SPICE
Internet-Draft
Intended status: Standards Track
Expires: 28 September 2026

A. Prasad
Oracle
R. Krishnan
JPMorgan Chase & Co
D. Lopez
Telefonica
S. Addepalli
Aryaka

27 March 2026

Cryptographically Verifiable Actor Chains for OAuth 2.0 Token Exchange
draft-mw-spice-actor-chain-04

Abstract

Multi-hop service-to-service and agentic workflows need a standardized way to preserve and validate delegation-path continuity across successive token exchanges. This document defines six actor-chain profiles for OAuth 2.0 Token Exchange {{!RFC8693}}. {{!RFC8693}} permits nested act claims, but prior actors remain informational only and token exchange does not define how a delegation path is preserved and validated across successive exchanges.

This document profiles delegation-chain tokens and defines profile-specific processing for multi-hop workflows. The six profiles are: Declared Full Disclosure; Declared Subset Disclosure; Declared Actor-Only Disclosure; Verified Full Disclosure; Verified Subset Disclosure; and Verified Actor-Only Disclosure.

These profiles preserve the existing meanings of sub, act, and may_act. They support same-domain and cross-domain delegation and provide different tradeoffs among visible chain-based authorization, cryptographic accountability, auditability, privacy, and long-running workflow support. Plain RFC 8693 impersonation-shaped outputs remain valid RFC 8693 behavior but are outside this profile family.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at https://datatracker.ietf.org/drafts/current/.

Copyright Notice

Table of Contents

1. Part I. Core Specification

2. Introduction

In service-to-service, tool-calling, and agent-to-agent systems, including those implementing the Model Context Protocol (MCP) and the Agent2Agent (A2A) protocol, one workload often receives a token, performs work, and then exchanges that token to call another workload. {{!RFC8693}} defines token exchange and the act claim for the current actor, but it does not define a standardized way to preserve and validate delegation-path continuity across a sequence of exchanges.

This document defines six interoperable actor-chain profiles for OAuth 2.0 Token Exchange. They preserve the existing meanings of sub, act, and may_act, and they define when nested act becomes authoritative visible actor-chain state for profile-controlled processing. Depending on the selected profile, the next hop can see the full visible chain, an Authorization-Server- selected subset of that chain, only the current actor, or no inline act at all.

The profiles are organized in two branches. The declared branch relies on the Authorization Server to assert visible chain continuity. The verified branch adds actor-signed step proofs and cumulative commitment state so that later participants can validate stronger continuity and audit evidence. All profiles support both same-domain continuation and cross-domain re-issuance. This specification does not require any particular presenter-binding or actor- authentication mechanism; those remain deployment-specific.

This specification defines a JWT / JSON Web Signature (JWS) binding for the interoperable base protocol. A future version or companion specification MAY define an equivalent COSE/CBOR binding.

3. Terminology

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

This document also leverages terminology from OAuth 2.0 Token Exchange {{!RFC8693}}, the SPICE Architecture {{!I-D.ietf-spice-arch}}, and the RATS Architecture {{!RFC9334}}.

* *Actor*: A workload, service, application component, agent, or other authenticated entity that receives a token, performs work, and MAY subsequently act toward another actor.

* *Current actor*: The authenticated entity presently performing token exchange.

* *Presenting actor*: The actor that presents an inbound token to a recipient.

Example: when B exchanges a token at the Authorization Server, B is the current actor. When B later presents the resulting token to C, B is the presenting actor.

* *Recipient*: The actor or resource server identified as the intended target of an issued token.

* *Actor chain*: The ordered sequence of actors that have acted so far in one workflow instance.

* *Ordinary token*: The access token issued for presentation to the next hop. It is distinct from step proofs, bootstrap context handles, commitment objects, and receiver acknowledgments.

* *Visible chain*: The ordered actor sequence represented by the act structure of an artifact when actp is present. The outermost act identifies the current actor. Any nested act members identify prior visible actors only.

