforming act claim in the token or an explicitly documented opaque-token/ introspection path providing equivalent claims.

When an AS has obtained and processed Protected Resource Metadata for the target RS and that metadata includes actor_profile_required: true, the AS MUST reject any token request that would produce a token non-conformant with this profile for use at that resource unless an explicitly supported introspection compatibility path will provide equivalent actor-profile information to the RS. An RS that enforces this policy for a given request path MUST reject a request it determines is attempting delegated access if neither a conforming act claim nor equivalent introspection-derived actor-profile information is available to the RS. This parameter does not require otherwise acceptable non-delegated requests for the same resource to carry act, and it does not by itself describe the RS's full actor authorization logic.

This parameter is resource-scoped, not path-scoped; [RFC9728] does not define sub-resource granularity for Protected Resource Metadata. An RS that requires actor-profile conformance only on specific request paths (e.g., /payments but not /profile) MUST apply enforcement at the request layer. Such an RS MAY set this parameter to true as a conservative resource-wide signal, but clients and deployment documentation SHOULD recognize that path-specific enforcement can be stricter than resource metadata alone expresses.

Clients discover which actor entity profile values the RS's AS will accept by consulting entity_profiles_supported.actor in the AS metadata for one of the authorization servers listed in the resource's authorization_servers array ([RFC9728]). When authorization_servers lists multiple entries, the client SHOULD select the AS that issued or will issue the token being presented.

Example Protected Resource Metadata fragment:

```
{
  "resource": "https://api.travel-provider.example",
  "authorization_servers": [
    "https://as.travel-provider.example"
  ],
  "actor_profile_required": true
}
```

10.3. Transaction Token Capability Signaling

Transaction Token support under this profile for Token Exchange paths is advertised through actor_profile_token_exchange.requested_token_types_supported. When an AS or TTS can issue Transaction Tokens as delegated Token Exchange outputs under this profile, it MUST list urn:ietf:params:oauth:token-type:txn_token in actor_profile_token_exchange.requested_token_types_supported. This document does not define any separate Transaction Token discovery parameter.

10.4. Capability Signaling Usage

Clients use Protected Resource Metadata ([RFC9728]) to determine whether a resource advertises actor-profile conformance (actor_profile_required), and the associated AS metadata ([RFC8414]) to assess JWT authorization-grant support (authorization_grant_profiles_supported), Token Exchange compatibility (actor_profile_token_exchange), and accepted actor entity profiles (entity_profiles_supported.actor). When this profile is combined with Identity Chaining ([I-D.ietf-oauth-identity-chaining]), clients SHOULD additionally consult identity_chaining_requested_token_types_supported; the two parameter sets are independent.

The metadata in this document does not advertise authorization-code actor-selection mechanisms or per-scope/per-path actor type restrictions. Deployments that need either capability rely on deployment documentation, bilateral agreement, or a companion profile. When a delegated request carries act.sub_profile, its value SHOULD be drawn from entity_profiles_supported.actor when that metadata is available.

Example client preflight failure: if the RS metadata advertises "actor_profile_required": true but the target AS metadata advertises "entity_profiles_supported": { "actor": ["service"] } and the client's acting entity profile is ai_agent, the client would ordinarily stop before making the token request because the AS does not advertise support for the actor type the client would need to represent.

11. Companion Profiles and Extension Points

This document defines current-token delegated identity while leaving room for companion profiles to define supplementary behavior such as provenance, transparency, or deployment-specific audit material.

A companion profile layered on top of this one:

* MAY define additional top-level JWT claims, OAuth metadata parameters, or introspection response parameters that apply only when a token already conforms to this profile;

* MUST preserve the meanings of the token's top-level sub, the outermost act.sub, the (act.iss, act.sub) actor identifier pair, the nested act chain ordering, and the top-level cnf claim for the current presenter;

* MUST NOT reinterpret act.iss, nested act objects, or the top-level cnf claim as independently trusted prior-hop provenance artifacts;

* SHOULD define any supplementary provenance, receipt, or chain-wide state in separate top-level claims or equivalent companion mechanisms rather than by overloading members inside inherited act objects;

* if it defines data that aligns to the visible act chain, MUST specify the alignment rules, the behavior when coverage is partial, and the behavior when introspection or privacy filtering suppresses part of the visible chain.

An implementation that conforms only to this core profile MUST ignore unrecognized companion-profile claims, metadata parameters, and introspection response parameters unless another specification or local policy defines their meaning. A deployment that requires support for a companion profile MUST express that requirement through the companion profile's own metadata, through out-of-band agreement, or through another explicit local-policy mechanism.

12. Deployment Considerations

This section provides deployment and migration guidance for adopting the OAuth Actor Profile in systems that currently rely on implicit delegation or older act semantics.

12.1. Migration and Adoption

12.1.1. RFC 8693 Backwards Compatibility

This profile defines a conformant act object structure that requires iss in addition to the sub required by [RFC8693], and adds sub_profile for entity-type classification. This is a profile requirement: [RFC8693] act objects that include only sub remain valid under that specification, but they do not conform to this profile. Deployments that receive an act object that conforms only to [RFC8693] but omits this profile's required members MUST treat that actor object as not conforming to this profile.

When a token or assertion is required by local policy or advertised metadata to conform to this profile, such non-conforming act objects MUST be rejected. When profile conformance is not required, implementations MAY continue to process a base [RFC8693] act object according to local policy, but they MUST NOT infer profile-defined semantics for claims that are absent.

The migration path for deployments currently using RFC 8693 act objects is:

1. ASes that issue tokens carrying act claims MUST begin emitting act.iss in all newly issued tokens once support for this profile is enabled. Existing consumers that do not recognize act.iss will ignore it.

2. Update consumers to validate act.iss per Actor Object Structure (Section 3.4) once issuers have deployed step 1.

3. Once all token issuers and consumers on a given path have been updated, resource servers can enforce profile conformance by setting actor_profile_required: true in their Protected Resource Metadata. ASes that wish to accept only profile-conformant inbound assertions can do so via local policy once issuers on inbound paths have deployed step 1.

Steps 2 and 3 are RECOMMENDED; step 1 is REQUIRED for any AS that issues actor-profile tokens. This graduated approach avoids a flag-day cutover and allows incremental rollout across trust domains.

When an AS receives an inbound token or assertion whose act object omits act.iss (that is, the object conforms only to [RFC8693] and not to this profile):

* If local policy or advertised metadata requires actor-profile conformance for the request path, the AS MUST reject the request. It MUST return invalid_request consistent with Error Responses (Section 9).

* If actor-profile conformance is not required, the AS MAY process the inbound act object under local policy for non-profile behavior, subject to the following constraints:

  - The AS MUST NOT rewrite an inherited actor entry to add act.iss; inherited entries are immutable.

  - The AS MUST reject any request that would carry the non-conforming chain into a profile-conforming output token.

  - When the AS issues a non-profile-conforming output token under local policy, it MUST NOT silently drop the inbound act claim.

Implementations that previously treated confirmation members inside act as active sender-constraining mechanisms should note that this document defines proof-of-possession only through the top-level cnf claim and the immediate presenter. Deployments that relied on per-hop actor-key verification for multi-hop security properties will need a separate provenance mechanism or profile rather than the core actor profile defined here.

12.1.2. Migrating from Implicit to Explicit Delegation

Migration from implicit delegation is the process of making actor identity explicit in tokens where the acting party was previously inferred from client_id, azp, or token-request context. The normative client-identity rules are defined in Client Identity and Delegation (Section 14.7).

Deployments that currently rely on implicit delegation can migrate incrementally. A common transition pattern is to emit both legacy client-oriented identifiers and explicit actor claims during rollout, measure and reconcile mismatches, and then require act where explicit delegation is needed.

A deployment that chooses explicit delegation only can reject non-act delegated requests entirely and omit legacy client-identity reconciliation.

One safe migration pattern is:

* Clients that can obtain explicit-delegation tokens under this profile SHOULD prefer those tokens over relying on legacy implicit client-identity interpretation.

* If act is absent, deployments MAY continue to apply legacy implicit client-based policy according to local policy, and existing client-based authorization logic MAY remain in place during migration.

* If both explicit and implicit signals are present, apply the reconciliation rules in Client Identity and Delegation (Section 14.7) and record mismatches during rollout.

* An RS that enforces explicit delegation under this profile for a given request path MUST NOT treat the successful presentation of a non-act token as an acceptable substitute for a delegated token merely because legacy client-based policy would otherwise allow the request.

During transition, issuers SHOULD emit both legacy client-oriented identifiers and explicit actor claims whenever doing so is feasible for the deployment.

The legacy form carries only client_id (and optionally azp) to identify the acting party. The explicit form adds an act block:

```
{
  "iss": "https://as.example.com",
  "sub": "https://idp.example.com/users/alice",
  "client_id": "travel-assistant-client-id",
  "azp": "travel-assistant-client-id",
  "act": {
    "sub": "https://agents.example.com/travel-assistant",
    "iss": "https://as.example.com",
    "sub_profile": "ai_agent"
  },
  "scope": "booking:create"
}
```

In this example, client_id and azp remain as auxiliary client-identity inputs while act.sub carries the explicit actor identity. See Client Identity and Delegation (Section 14.7) for the normative treatment of these claims when both are present.

Mismatch example, where the client and actor identify different parties:

```
{
  "iss": "https://as.example.com",
  "sub": "https://idp.example.com/users/alice",
  "client_id": "travel-assistant-client-id",
  "act": {
    "sub": "https://agents.other-provider.example/concierge-bot",
    "iss": "https://as.other-provider.example",
    "sub_profile": "ai_agent"
  },
  "scope": "booking:create"
}
```

This example illustrates the mismatch case covered by Client Identity and Delegation (Section 14.7): the OAuth client identifier and the explicit actor identifier name different parties unless trusted local mapping rules bind them.

12.2. Trusting Actor Identifier Pairs

This section gives non-normative examples for the trust decision required by Validate Outermost Actor (Section 3.6.2.2). Actor resolution under this profile starts from the (act.iss, act.sub) pair; the token's top-level iss identifies the token issuer and is the actor identifier context only when the two happen to be the same entity. As described in Representation and Policy (Section 3.3.1), this profile does not define a universal trust framework or proof algorithm for establishing that act.iss is authoritative for act.sub.

Examples of local validation approaches include:

* federation metadata or trust-framework configuration that authorizes the token issuer to assert actor identifiers in the act.iss context (for example, [OpenID.Federation]);

* pre-registration entries that explicitly authorize the token issuer to assert a specific (act.iss, act.sub) pair or identifiers of that form; and

* bilateral or deployment-local policy rules that authorize the token issuer to carry the specific class of actor identifier used in act.sub.

For HTTPS URI identifiers, one possible local rule is URL namespace containment. Under that style of rule, deployments might accept a token issuer's assertion of an actor identifier pair when the scheme, host, and port of act.iss and act.sub match, or when act.sub falls within a path prefix of act.iss that has been explicitly configured for that issuer. In those deployments, scheme and host comparison

typically follows the case-insensitive rules in [RFC3986], Section 3.2.2, while path-prefix matching is usually case-sensitive and boundary-aware. Subdomain relationships alone are often insufficient to establish trust without explicit configuration.

Examples:

* A token issued by https://as.enterprise.example with act.iss = https://as.enterprise.example and act.sub = https://as.enterprise.example/agents/travel-assistant would commonly satisfy a same-host local trust rule.

* A token issued by https://as.enterprise.example with act.iss = https://as.enterprise.example and act.sub = https://idp.enterprise.example/users/alice would not ordinarily satisfy URL containment alone, because the host differs.

* For non-HTTPS identifier schemes such as workload-identity URNs, deployments typically rely on registry, federation, or other explicit local trust configuration rather than URL-based rules.

12.3. Token Lifetime for Delegation Chains

Delegated tokens carry the combined exposure surface of all principals in the delegation chain. A single issued token may authorize actions by an actor whose delegation grant has since been revoked; the token remains valid until its exp time. Because revocation cannot retroactively invalidate already-issued tokens, lifetime is the primary control.

Deployments SHOULD use shorter token lifetimes for delegated tokens than for non-delegated tokens of equivalent scope. In cross-domain flows, the upstream delegation artifact (e.g., an ID-JAG or delegated access token) limits how long the downstream actor can continue to request tokens without re-authenticating the delegation chain. Transaction Tokens are already expected to be short-lived; JWT access tokens and JWT assertion grants in multi-hop chains SHOULD be issued with lifetimes no longer than necessary to complete the task they authorize.

Deployments with deep chains (three or more nested act objects) SHOULD account for the fact that any revocation event at any hop in the chain cannot propagate to already-issued downstream tokens. In environments with significant revocation risk (for example, AI agent delegation chains where user consent can be withdrawn at any time), deployments SHOULD combine short-lived tokens with active introspection at sensitive resource servers rather than relying solely on exp.

General guidance on access token lifetime and security tradeoffs is provided in [RFC9700].

13. Conformance

This section enumerates per-role requirements for claiming conformance to this profile. Profile scope (representation versus policy, supported token formats, and supported request semantics) is defined in Profile Scope (Section 3.3). An implementation claiming conformance MUST satisfy the requirements listed for each role it performs.

13.1. Issuing Authorization Server

An issuing AS that claims conformance to this profile MUST:

* emit act.iss in every newly issued token carrying act (Actor Object Structure (Section 3.4));
* apply the chain validation and construction algorithm in Delegation Chain Validation and Construction (Section 3.6);
* define and enforce a local maximum delegation depth (Delegation Chains (Section 3.5));
* preserve sub across token issuance, translating only under a trusted local mapping (JWT Access Token Output (Section 6.5.2));
* apply the presenter-transition model in Presenter Transition Model (Section 6.1) when performing Token Exchange;
* reject any actor_token carrying act (Actor Tokens (Section 6.3));
* return the error codes specified in Error Responses (Section 9) for the listed failure conditions;
* preserve act and top-level sub_profile in introspection responses for active delegated tokens, when introspection is supported (Token Introspection (Section 8.4)).

13.2. Transaction Token Service

A TTS that claims conformance to this profile MUST:

* satisfy the issuing-AS requirements above for Transaction Token output;

* set top-level iss on every Transaction Token carrying act (Transaction Tokens (Section 7.1));

* apply the TTS presenter-authentication and output rules in Transaction Token Output Rules (Section 7.4);

* treat req_wl as supporting workload context, not as a substitute for the outermost act.sub (Actor Claim in Transaction Tokens (Section 7.1.1)).

13.3. Resource Server

An RS that claims conformance to this profile MUST:

* validate act only when the outer token issuer is trusted to convey it (Delegation Chain Integrity and Trust (Section 14.1));

* treat client_id and azp as client-identity inputs only, not as actor identifiers, when act is present (Client Identity and Delegation (Section 14.7));

* evaluate proof of possession against the top-level cnf only (Sender Constraint and Proof-of-Possession Validation (Section 3.7));

* use the WWW-Authenticate challenge scheme appropriate to the token’s binding mechanism (JWT Access Token Processing (Section 8.2)).

An RS that enforces actor authorization additionally MUST evaluate the (sub, outermost act.sub) pair per Actor Authorization (Section 8.1).