* *Actor-visible chain*: The exact ordered actor sequence that the current actor is permitted to know and extend for the next hop. In the verified profiles, this is the exact visible chain that the current actor verified on the inbound hop, with that actor appended when it later acts.

* *Authoritative workflow chain state*: Authorization-Server-retained state for the accepted workflow instance. It MAY be richer than the visible chain disclosed in any given issued token. It is used for continuity, branch correlation, forensic review, legal audit, and any policy-controlled future redisclosure. This specification does not require that such retained state be disclosed inline in ordinary tokens.

In the full-disclosure profiles, the visible chain, the actor-visible chain, and the authoritative workflow chain state collapse to the same chain. They diverge only under subset and actor-only profiles, where the visible chain may be narrower than the actor-visible chain, which in turn may be narrower than the authoritative workflow chain state.

* *Proof-bound chain state*: The cumulative cryptographic state carried in actc that binds prior accepted chain state to a newly accepted hop.

* *Step proof*: A profile-defined proof signed by the current actor that binds that actor’s participation to the workflow, prior accepted state, the profile-defined actor-visible chain for the hop, and target context.

* *Target context*: The canonical representation of the next-hop target that a profile-defined proof or acknowledgment binds to. It always includes the intended audience and MAY additionally include other target-selection inputs. If no such additional inputs are used, it is identical to aud.

* *Bootstrap context*: An opaque handle issued by the Authorization Server only to start a verified profile workflow. It lets the initial actor redeem bound bootstrap state at the token endpoint without carrying that state inline.

* *Actor-chain identifier (acti)*: A stable identifier minted once at workflow start and retained for the lifetime of the workflow instance.

* *Cross-domain re-issuance*: A second token exchange performed at another domain's Authorization Server in order to obtain a local token trusted by the next recipient, without extending the actor chain.

* *Home Authorization Server*: The Authorization Server at which the current actor normally performs the chain-extending token exchange for the next hop. In same-domain operation it is the issuer that validates prior chain state and issues the next ordinary token.

* *Continuity*: The property that the presenting actor of an inbound artifact is the same actor that the workflow state and any required disclosed current-actor information identify as the expected presenter.

* *Append-only processing*: The rule that a new actor is appended to the prior accepted workflow chain state, and to any disclosed visible chain fragment derived from it, without insertion, deletion, reordering, or modification of prior actors.

* *Terminal recipient*: A recipient that performs work locally and does not extend the actor chain further.

* *Refresh-Exchange*: A token-exchange operation by the same current actor that preserves accepted chain state while refreshing transport characteristics such as expiry according to local policy.

* *Local policy*: Deployment-specific authorization and risk rules used when this specification defers a decision to the Authorization Server or recipient. Local policy can include trust relationships, authenticated client identity, permitted audiences or resources, chain-depth limits, profile support, and business or risk controls.

4. Relationship to RFC 8693 Claims

This specification extends OAuth 2.0 Token Exchange {{!RFC8693}} without changing the base meanings of sub, act, or may_act. It profiles delegation-chain workflow continuity and profile-controlled disclosure across token exchange hops.

When actp is absent, act has ordinary RFC 8693 semantics. Nested prior act claims, if present, remain informational only for access-control purposes, consistent with {{!RFC8693}}.

The following rules apply when actp is present and points to a profile defined by this specification:

* sub continues to identify the subject of the issued token.
* acti identifies the stable workflow instance for the accepted actor-chain state.
* If act is present, it is the authoritative disclosed actor-chain fragment for that artifact. The outermost act, when present, identifies the disclosed current actor. Each nested act member, when present, identifies the immediately prior disclosed actor.
* Reduced actor disclosure, including omission of prior actors, omission of the current actor from a disclosed subset, or omission of act entirely, does not turn a profiled token into an ordinary impersonation output. When actp is present, the token remains a delegation-chain token under this specification.
* Plain RFC 8693 outputs that do not carry this specification's profile signals remain outside this profile family whether or not they contain act.

4.1. Disclosure Invariant

For any selected actp, the Authorization Server is the policy decision and policy enforcement point for actor-chain disclosure. The selected profile defines the maximum actor-chain information, if any, that may be visible inline to the next hop.