13.4. Client

A client that claims conformance to this profile SHOULD treat actor_profile_required: true as an indication that delegated access for the resource requires act-carrying tokens (Protected Resource Metadata (Section 10.2)), and adjust its grant selection or token request accordingly.

14. Security Considerations

As described in Representation and Policy (Section 3.3.1), this document does not define a trust framework for proving that an actor identifier context is authoritative for an actor identifier, proving delegation approval, or validating subject-identifier translation. Security for those decisions depends on deployment-specific policy and external agreements.

14.1. Delegation Chain Integrity and Trust

An attacker who can inject or forge act claims can impersonate an arbitrary actor and exercise a subject's permissions without authorization. The primary mitigation is to accept act claims only in tokens whose issuer is trusted to assert the delegated actor relationship. RS implementations MUST validate the token signature before extracting actor claims, and MUST verify that the token issuer is trusted to convey the claims it carries.

Because inner act objects are set by upstream ASes and not re-signed at each hop, the integrity of the entire delegation chain rests on the outermost token's signature. Implementations SHOULD use short token lifetimes and MUST reject tokens whose exp has passed, regardless of chain depth.

Inner act objects are prior-actor context under Carry Prior-Actor Context (Section 3.6.2.4). Security policies that rely on inner actor identities for access control are deployment-specific and generally lower-assurance than policies based on sub and the outermost act.sub.

When a token crosses organizational boundaries, the receiving AS or RS MUST apply appropriate trust evaluation. ASes performing Token Exchange MUST evaluate cross-domain delegation grants explicitly and SHOULD NOT grant cross-domain actors the same rights as same-domain actors absent an explicit trust decision that makes them equivalent.

14.2. Self-Issued Authorization Grants

This section addresses self-issued JWT _authorization grants_ (JWT Assertion Grants (Section 4)); it does not apply to RFC 7523 client assertions used as actor_token (JWT Client Assertion (Section 6.3.1)), where iss = sub = client_id is the conformant pattern defined by [RFC7523].

In a self-issued assertion grant, the acting entity is itself the JWT iss and directly asserts delegation to itself without any upstream AS having authenticated the actor or pre-validated the delegation

relationship. Self-issued authorization grants are outside the interoperable scope of this document and MUST be rejected by default (JWT Assertion Grant Structure (Section 4.1)). This section specifies the security controls that a deployment MUST independently establish when another specification or local policy explicitly enables self-issued authorization grant acceptance.

Because no upstream AS vouches for the actor's identity or the delegation relationship, the receiving AS MUST NOT treat the self-asserted delegation claim alone as a sufficient authorization basis. When a deployment enables self-issued authorization grants, the receiving AS MUST at minimum:

* Validate the JWT signature using the key identified in the JWT header, obtained from a pre-registered or otherwise independently trusted source for the self-issuing party.

* Verify the exp, iat, and nbf claims per [RFC7519].

* Reject assertions whose jti has already been accepted within the assertion's validity window to prevent replay.

* Verify proof of possession per the token-endpoint mechanism in use (DPoP per [RFC9449] or mTLS per [RFC8705]).

* Apply the actor-profile validation and proof-of-possession requirements in Authorization Grant Processing (Section 4.2).

* Establish the delegation relationship from an independent authorization basis such as a pre-registered grant, explicit consent record, or equivalent deployment-specific artifact.

In the absence of these controls, an attacker can self-assert an arbitrary (sub, act.sub) pair and bypass actor-profile authorization enforcement. Deployments MUST confine self-issued authorization grants to within a single trust domain and MUST NOT propagate them across organizational boundaries. This document defines no discovery or negotiation mechanism for self-issued authorization grant acceptance; any such mechanism is the responsibility of the enabling specification or local policy.

14.3. Assertion Replay Prevention

JWT assertion grants carrying act are higher-value replay targets than plain JWT assertions because a successful replay can establish an unauthorized delegation chain. [RFC7523] recommends jti replay prevention as a SHOULD; this profile strengthens that requirement for non-sender-constrained grants as described in Authorization Grant Processing (Section 4.2). Deployments that do not apply sender constraint MUST maintain a jti replay cache for the duration of each assertion's validity window. Short assertion lifetimes (on the order of minutes) bound the required cache retention period. Deployments using DPoP or mTLS sender constraint benefit from proof-of-possession as primary replay resistance; jti replay prevention remains RECOMMENDED as defense-in-depth for those paths.

14.4. Token Substitution

An attacker who can present a token with a crafted sub_profile or delegation chain could attempt to escalate privileges. ASes MUST validate inbound sub_profile values against the syntax requirements of this document, the applicable registry or deployment-specific allowed set where such checks are part of local policy, and the local policy applicable to the token they are issuing. They MUST preserve unrecognized but syntactically valid values as required by Actor Object Structure (Section 3.4), and they MUST reject values that are malformed or disallowed by local policy.

14.5. Confused Deputy

A resource server that evaluates only the subject principal when an act claim is present is susceptible to a confused deputy attack: a malicious actor exploits a subject's pre-existing permissions without the subject's ongoing consent simply by presenting a token that names the subject in sub. The mitigation is authorization of the (sub, outermost act.sub) pair before granting access. Resource servers SHOULD implement such evaluation for delegated tokens under this document.

14.6. Actor-Authorization Bypass

A resource server that accepts delegated tokens but fails to enforce the (sub, outermost act.sub) relationship required by its local policy allows an attacker to bypass that policy by exploiting gaps in enforcement logic. Resource servers that require actor authorization SHOULD apply that evaluation on every request path where delegated access is accepted, including introspection-based validation paths when used. Deployments that signal delegated-token requirements with actor_profile_required: true SHOULD ensure that the documented

request paths requiring delegated access are aligned with their actual enforcement behavior so that clients do not over-read the signal.

14.7. Client Identity and Delegation

Client identity, such as client_id, azp, or authenticated client context, is widely used in deployed systems as an authorization input. Under this document, those values remain auxiliary client-identity signals, while the outermost act.sub is the explicit delegated-actor signal when present. The following normative rules apply:

* When act is present, interoperable processing SHOULD use it as the explicit delegated-actor signal rather than substituting client_id, azp, or other client-identity signals. Deployments that rely on such substitution are outside the interoperable scope of this profile.

* When a single client_id registration fronts multiple distinct acting entities (for example, an agent orchestration platform executing requests on behalf of different agent instances), client_id alone does not identify the runtime actor. Each such request SHOULD carry act.sub identifying the specific acting principal.

* During token issuance, client_id and azp MUST NOT be rewritten to represent delegation state that belongs in act; see JWT Access Token Output (Section 6.5.2) for propagation rules.

* When both explicit (act.sub) and implicit (client_id, azp) signals are present and local policy expects them to identify the same party, implementations SHOULD apply trusted local mapping rules and either reconcile the identifiers or treat them as distinct according to local policy.

* When a protected resource or authorization path enforces explicit delegation under this profile, implementations MUST NOT downgrade to non-act processing solely because another token-acquisition path or legacy policy input remains available.

The detailed migration rules and transition patterns are defined in Migrating from Implicit to Explicit Delegation (Section 12.1.2).

14.8. sub_profile Trust

The sub_profile claim is asserted by the token issuer and is only as trustworthy as that issuer. Resource servers MUST NOT trust sub_profile values in tokens issued by untrusted parties. Resource server operators SHOULD configure a list of accepted entity-type profiles per trust domain.

14.9. Subject Namespace Translation

Federated deployments sometimes need to re-express sub when a token crosses a trust-domain boundary and the issuer uses a different subject-identifier namespace. This document therefore permits an AS or TTS to translate sub only to re-express the same underlying subject in a different namespace, as described in JWT Access Token Output (Section 6.5.2) and Transaction Token Output Rules (Section 7.4).

This document does not define an interoperable mechanism for proving, conveying, or independently verifying subject equivalence across namespaces. A translated sub is therefore authoritative only within the trust context of the issuer that performed the translation. It does not, by itself, create portable proof that the original and translated identifiers are equivalent, and it does not define a general cross-token correlation mechanism across trust domains.

Subject-namespace translation is a high-assurance operation. An issuer MUST NOT translate sub unless local policy establishes that the original and translated identifiers refer to the same underlying subject. Trust to perform that mapping is separate from trust to sign tokens and SHOULD be established explicitly.

A receiving AS or RS that relies only on the issued token MAY evaluate the translated sub as the subject in the issuer’s namespace. A receiving AS or RS that needs proof that the translated subject is equivalent to an upstream subject MUST obtain additional evidence outside this profile, such as another specification, an identity-chain mechanism, or an explicit trust agreement. If such proof is required and cannot be established, the AS or RS SHOULD reject the token rather than treat the translated sub as advisory.

Deployments that need portable proof of subject equivalence across namespaces, or independently verifiable subject-mapping evidence, require another specification or companion profile; such a mechanism is outside the scope of this document.

14.10. Presenter Binding

Without top-level presenter proof of possession, a leaked token can be replayed by any party.

* The RS SHOULD require the presenter-proof mechanism appropriate to the token type and deployment for the top-level cnf.jkt or other top-level confirmation information. For example, JWT access tokens commonly use DPoP or mTLS, while Transaction Tokens can use the workload proof mechanism defined by their deployment profile.

* Deployments that use sender-constrained tokens for delegated access SHOULD apply that protection to the current presenter to reduce delegation-token theft risk.

This document does not define per-hop actor-key provenance within the delegation chain. Deployments that need stronger assurance for prior-hop provenance MUST use an additional mechanism outside the scope of this document, such as signed hop receipts, transparency-log-based recording, or another future extension; they MUST NOT overload act.iss or redefine nested act semantics to carry that provenance. Companion profiles that supply such mechanisms MUST follow Companion Profiles and Extension Points (Section 11).

14.11. Delegation Depth Limits

Unbounded delegation chains increase attack surface and complicate policy evaluation. Depth support and interoperability requirements are defined in Delegation Chains (Section 3.5). Implementations that encounter chains exceeding their configured local maximum MUST reject the token to prevent denial-of-service through chain parsing.

14.12. Actor Identity Rotation

The canonical actor identifier under this profile is the (act.iss, act.sub) pair. Deployments SHOULD choose act.sub to be a durable, stable identifier independent of ephemeral key material. In particular, deployments SHOULD NOT use a JWK thumbprint or other key-derived value as act.sub; doing so silently changes the actor identity on every key rotation, which can break delegation grants and policy bindings that reference the prior identifier. Key rotation (replacing the DPoP key or mTLS certificate bound to an actor) does not require changing act.sub when the identifier is key-independent.

When act.sub itself must change (for example, because an agent instance is replaced, a workload is renamed, or an actor identifier namespace migrates), existing delegation grants and issued tokens continue to reference the old identifier. Deployments MUST

explicitly re-establish delegation grants for the new identity; the old grants do not automatically transfer. Outstanding tokens issued under the old identity remain valid until their exp time; short token lifetimes bound the exposure window during the transition.

14.13. Delegation Revocation

The revocation-related requirements in this section are limited to how this profile interacts with already-issued tokens and refresh behavior.

Token revocation ([RFC7009]) applies to individual tokens but does not revoke an underlying delegation relationship or invalidate already-issued downstream tokens in a delegation chain. When Alice revokes her delegation to an agent, access tokens already issued to downstream actors remain valid until their exp time. Short token lifetimes are the primary mitigation; see [RFC9700] for general access token lifetime guidance.

The AS SHOULD refuse to issue new tokens for a (subject, actor) pair when it has authoritative knowledge that the delegation relationship has been revoked. Implementations MUST NOT use delegation chain depth as a rationale for skipping revocation checks.

When a deployment uses long-lived delegated refresh behavior, revocation of the delegation relationship may take effect later than it would in a model that requires re-presentation of a current upstream delegation artifact at each token request. This document does not standardize refresh-token revocation or revalidation policy; deployments should account for that tradeoff explicitly.

The internal mechanism by which an AS tracks delegation state (whether a formal registry, a consent store, a policy engine, or another form) is a deployment and product decision outside the scope of this document. Resource servers in security-sensitive deployments may use token introspection ([RFC7662], Token Introspection (Section 8.4)) when request-time revocation state is needed, or may rely on short token lifetimes; the choice of revocation strategy is deployment-specific.

15. Privacy Considerations

Delegation chains can reveal sensitive information about user behavior, enterprise topology, software suppliers, and internal tool composition. Issuers therefore SHOULD disclose only the actor information needed by the relying party for authorization, audit, or policy enforcement.

Cross-domain deployments SHOULD prefer stable but non-reassigned identifiers and SHOULD consider pairwise identifiers for human subjects when a globally correlatable identifier is not required by the use case.

When the same logical entity can appear in different identifier namespaces, such as azp, req_wl, and act.sub, issuers and relying parties SHOULD use explicit issuer scoping and locally trusted mapping rules rather than string equality alone to determine whether those identifiers refer to the same entity.

Issuers SHOULD minimize disclosure of prior actors by audience and token-design decisions made before issuance. Once an issuer chooses to preserve a delegation chain in a token under this profile, it SHOULD preserve the validated chain intact for that token. If local privacy requirements would require omitting a chain element that would otherwise be security-relevant to the recipient's evaluation, the issuer SHOULD reject the request rather than silently truncating the chain.

The txn claim in Transaction Tokens ([I-D.ietf-oauth-transaction-tokens]) is a stable, globally unique identifier shared across all tokens in a single business transaction. When Transaction Tokens cross organizational boundaries, txn enables cross-domain correlation of all service calls within a transaction by any party that observes multiple tokens. Cross-domain propagation and lifetime rules for txn are governed by [I-D.ietf-oauth-transaction-tokens]; deployments SHOULD consult that specification's privacy guidance when propagating Transaction Tokens across trust-domain boundaries. This document notes that txn values, like other stable identifiers, should be treated as sensitive in contexts where cross-domain correlation of user activity is not required or authorized.

The act.sub_profile claim discloses the entity type of the actor, including values such as ai_agent that reveal that a request is being made by an automated agent on behalf of the subject. In some jurisdictions or deployment contexts, this disclosure may be legally significant or may reveal sensitive information about user behavior and tool composition that should not flow across trust-domain boundaries. Issuers SHOULD consider audience-specific disclosure constraints when including act.sub_profile in cross-domain tokens, and SHOULD omit or suppress actor entity-type values when the recipient does not require them for authorization, audit, or policy enforcement.

The req_wl claim in Transaction Tokens can also expose sensitive information about internal workload topology and service composition. Transaction Token Services SHOULD disclose req_wl only to relying parties that need that information for authorization, audit, or policy enforcement, and SHOULD avoid propagating internal-only workload identifiers across trust-domain boundaries unless such disclosure is explicitly required by the deployment.