Accordingly:

* no other field, identifier encoding, compatibility form, or exchange option may disclose more actor-chain information than the selected profile permits;
* recipients and intermediaries MUST NOT infer hidden actors from omitted entries, identifier structure, or other claims; and
* when a profile hides prior actors or the workflow subject's globally useful identifier, the Authorization Server MUST enforce that outcome across the entire issued artifact, not only within act.

Nothing in this specification redefines RFC 8693 delegation or impersonation semantics. With actp absent, this document adds no new requirements on how nested act is interpreted. With actp present, this document defines an explicit profile-controlled extension in which the visible act structure becomes authoritative for actor-chain processing under that profile.

5. Scope and Model

This document specifies a family of profiles for representing and validating actor progression across a workflow path using OAuth 2.0 Token Exchange.

Implementers primarily interested in interoperable behavior can focus first on Common Basics, Common Validation Procedures, the profile sections, and Appendices A, B, G, and H. Special preserve-state exchanges, metadata, and the deeper enforcement rules are intentionally surfaced later so the reader can learn the ordinary profile flow first. The appendices later in the document contain background, rationale, threat discussion, and operational guidance that may be useful for review and deployment planning but are not required for a first implementation pass.

5.1. Workflow Model

The basic workflow path model is:

{::nomarkdown} <artwork type="ascii-art"> A -> B -> C -> D </artwork> {:/nomarkdown}

The first actor initializes the workflow. Each subsequent actor MAY:

1. validate an inbound token;
2. perform work; and
3. exchange that token for a new token representing itself toward the next hop.

5.2. Profile Summary

This document defines six profiles:

* *Declared Full Disclosure*, which carries the full visible nested act chain in ordinary tokens and relies on AS-asserted chain continuity under a non-collusion assumption;
* *Declared Subset Disclosure*, which carries a recipient-specific disclosed nested act chain in ordinary tokens and relies on the issuing AS for both chain continuity and disclosure policy;
* *Declared Actor-Only Disclosure*, which carries only the outermost current actor in ordinary tokens and makes that actor the sole inline authorization input for the next hop;
* *Verified Full Disclosure*, which is the verified full-disclosure case in which every actor and downstream recipient sees the full visible nested act chain;

* *Verified Subset Disclosure*, which preserves cumulative commitment state and lets the Authorization Server disclose a recipient-specific visible nested act chain while the current actor signs the exact actor-visible chain it was allowed to see and extend; and
* *Verified Actor-Only Disclosure*, which keeps the verified proofs and cumulative commitment state but discloses only the outermost current actor in ordinary tokens.

The six profiles are organized in two branches so that later profiles can be read as deltas, not as full restatements:

* the *declared branch*, rooted at Declared Full Disclosure with two concise disclosure-mode deltas: subset and actor-only; and
* the *verified branch*, consisting of one common verified processing section plus three disclosure modes: full disclosure, subset disclosure, and actor-only disclosure.

Each derived profile inherits all requirements of its branch root or common verified processing section except as modified in that profile. Readers therefore need only read:

* *Declared Full Disclosure* for the declared branch;
* the concise delta sections for the declared subset-disclosure and declared actor-only variants; and
* *Common Processing for the Verified Branch* plus the concise verified profile sections for the verified branch.

Later sections introduce the additional verified-profile mechanisms only where they are needed: a one-time bootstrap at workflow start, a per-hop actor-signed step proof, cumulative commitment state in actc, and the optional hop_ack extension. Every profile still issues the same basic next-hop ordinary token.

The following table is a quick orientation aid.