16. IANA Considerations

16.1. OAuth URI Registration

This document requests IANA to register the following value in the "OAuth URI" registry:

* URN: urn:ietf:params:oauth:grant-profile:actor-profile
* Common Name: OAuth Actor Profile for Delegation
* Change Controller: IETF
* Reference: Authorization Server Metadata (Section 10.1) of this document

16.2. OAuth Authorization Server Metadata Registry

This document requests IANA to register the following values in the "OAuth Authorization Server Metadata" registry ([RFC8414]):

* Metadata Name: actor_profile_token_exchange
* Metadata Description: JSON object advertising coarse Token Exchange capabilities for requests in which actor-profile processing can apply
* Change Controller: IETF
* Reference: Authorization Server Metadata (Section 10.1) of this document

16.3. OAuth Protected Resource Metadata Registry

This document requests IANA to register the following values in the "OAuth Protected Resource Metadata" registry ([RFC9728]):

* Metadata Name: actor_profile_required

* Metadata Description: Boolean indicating whether the RS advertises that delegated requests for this resource are expected to provide actor-profile information conforming to this document's semantics

* Change Controller: IETF

* Reference: Protected Resource Metadata (Section 10.2) of this document

16.4. OAuth Error Registry

This document requests IANA to register the following value in the "OAuth Extensions Error Registry" ([RFC6749], Section 11.4):

* Error Name: actor_unauthorized

* Error Usage Location: Token endpoint response, resource server response

* Related Protocol Extension: OAuth Actor Profile for Delegation

* Change Controller: IETF

* Reference: Error Responses (Section 9) of this document

16.5. OAuth Token Introspection Response Registry

This document requests IANA to register the following value in the "OAuth Token Introspection Response" registry ([RFC7662], Section 3.3):

* Claim Name: chain_complete

* Claim Description: Boolean indicating whether the act delegation chain in the introspection response is complete. When false, one or more inner act chain entries have been omitted from the response for privacy reasons. When absent, the chain SHOULD be treated as complete unless local policy or deployment context indicates otherwise.

* Change Controller: IETF

* Reference: Token Introspection (Section 8.4) of this document

16.6. JWT Claims Registry

This document does not request independent JWT Claims Registry entries for the act object sub-claims (iss, sub_profile, and any extension claims) it defines or profiles. These values appear only within the JSON object value of the act claim, which is already registered in the JWT Claims Registry by [RFC8693]. Sub-object keys within a registered claim are scoped to that claim's JSON object and do not require separate top-level registry entries.

16.7. OAuth Token Type Registry

This document makes no independent requests to the "OAuth Token Type" registry for urn:ietf:params:oauth:token-type:txn_token. That URI is defined and registered by [I-D.ietf-oauth-transaction-tokens]. Its inclusion as a defined value for actor_profile_token_exchange.requested_token_types_supported in Authorization Server Metadata (Section 10.1) is contingent on the progression of [I-D.ietf-oauth-transaction-tokens].

16.8. OAuth Entity Profiles Registry

This document makes no independent requests to the "OAuth Entity Profiles" registry. It normatively depends on the "Actor Profile" usage location, the actor array in entity_profiles_supported, and the registration of user, service, and ai_agent with that usage location, all of which are defined and requested by [I-D.mora-oauth-entity-profiles]. The IANA actions for those entries are contingent on the progression of [I-D.mora-oauth-entity-profiles].

17. References

17.1. Normative References

[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/info/rfc3986>.

[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/info/rfc6750>.

[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/info/rfc7009>.

[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/info/rfc7519>.

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

[RFC7521] Campbell, B., Mortimore, C., Jones, M., and Y. Goland, "Assertion Framework for OAuth 2.0 Client Authentication and Authorization Grants", RFC 7521, DOI 10.17487/RFC7521, May 2015, <https://www.rfc-editor.org/info/rfc7521>.

[RFC7523] Jones, M., Campbell, B., and C. Mortimore, "JSON Web Token (JWT) Profile for OAuth 2.0 Client Authentication and Authorization Grants", RFC 7523, DOI 10.17487/RFC7523, May 2015, <https://www.rfc-editor.org/info/rfc7523>.

[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/info/rfc8705>.

[RFC7662] Richer, J., Ed., "OAuth 2.0 Token Introspection", RFC 7662, DOI 10.17487/RFC7662, October 2015, <https://www.rfc-editor.org/info/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/info/rfc7800>.

[RFC8126] Cotton, M., Leiba, B., and T. Narten, "Guidelines for Writing an IANA Considerations Section in RFCs", BCP 26, RFC 8126, DOI 10.17487/RFC8126, June 2017, <https://www.rfc-editor.org/info/rfc8126>.

[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/info/rfc8414>.

[RFC9728] Jones, M.B., Hunt, P., and A. Parecki, "OAuth 2.0 Protected Resource Metadata", RFC 9728, DOI 10.17487/RFC9728, April 2025, <https://www.rfc-editor.org/info/rfc9728>.

[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/info/rfc8693>.

[RFC9068] Bertocci, V., "JSON Web Token (JWT) Profile for OAuth 2.0 Access Tokens", RFC 9068, DOI 10.17487/RFC9068, October 2021, <https://www.rfc-editor.org/info/rfc9068>.

[RFC8707] Campbell, B., Bradley, J., and H. Tschofenig, "Resource Indicators for OAuth 2.0", RFC 8707, DOI 10.17487/RFC8707, February 2020, <https://www.rfc-editor.org/info/rfc8707>.

[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/info/rfc9449>.

[I-D.ietf-oauth-transaction-tokens] Tulshibagwale, A., Fletcher, G., and P. Kasselman, "Transaction Tokens", Work in Progress, Internet-Draft, draft-ietf-oauth-transaction-tokens-08, 2 March 2026, <https://datatracker.ietf.org/doc/html/draft-ietf-oauth-transaction-tokens-08>.

[I-D.ietf-wimse-workload-creds] Campbell, B., Salowey, J. A., Schwenkschuster, A., Sheffer, Y., and Y. Rosomakho, "WIMSE Workload Credentials", Work in Progress, Internet-Draft, draft-ietf-wimse-workload-creds-00, 3 November 2025, <https://datatracker.ietf.org/doc/html/draft-ietf-wimse-workload-creds-00>.

[I-D.ietf-wimse-wpt] Campbell, B. and A. Schwenkschuster, "WIMSE Workload Proof Token", Work in Progress, Internet-Draft, draft-ietf-wimse-wpt-01, 2 March 2026, <https://datatracker.ietf.org/doc/html/draft-ietf-wimse-wpt-01>.

[I-D.mora-oauth-entity-profiles] Mora, S. C., Dingle, P., and K. McGuinness, "OAuth Entity Profiles", 17 April 2026, <https://www.ietf.org/archive/id/draft-mora-oauth-entity-profiles-01.txt>.

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

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

17.2. Informative References

[RFC9493] Backman, A., Ed., Scurtescu, M., and P. Jain, "Subject Identifiers for Security Event Tokens", RFC 9493, DOI 10.17487/RFC9493, December 2023, <https://www.rfc-editor.org/info/rfc9493>.

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

[RFC9700] Lodderstedt, T., Bradley, J., Labunets, A., and D. Fett, "Best Current Practice for OAuth 2.0 Security", BCP 240, RFC 9700, DOI 10.17487/RFC9700, January 2025, <https://www.rfc-editor.org/info/rfc9700>.

[I-D.parecki-oauth-jwt-dpop-grant] Parecki, A., "JWT Authorization Grants with DPoP", 30 January 2026, <https://datatracker.ietf.org/doc/html/draft-parecki-oauth-jwt-dpop-grant-01>.

[I-D.ietf-oauth-identity-chaining] Schwenkschuster, A., Kasselmann, P., Burgin, K., Jenkins, M., Campbell, B., and A. Parecki, "OAuth Identity and Authorization Chaining Across Domains", Work in Progress, Internet-Draft, draft-ietf-oauth-identity-chaining-10, 24 April 2026, <https://datatracker.ietf.org/doc/html/draft-ietf-oauth-identity-chaining-10>.