Profile	Visible act in ordinary tokens	actc	Recipient-specific disclosure	Next-hop authorization basis	Primary trust/ evidence model
Declared Full Disclosure	Full nested chain	No	No	Full visible chain	AS-asserted continuity
Declared Subset	Disclosed nested	No	Yes	Disclosed visible subset	AS-asserted

Disclosure	subset or omitted act			only; undisclosed actors unavailable from the artifact	continuity plus AS disclosure policy
Declared Actor-Only Disclosure	Outermost current actor only	No	Fixed actor-only	Current actor only	AS-asserted continuity plus actor-only disclosure policy
Verified Full Disclosure	Full nested chain	Yes	No	Full visible chain plus commitment continuity	Actor-signed visible-chain proofs plus cumulative commitment
Verified Subset Disclosure	Disclosed nested subset or omitted act	Yes	Yes	Disclosed visible subset only; undisclosed actors unavailable from the artifact, plus commitment continuity	Actor-signed visible-chain proofs plus recipient-specific disclosure
Verified Actor-Only Disclosure	Outermost current actor only	Yes	Fixed actor-only	Current actor only plus commitment continuity	Actor-signed visible-chain proofs plus cumulative commitment with actor-only disclosure

Table 1

5.3. Branching and Non-Goals

Application logic may branch, fan out, and run in parallel. This document standardizes one visible path per issued token, not a full call-graph language. An Authorization Server MAY mint multiple accepted successor tokens from one prior accepted state, each with its own jti and canonical target_context. Such successor tokens MAY share the same acti and earlier workflow history. Deployments that require strict linear continuation MAY instead enforce a local single-successor policy under which the Authorization Server accepts at most one successor from any accepted prior state. This base specification does not define an interoperable on-the-wire signal for single-successor mode, sibling invalidation, or proof that no parallel successor exists. This document does not define merge semantics, sibling-discovery semantics, or inline branch-selection semantics.

Post-facto reconstruction of branching is a forensic or legal-audit concern, not a normal online authorization requirement. Retained Authorization Server records, timestamps, commitment state, and causal links among presenting actors, current actors, and subsequent actors can often reveal much of the effective call graph, but this base specification alone does not guarantee a complete standardized graph across all branches.

An issued token MAY still carry an aud string or array according to JWT and OAuth conventions, but each issued token and each step proof binds exactly one canonical next-hop target_context. Additional target contexts require additional successor tokens.

Repeated ActorID values within one workflow instance are permitted. A sequence such as [A,B,C,D,A,E] denotes that actor A acted more than once in the same workflow instance. Collecting all accepted hop evidence for one acti, such as retained tokens, proofs, commitments, and exchange records, can therefore reconstruct the accepted hop sequence, including repeated-actor revisits.

6. Protocol Overview

6.1. Workflow Progression

The actor chain advances only when an actor acts. Mere receipt of a token does not append the recipient.

If A calls B, and B later calls C, then:

1. A begins the workflow and becomes the initial actor.
2. When A calls B, B validates a token representing A.
3. When B later exchanges that token to call C, B becomes the next actor.
4. C is not appended merely because C received a token. C is appended only if C later acts toward another hop.

Compact end-to-end examples appear in Appendix B.

6.2. Same-Domain and Cross-Domain Hops

Within one trust domain, the current actor exchanges its inbound token at its home Authorization Server, meaning the Authorization Server that validates the prior chain state for that actor and issues the next ordinary token.

Across a trust boundary, if the next recipient does not trust the current Authorization Server directly, the current actor performs a second token exchange at the next domain’s Authorization Server. That second exchange preserves the already-established chain state and does not append the next recipient.

The trust/evidence differences among profiles are summarized in the profile matrix above and discussed further in Appendix J. The special preserve-state cases for cross-domain re-issuance and Refresh-Exchange are defined later, after the ordinary profile flows.

7. Common Basics

7.1. Common Token Requirements

Unless stated otherwise, "ordinary token" below means the access token issued to the current actor for presentation to the next hop. This section is about those tokens, not about verified-profile step proofs, bootstrap context handles, or hop_ack objects.

In the interoperable self-contained ordinary-token binding defined by this document, tokens issued under any profile:

* MUST be short-lived;
* MUST contain:
  - an issuer claim iss;
  - a profile identifier claim actp;
  - an actor-chain identifier claim acti;
  - a subject claim sub;
  - a unique token identifier claim jti;
  - an audience value aud; and
  - an expiry value exp.

A profile-governed ordinary token MAY additionally contain act according to the selected profile and Authorization Server policy.

For interoperability and predictable freshness, deployments SHOULD use ordinary-token lifetimes on the order of minutes rather than hours; a default range of 1 to 10 minutes is RECOMMENDED. Validators SHOULD allow only modest clock skew when evaluating exp, typically no more than 60 seconds unless local clock discipline justifies a tighter bound.

Proof-bound profiles additionally MUST carry actc.

The token claims used by this document have these roles:

* iss identifies the issuer namespace of the ordinary token and of that token's top-level sub;
* actp identifies the selected actor-chain profile;
* acti identifies the workflow instance;
* sub identifies the workflow subject of the token;
* act, when present, carries the authoritative disclosed actor-chain fragment for that artifact; and
* actc, when present, carries cumulative commitment state for stronger tamper evidence, continuity, and auditability.

This base specification defines interoperable direct claim carriage for self-contained ordinary tokens. Deployments that instead use opaque access tokens MAY keep authoritative workflow state only at the issuing Authorization Server and MAY disclose actor-chain information, if any, through a companion token-validation interface such as token introspection {{!RFC7662}}.

This specification does not define such companion interfaces. If the artifact presented for validation does not expose enough information to satisfy the selected profile's requirements, the implementation MUST treat that as a profile-validation failure.

At workflow bootstrap, the issuing Authorization Server MUST establish the workflow subject according to local policy and the selected disclosure profile. The resulting sub value MAY be an ordinary subject identifier, a pairwise identifier, or a workflow-local stable alias. For privacy-sensitive subset-disclosure operation, the Authorization Server SHOULD choose a stable representation that does not reveal a globally useful subject identifier to recipients that are not entitled to learn it.

For same-domain token exchange and Refresh-Exchange, this specification preserves that exact chosen sub representation within the same domain for the lifetime of the workflow unless a later

permitted cross-domain alias transition occurs. Cross-domain re-issuance MAY translate sub only under the semantic- equivalence rule defined later. This document does not define a same-workflow subject-transition mechanism.

The chosen sub representation for a workflow MUST remain consistent with the selected actp disclosure constraints. In particular, sub MUST NOT disclose an actor identity or other actor-chain information that the selected profile is intended to withhold from the relevant recipient class.

In this self-contained JWT/JWS binding, a returned ordinary token is visible to the current actor that receives it and to the next recipient that validates it. Therefore, when a profile returns a visible act, the Authorization Server MUST NOT disclose in that returned token any actor identity that the current actor is not permitted to learn. A future recipient-protected disclosure mechanism or encrypted binding may support stronger recipient-only redisclosure, but that is outside this base specification.

7.2. Actor-Chain Identifier

The acti value:

* MUST be minted once at workflow start by the issuing Authorization Server;
* MUST be generated using a cryptographically secure pseudorandom number generator (CSPRNG) with at least 122 bits of entropy;
* MUST remain unchanged for the lifetime of that workflow instance; and
* MUST NOT be used to signal profile selection.

Implementation note: standard UUID version 4 (UUIDv4), which provides 122 bits of random entropy, is acceptable for acti in this version. Deployments MAY use stronger generation (for example, full 128-bit random values) by local policy.

Profile selection MUST be signaled explicitly using the token request parameter actor_chain_profile and the corresponding token claim actp.

7.3. Target Context Requirements

target_context is the canonical next-hop target value bound into verified profile step proofs and, when used, into hop_ack.

In this base specification, aud identifies the intended recipient service or audience of the issued token. An optional resource value can further identify a narrower protected resource, API surface, or

object within that audience. When no finer-grained targeting inputs are used, the canonical target context is still a JSON object that contains only aud.

The following normative requirements apply to target_context.

For every profile-defined signed, hashed, compared, retained, or communicated use in this specification, target_context MUST be a JSON object.

target_context MUST contain an aud member carrying the verified audience information exactly in the profile-defined canonical representation. If aud is a string, target_context.aud MUST be that same JSON string. If aud is an array of strings, target_context.aud MUST be that exact JSON array, preserving element order.