[I-D.ietf-oauth-identity-assertion-authz-grant] Parecki, A., McGuinness, K., and B. Campbell, "Identity Assertion JWT Authorization Grant", 22 April 2026, <https://www.ietf.org/archive/id/draft-ietf-oauth-identity-assertion-authz-grant-03.txt>.

[OpenID.Core] OpenID Foundation, "OpenID Connect Core 1.0", 8 November 2014, <https://openid.net/specs/openid-connect-core-1_0.html>.

[OpenID.Federation]
OpenID Foundation, "OpenID Federation 1.0", 1 May 2024,
<https://openid.net/specs/openid-federation-1_0.html>.

Appendix A. Service-to-Service Delegation Example

This appendix gives a non-AI example of the actor profile in a same-domain service-to-service delegation flow. A payroll batch processor acts on behalf of a human payroll administrator to call a payroll API; the payroll API then exchanges that access token for an internal Transaction Token used to write an audit record.

A.1. Scenario

Party	Identifier
Payroll Administrator	https://idp.example.com/users/pat
Enterprise AS	https://as.example.com
Payroll Batch Processor	https://services.example.com/payroll-batch
Payroll API	https://services.example.com/payroll-api
Audit TTS	https://tts.example.com
Audit Service	https://internal.example.com/audit

Table 4

The batch processor is an OAuth client and also the acting service. The client registration remains identified by client_id; the acting service is represented explicitly in act.sub.

A.2. Access Token

The Enterprise AS issues a JWT access token for the Payroll API:

```
{
  "iss": "https://as.example.com",
  "sub": "https://idp.example.com/users/pat",
  "sub_profile": "user",
  "client_id": "payroll-batch-client",
  "aud": "https://services.example.com/payroll-api",
  "scope": "payroll:run",
  "cnf": { "jkt": "SvcJKT-123" },
  "act": {
    "sub": "https://services.example.com/payroll-batch",
    "iss": "https://as.example.com",
    "sub_profile": "service"
  }
}
```

This is a single-hop actor object: act is present, but it contains no nested act. sub identifies the payroll administrator, while act.sub identifies the service exercising that administrator's authorization.

A.3. Transaction Token

After processing the payroll request, the Payroll API exchanges the inbound access token at the Audit TTS to call the internal Audit Service. The Payroll API is the requesting workload (req_wl). The TTS validates the inbound delegation chain, preserves it as an inner act, and adds a new outermost actor for the Payroll API:

```
{
  "iss": "https://tts.example.com",
  "sub": "https://idp.example.com/users/pat",
  "sub_profile": "user",
  "req_wl": "https://services.example.com/payroll-api",
  "aud": "https://internal.example.com/audit",
  "scope": "audit:create",
  "txn": "550e8400-e29b-41d4-a716-446655440099",
  "cnf": { "jkt": "ApiJKT-456" },
  "act": {
    "sub": "https://services.example.com/payroll-api",
    "iss": "https://as.example.com",
    "sub_profile": "service",
    "act": {
      "sub": "https://services.example.com/payroll-batch",
      "iss": "https://as.example.com",
      "sub_profile": "service"
    }
  }
}
```

The subject remains the payroll administrator. The new outermost actor is the Payroll API (the workload presenting the exchange request), and the preserved inner actor is the payroll batch processor. This example demonstrates that the profile applies equally to non-AI service delegation and to transformations from a single-hop actor object into a multi-hop delegation chain.

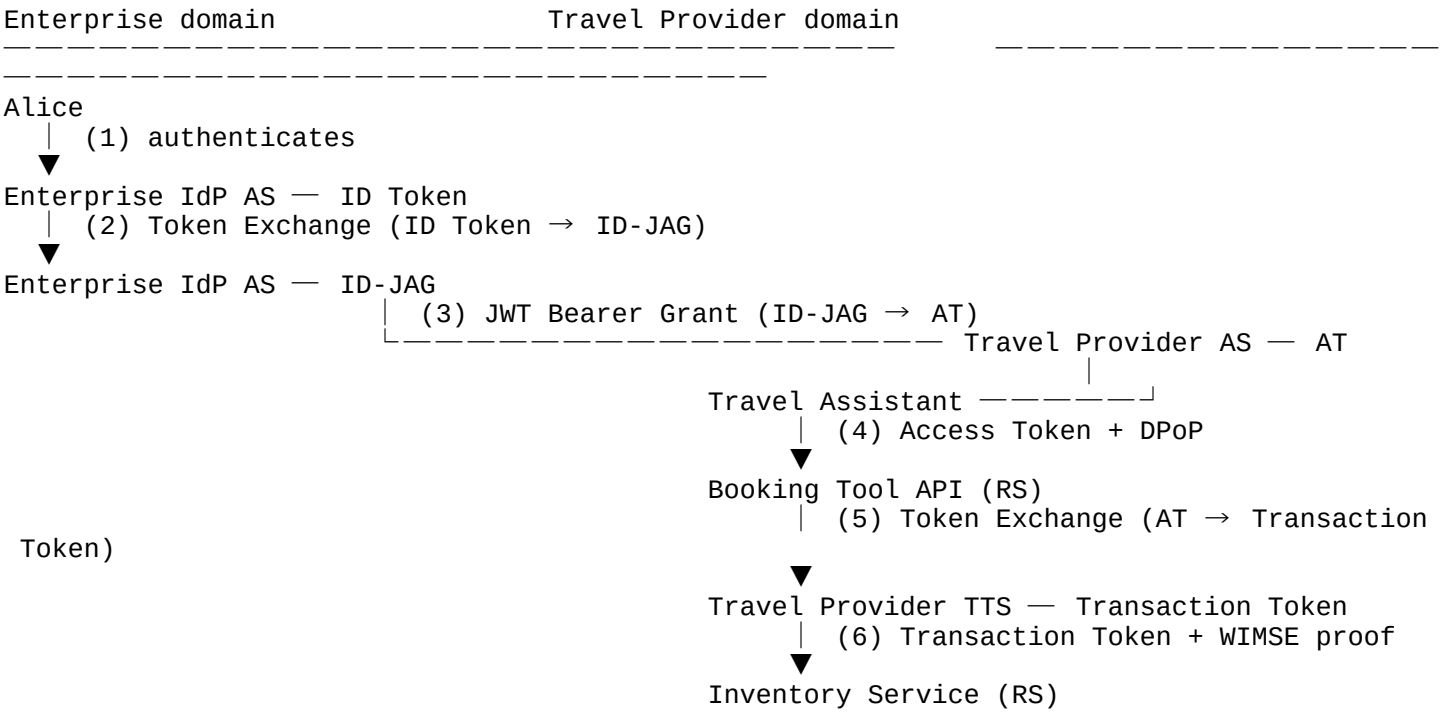
Appendix B. Cross-Domain AI Agent Flow: ID Token to Transaction Token

This appendix traces a single user request across two trust domains, highlighting the actor-profile claim structures and processing requirements specific to this document. Standard validation steps (JWT signature verification, sender-constrained access token proof, Transaction Token presenter proof, and Token Exchange mechanics) are delegated to the underlying token specifications and deployment profile.

All claim values, JKT thumbprints, and domain names are synthetic.

B.1. Scenario and Parties

Alice’s travel-assistant agent authenticates to the Enterprise IdP AS to obtain an ID Token. The agent then performs a Token Exchange at the same AS to obtain the ID-JAG. The ID-JAG is then presented to the Travel Provider AS using the JWT bearer grant, as described in Authorization Grant Processing (Section 4.2). The agent exchanges the ID-JAG for an access token at the Travel Provider AS and calls the Booking Tool API. The Booking Tool exchanges the access token for a Transaction Token to call an internal inventory service.