If no additional target-selection values are used, target_context MUST be the single-member object { "aud": aud }.

A deployment MAY additionally include:

* resource, when the hop is bound to a narrower protected resource or API surface within the audience;
* request_id, when the deployment expects multiple distinct accepted successors under the same prior state and the same nominal target; and
* other local extension members used by same-domain local authorization policy.

This base specification assigns interoperable security semantics only to aud, optional resource, and optional request_id. Cross-domain equivalence or narrowing decisions defined by this document MUST be evaluated using only those members unless a future companion specification defines additional target-context members and their semantics.

When a deployment expects multiple distinct successors under the same prior state and the same nominal target, it MUST include a request-unique discriminator such as request_id inside target_context.

target_context members MUST NOT disclose actor identities or other actor-chain information that the selected actp would withhold from the relevant holder of the artifact.

An Authorization Server that validates, preserves, or audits workflow continuity using target_context MUST retain the exact canonical target_context value for the accepted hop, or enough original request material to reconstruct that exact canonical value later.

Whenever target_context is incorporated into a profile-defined signature or commitment input in this JWT-based version, it MUST be represented as a JSON object and canonicalized exactly once as part of the enclosing JSON Canonicalization Scheme (JCS)-serialized payload object. Equality checks over target_context MUST therefore compare the exact JSON object value after JCS canonicalization. Implementations MUST NOT collapse an audience array to a string, replace an object with a bare aud value, reorder array elements, or otherwise rewrite the verified audience structure before signing or comparing target_context.

7.4. Canonicalization

All profile-defined signed or hashed inputs MUST use a canonical serialization defined by this specification.

In this version of the specification, CanonicalEncode(x) means JCS {{!RFC8785}} applied to the JSON value x.

Hash_halg(x) denotes the raw hash output produced by applying the selected commitment hash algorithm halg to the octet sequence x.

b64url(x) denotes the base64url encoding of the octet sequence x without trailing padding characters, as defined by {{!RFC7515}} Appendix C.

Canonical profile-defined proof payloads MUST be serialized using JCS {{!RFC8785}}.

7.5. Actor Identity Representation

This specification requires a canonical representation for actor identity in profile-defined visible-chain entries and step proofs.

Each canonical actor identifier used by this specification MUST be represented as an ActorID structure containing exactly two members:

* iss: the issuer identifier naming the namespace in which the actor subject value is defined; and
* sub: the subject identifier of the actor within that issuer namespace.

An ActorID is a JSON object with members iss and sub, serialized using JCS {{!RFC8785}} when incorporated into profile-defined signed or hashed inputs.

An ActorID:

* MUST be stable for equality comparison within a workflow instance;
* MUST be bound to the actor identity accepted under local policy for the relevant exchange, proof-verification, or acknowledgment-validation step;
* MUST be compared using exact equality of the pair (iss, sub); and
* SHOULD support pairwise or pseudonymous subject values where deployment policy allows.

When deriving an ActorID from a validated inbound token:

* for the token subject, use { "iss": token.iss, "sub": token.sub };
* for a validated act claim that contains both iss and sub, use those two values directly; and
* for a validated act claim that contains sub but omits iss, use the enclosing token’s iss as the ActorID iss value and the act.sub value as the ActorID sub value.

If no usable act claim is present and a profile needs the presenting actor, that actor MUST be established from actor authentication or other locally trusted inputs outside the scope of this specification and mapped into the same ActorID representation the issuing Authorization Server uses for proof construction.

When actp is present, the act structure used by this specification is an ActorChainNode. An ActorChainNode is an ActorID object plus an OPTIONAL nested member named act whose value is another ActorChainNode representing the immediately prior visible actor. Newly issued profile-defined act structures MUST carry explicit iss and sub in every visible node. Implementations MUST be able to decode and normalize a validated inbound visible chain even when a node omits iss and inherits the enclosing issuer according to the derivation rule above.

The helper functions used throughout this document are therefore:

{::nomarkdown} <sourcecode type="text"> VisibleChain(act) = ordered list of ActorID values obtained by recursively reading nested act from the innermost prior actor to the outermost current actor.

EncodeVisibleChain([A]) = {"iss": A.iss, "sub": A.sub}
EncodeVisibleChain([A,B]) = {"iss": B.iss, "sub": B.sub, "act": {"iss": A.iss, "sub": A.sub}} EncodeVisibleChain([A,B,C]) = {"iss": C.iss, "sub": C.sub, "act": {"iss": B.iss, "sub": B.sub, "act": {"iss": A.iss, "sub": A.sub}}} </sourcecode> {:/nomarkdown}

In examples and formulas, [A,B] denotes the ordered visible chain of ActorID values for actors A and B, while EncodeVisibleChain([A,B]) denotes the nested JSON representation carried in act.

7.6. Artifact Typing

JWT-based artifacts defined by this specification MUST use explicit typ values.

The following JWT typ values are defined:

* act-step-proof+jwt
* act-commitment+jwt
* act-hop-ack+jwt

Verifiers MUST enforce mutually exclusive validation rules based on artifact type and MUST NOT accept one artifact type in place of another. They MUST verify the expected JWT typ, exact ctx value where applicable, and artifact-specific payload structure defined by the relevant binding section of this specification.

7.7. Issued Token Type

Unless another application profile explicitly states otherwise, tokens issued under this specification are access tokens.

Token exchange responses MUST use the RFC 8693 token type fields consistently with the underlying representation and deployment.

7.8. Commitment Hash Algorithms

Proof-bound profiles use a named hash algorithm for construction of actc. Commitment hash algorithm identifiers are values from the IANA Named Information Hash Algorithm Registry {{!RFC6920}} {{IANA.Hash.Algorithms}}.

The following requirements apply:

* halg MUST be a text string naming a hash algorithm from the IANA Named Information Hash Algorithm Registry.
* Implementations supporting verified profiles MUST implement sha-256.
* Implementations SHOULD implement sha-384.
* Every actc object and every verified profile bootstrap context MUST carry an explicit halg value. Verifiers MUST NOT infer or substitute halg when it is absent.
* Verifiers MUST enforce a locally configured allow-list of acceptable commitment hash algorithms and MUST NOT accept algorithm substitution based solely on attacker-controlled inputs.
* Hash algorithms with truncated outputs, including truncated sha-256 variants, MUST NOT be used with this specification.

* Additional registry values MAY be used only if they are permitted by this specification or a future Standards Track update to it, and verifiers MUST reject locally deprecated or disallowed algorithms.
* An Authorization Server MUST NOT initiate a new workflow using a locally deprecated or disallowed algorithm. Whether an already-issued workflow using such an algorithm may continue is a matter of local policy.

7.9. Commitment Function

Proof-bound profiles use actc to bind each accepted hop to the prior accepted state. The commitment hash algorithm is selected once for the workflow by the issuing Authorization Server during bootstrap and remains fixed for the lifetime of that workflow instance.

Each actc value is a signed commitment object whose payload contains:

* ctx: the context string actor-chain-commitment-v1;
* iss: the issuer identifier of the Authorization Server that signs this commitment object;
* acti: the actor-chain identifier;
* actp: the active profile identifier;
* halg: the hash algorithm identifier;
* prev: the prior commitment digest, or the bootstrap initial_chain_seed at workflow start;
* step_hash: b64url(Hash_halg(step_proof_bytes)); and
* curr: b64url(Hash_halg(CanonicalEncode({ctx, iss, acti, actp, halg, prev, step_hash}))).

The curr value MUST be computed over exactly the seven members ctx, iss, acti, actp, halg, prev, and step_hash, excluding curr itself. The resulting digest is then inserted as the transported curr member.