Enterprise domain
Travel Provider domain
Alice
(1) authenticates
Enterprise IdP AS — ID Token
(2) Token Exchange (ID Token → ID-JAG)
Enterprise IdP AS — ID-JAG
(3) JWT Bearer Grant (ID-JAG → AT)
Travel Provider AS — AT
Travel Assistant
(4) Access Token + DPoP
Booking Tool API (RS)
(5) Token Exchange (AT → Transaction Token)
Travel Provider TTS — Transaction Token
(6) Transaction Token + WIMSE proof
Inventory Service (RS)

Party	Identifier	Trust Domain
Alice	https://idp.enterprise.example/users/alice	Enterprise
Enterprise IdP AS	https://as.enterprise.example	Enterprise
Travel Assistant	https://agents.enterprise.example/travel-assistant	Enterprise
Travel Provider AS	https://as.travel-provider.example	Travel Provider
Travel Provider TTS	https://tts.travel-provider.example	Travel Provider
Booking Tool	https://tools.travel-provider.example/booking-tool	Travel Provider
Inventory Service	https://internal.travel-provider.example/inventory	Travel Provider

Table 5

Presenter key bindings:

Principal	JWK Thumbprint (jkt)
Travel Assistant	AgentJKT-NzbLsXh8uDCcd7MN
Booking Tool	ToolJKT-0ZcOCORZNYy9ZhHi

Table 6

B.2. Capability Discovery (Preflight)

The agent consults the Travel Provider AS metadata (Metadata and Discovery (Section 10)) as an advisory compatibility check before initiating the flow:

```
{
  "issuer": "https://as.travel-provider.example",
  "grant_types_supported": [
    "urn:ietf:params:oauth:grant-type:jwt-bearer",
    "urn:ietf:params:oauth:grant-type:token-exchange"
  ],
  "authorization_grant_profiles_supported": [
    "urn:ietf:params:oauth:grant-profile:id-jag",
    "urn:ietf:params:oauth:grant-profile:actor-profile"
  ],
  "actor_profile_token_exchange": {
    "subject_token_types_supported": [
      "urn:ietf:params:oauth:token-type:jwt"
    ],
    "actor_token_types_supported": [
      "urn:ietf:params:oauth:token-type:jwt"
    ],
    "requested_token_types_supported": [
      "urn:ietf:params:oauth:token-type:access_token",
      "urn:ietf:params:oauth:token-type:txn_token"
    ]
  },
  "entity_profiles_supported": {
    "subject": ["user", "ai_agent"],
    "actor":   ["user", "ai_agent", "service"]
  }
}
```

The agent confirms that its sub_profile (ai_agent) is in entity_profiles_supported.actor, that ID-JAG and actor-profile grants are advertised in authorization_grant_profiles_supported, and that the planned input/output token types appear in actor_profile_token_exchange. These signals are coarse compatibility indicators only; the agent proceeds because the advertised capabilities cover its planned path.

B.3. Step 1: User Authentication (ID Token)

Alice authenticates to the Enterprise IdP AS, which issues an ID Token. An ID Token implicitly identifies a user; the entity type is not carried in a sub_profile claim and is established by the AS in subsequent issued tokens (Step 2):

```
{
  "iss": "https://as.enterprise.example",
  "sub": "https://idp.enterprise.example/users/alice",
  "aud": "https://agents.enterprise.example/travel-assistant",
  "exp": 1743379200,
  "iat": 1743375600
}
```

B.4. Step 2: Enterprise Token Exchange (ID Token to ID-JAG)

The agent presents Alice's ID Token as subject_token in a Token Exchange request to the Enterprise IdP AS, requesting an ID-JAG ([I-D.ietf-oauth-identity-assertion-authz-grant]). The agent's RFC 7523 client assertion serves as both client_assertion (for client authentication) and actor_token (for actor identity), per JWT Client Assertion (Section 6.3.1). The Enterprise IdP AS authenticates the client, verifies the ID Token audience matches that client, and uses local delegation policy to construct the issued ID-JAG:

```
POST /token HTTP/1.1
Host: as.enterprise.example
Content-Type: application/x-www-form-urlencoded
DPoP: <AgentJKT-proof>

grant_type=urn%3Aietf%3Aparams%3Aoauth%3Agrant-type%3Atoken-exchange
&subject_token=<alice-id-token>
&subject_token_type=urn%3Aietf%3Aparams%3Aoauth%3Atoken-type%3Aid_token
&requested_token_type=urn%3Aietf%3Aparams%3Aoauth%3Atoken-type%3Aid-jag
&audience=https%3A%2F%2Fas.travel-provider.example%2F
&resource=https%3A%2F%2Fas.travel-provider.example
&scope=booking%3Acreate
&client_id=https%3A%2F%2Fagents.enterprise.example%2Ftravel-assistant
&client_assertion_type=urn%3Aietf%3Aparams%3Aoauth%3Aclient-assertion-type%3Ajwt-bearer
&client_assertion=<travel-assistant-client-assertion>
&actor_token=<travel-assistant-client-assertion>
&actor_token_type=urn%3Aietf%3Aparams%3Aoauth%3Atoken-type%3Ajwt
```

The Enterprise IdP AS applies scope reduction and validates the client-bound proof-of-possession according to RFC 9449. In this example, it binds the issued ID-JAG to the key demonstrated in the DPoP proof, validates the shared JWT under both the client-authentication and actor_token rules for RFC 7523 client assertions, determines from local policy that the authenticated client is the delegated actor for Alice in this flow, and issues the ID-JAG as a JWT output of Token Exchange with the delegation chain established per Actor Profile for Delegation (Section 3):

```
{
  "iss": "https://as.enterprise.example",
  "sub": "https://idp.enterprise.example/users/alice",
  "sub_profile": "user",
  "client_id": "https://agents.enterprise.example/travel-assistant",
  "azp": "https://agents.enterprise.example/travel-assistant",
  "aud": "https://as.travel-provider.example/token",
  "jti": "ent-idj-20260401-001",
  "exp": 1743379200,
  "iat": 1743375600,
  "scope": "booking:create",
  "cnf": { "jkt": "AgentJKT-NzbLsXh8uDCcd7MN" },
  "act": {
    "sub": "https://agents.enterprise.example/travel-assistant",
    "iss": "https://as.enterprise.example",
    "sub_profile": "ai_agent"
  }
}
```

The act object records the agent as the authorized actor. The client_id and azp values identify the OAuth client used in the exchange, while act.sub identifies the delegated actor. In this example those identifiers are the same URI, so the shared RFC 7523 client assertion and actor_token remain conformant with [RFC7523] while still making the actor explicit in the issued token. The top-level cnf.jkt is set to AgentJKT because the agent is the current presenter at this stage.

B.5. Step 3: Agent Exchanges ID-JAG for Access Token at Travel Provider AS

The agent presents the ID-JAG as a JWT Bearer authorization grant ([RFC7523]) to the Travel Provider AS, which processes it as an ID-JAG per [I-D.ietf-oauth-identity-assertion-authz-grant] with the actor-profile rules in Authorization Grant Processing (Section 4.2):

```
POST /token HTTP/1.1
Host: as.travel-provider.example
Content-Type: application/x-www-form-urlencoded
DPoP: <AgentJKT-proof>

grant_type=urn%3Aietf%3Aparams%3Aoauth%3Agrant-type%3Ajwt-bearer
&assertion=<id-jag>
&scope=booking%3Acreate
```

The Travel Provider AS performs actor-profile processing per Authorization Grant Processing (Section 4.2): it verifies the request’s DPoP proof against the top-level cnf.jkt in the inbound ID-

JAG and checks that act.sub_profile (ai_agent) is permitted as an actor for the requested scope under local policy. It issues an access token preserving the delegation chain:

```
{
  "iss": "https://as.travel-provider.example",
  "sub": "https://idp.enterprise.example/users/alice",
  "sub_profile": "user",
  "client_id": "https://agents.enterprise.example/travel-assistant",
  "azp": "https://agents.enterprise.example/travel-assistant",
  "aud": "https://api.travel-provider.example",
  "jti": "tp-at-20260401-001",
  "exp": 1743379200,
  "iat": 1743375600,
  "scope": "booking:create",
  "cnf": { "jkt": "AgentJKT-NzbLsXh8uDCcd7MN" },
  "act": {
    "sub": "https://agents.enterprise.example/travel-assistant",
    "iss": "https://as.enterprise.example",
    "sub_profile": "ai_agent"
  }
}
```

Alice’s sub and sub_profile are preserved verbatim from the ID-JAG (JWT Access Token Output (Section 6.5.2)). The Travel Provider AS does not translate or substitute the enterprise subject identifier. The client_id and azp values reflect the OAuth client identity from the request context (the same agent URI appears in both the inbound ID-JAG and the outbound access token because the agent is both the asserting client and the actor in this flow), but they do not replace act.sub as the authoritative delegated-actor identifier.

B.6. Step 4: Agent Calls Booking Tool API

The agent presents the access token with a DPoP proof:

```
POST /bookings HTTP/1.1
Host: api.travel-provider.example
Authorization: DPoP <tp-access-token>
DPoP: <AgentJKT-proof>
Content-Type: application/json

{"origin": "SFO", "destination": "NYC", "depart": "2026-04-15"}
```

The Booking Tool RS applies authorization of the (sub, outermost act.sub) pair (Resource Server Processing (Section 8)): it evaluates Alice (sub, sub_profile: user) together with the Travel Assistant (act.sub, sub_profile: ai_agent) for the requested operation. The

act.sub_profile value is checked against entity_profiles_supported.actor per Authorization Server Metadata (Section 10.1).

B.7. Step 5: Booking Tool Exchanges Access Token for Transaction Token

The Booking Tool cannot reuse the received access token for internal calls: it is sender-constrained to AgentJKT, which the Booking Tool does not possess. It requests a Transaction Token from the TTS. In this example, the TTS receives the Booking Tool's WIMSE Workload Identity Token (WIT) as the Token Exchange actor_token and validates a Workload Proof Token (WPT). The WIT identifies the Booking Tool and carries its confirmation key, while the WPT proves possession of that key and binds the request to the accompanying access token:

```
POST /token HTTP/1.1
Host: tts.travel-provider.example
Content-Type: application/x-www-form-urlencoded
Workload-Proof-Token: <tool-wpt-with-wth-and-ath>

grant_type=urn%3Aietf%3Aparams%3Aoauth%3Agrant-type%3Atoken-exchange
&subject_token=<tp-access-token>
&subject_token_type=urn%3Aietf%3Aparams%3Aoauth%3Atoken-type%3Aaccess_token
&actor_token=<booking-tool-wit>
&actor_token_type=urn%3Aietf%3Aparams%3Aoauth%3Atoken-type%3Ajwt
&requested_token_type=urn%3Aietf%3Aparams%3Aoauth%3Atoken-type%3Atxn_token
&audience=https%3A%2F%2Ftravel-provider.example
&scope=inventory%3Acheck
&rctx={"req_ip":"198.51.100.42"}
```

The WIT is therefore the JWT actor_token defined by this profile, while the WPT provides the accompanying proof of possession required by the workload-credential profile.

The TTS applies actor-profile processing per Transaction Token Output Rules (Section 7.4): it preserves sub and sub_profile from the subject_token, sets req_wl to the authenticated Booking Tool, and creates a new outermost act object for the Booking Tool while nesting the subject_token's existing act claim beneath it. In this WIMSE-based deployment, the underlying Transaction Token mechanism also binds the issued token to the Booking Tool's presenter key (ToolJKT) identified in the WIT confirmation claim:

```
{
  "iss": "https://tts.travel-provider.example",
  "sub": "https://idp.enterprise.example/users/alice",
  "sub_profile": "user",
  "scope": "inventory:check",
  "req_wl": "https://tools.travel-provider.example/booking-tool",
  "aud": "https://travel-provider.example",
  "jti": "txn-tok-20260401-001",
  "txn": "550e8400-e29b-41d4-a716-446655440001",
  "exp": 1743375750,
  "iat": 1743375650,
  "tctx": {
    "action": "check-availability",
    "origin": "SFO",
    "destination": "NYC",
    "depart": "2026-04-15"
  },
  "rctx": { "req_ip": "198.51.100.42" },
  "cnf": { "jkt": "ToolJKT-0ZcOCORZNYy9ZhHi" },
  "act": {
    "sub": "https://tools.travel-provider.example/booking-tool",
    "iss": "https://as.travel-provider.example",
    "sub_profile": "service",
    "act": {
      "sub": "https://agents.enterprise.example/travel-assistant",
      "iss": "https://as.enterprise.example",
      "sub_profile": "ai_agent"
    }
  }
}
```

The presenter binding rotates at this step: cnf.jkt is now ToolJKT because the Booking Tool is the current presenter.

B.8. Step 6: Booking Tool Calls Inventory Service

```
GET /inventory?origin=SFO&dest=NYC&depart=2026-04-15 HTTP/1.1
Host: internal.travel-provider.example
Txn-Token: <txn-token>
Workload-Identity-Token: <booking-tool-wit>
Workload-Proof-Token: <tool-wpt-with-wth-and-tth>
```

The Inventory Service validates the WIT and WPT per the WIMSE specifications, then applies authorization of the (sub, outermost act.sub) pair (Resource Server Processing (Section 8)): Alice (sub) governs data access policy (e.g., travel tier), and the Booking Tool (act.sub) is the authorized internal workload for that request. The req_wl claim provides consistent TTS workload context for the same

service in this example. The nested act.act.sub (the Travel Assistant) is carried as prior delegation context and is not evaluated for access control at this internal tier, consistent with the guidance on inner actors in Actor Authorization (Section 8.1).

B.9. Summary of Token Transformations

Step	Token	sub	req_wl	act.sub (outermost)	act.act.sub (nested)	cnf.jkt
1	ID Token	Alice				
2	ID-JAG	Alice		Travel Assistant		AgentJKT
3	Access Token	Alice		Travel Assistant		AgentJKT
4	(API call)	Alice		Travel Assistant		AgentJKT
5	Transaction Token	Alice	Booking Tool	Booking Tool	Travel Assistant	ToolJKT
6	(internal call)	Alice	Booking Tool	Booking Tool	Travel Assistant	ToolJKT

Table 7

Key observations:

* In this example, sub (Alice) is unchanged across all trust domains and token transformations.

* The presenter-binding key rotates once, at Step 5 when the TTS re-binds the Transaction Token to the Booking Tool’s key.

* At Step 5 the TTS creates a new outermost act for the Booking Tool and nests the prior act chain beneath it.

Acknowledgments

The author thanks the OAuth Working Group for the foundational work on Token Exchange (RFC 8693), JWT-formatted access tokens (RFC 9068), DPoP (RFC 9449), and Transaction Tokens, upon which this document builds. The motivating use cases for this work emerged from the deployment of AI agent systems that require cross-domain delegation with explicit delegation chains and carried-forward delegation state within the trust model of the issuing domains.

Author’s Address

Karl McGuinness
Independent
Email: public@karlmcguinness.com