Let prev_digest denote the prior commitment-state digest for the step being processed: at bootstrap it is the initial_chain_seed, and for later steps it is the verified curr value extracted from the inbound actc. For the JWT binding defined in this version, let step_proof_bytes denote the ASCII bytes of the exact compact JWS string submitted as actor_chain_step_proof. Let as_issuer_id denote the issuer identifier that the Authorization Server places into the commitment object’s iss member, typically its issuer value. The commitment hash therefore binds the transmitted step-proof artifact, not merely its decoded payload.

When a profile-defined proof input refers to a prior actc, the value incorporated into the proof input MUST be that prior commitment's verified curr digest string, copied directly from the validated actc payload, not the raw serialized commitment object.

The abstract function used throughout this document is therefore:

{::nomarkdown} <sourcecode type="text"> Commit_AS(as_issuer_id, acti, actp, prev_digest, step_proof_bytes, halg) = AS-signed commitment object over payload { ctx, iss, acti, actp, halg, prev = prev_digest, step_hash = b64url(Hash_halg(step_proof_bytes)), curr = b64url(Hash_halg(CanonicalEncode({ctx, iss, acti, actp, halg, prev, step_hash}))) } </sourcecode> {:/nomarkdown}

The exact wire encoding of the signed commitment object is defined in the JWT binding in Appendix A.

In calls to Commit_AS, the iss input is the issuer identifier of the Authorization Server signing the new commitment object, and acti and actp are the workflow and profile values being preserved for that workflow state.

7.10. Common Cryptographic Operations

The verified profiles use a small number of proof-input templates. This section defines them once so that profile sections can state only their profile-specific substitutions.

Let:

* profile be the active actp value;
* acti be the stable actor-chain identifier;
* prev_state be either the returned base64url initial_chain_seed from bootstrap or the verified prior commitment digest string from actc.curr, as required by the profile;
* visible_actor_chain_for_hop be the exact ordered actor-visible chain for the hop after appending the authenticated current actor;
* workflow_sub be the exact preserved workflow sub string for the hop being extended within the current issuer domain;
* TC_next be the canonical target_context for the next hop, often the object { "aud": aud } but extended when local policy needs finer-grained target binding; and
* [N] denote the canonical ActorID JSON object representation of the authenticated current actor.

Symbols such as TC_B, TC_C, and TC_next denote the canonical target_context for the corresponding next hop.

Proof-bound profiles instantiate the following generic step-proof payload template:

{::nomarkdown} <sourcecode type="json"> Sign_N({ "ctx": ds(profile), "acti": acti, "prev": prev_state, "sub": workflow_sub, "act": EncodeVisibleChain(visible_actor_chain_for_hop), "target_context": TC_next }) </sourcecode> {:/nomarkdown}

The domain-separation string ds is profile-specific:

* actor-chain-verified-full-step-sig-v1 for Verified Full Disclosure;
* actor-chain-verified-subset-step-sig-v1 for Verified Subset Disclosure; and
* actor-chain-verified-actor-only-step-sig-v1 for Verified Actor-Only Disclosure.

These strings remain distinct even though the verified branch step-proof payload members are structurally aligned. The signed step-proof payload does not carry actp or another explicit profile identifier, and the meaning of the act member remains profile-dependent. Distinct domain-separation strings are therefore REQUIRED to bind the proof to the intended verified profile semantics and to prevent cross-profile proof confusion or accidental proof reuse.

In same-domain verified chain-extension hops, the step proof also binds the exact preserved workflow sub string for that hop. This protects against same-domain silent subject substitution without requiring this base specification to cryptographically bind later cross-domain sub aliasing.

The profile-specific meaning of visible_actor_chain_for_hop is:

* for *Verified Full Disclosure*, the full visible chain for the hop after appending the authenticated current actor;
* for *Verified Subset Disclosure*, the exact inbound visible chain verified by the current actor, with that current actor appended; and
* for *Verified Actor-Only Disclosure*, that same exact inbound disclosed visible chain verified by the current actor, with that current actor appended, even though the returned ordinary token later discloses only [N].

For verified subset-disclosure operation, any visible act disclosed to the next recipient in this JWT/JWS binding MUST be derived from the verified visible_actor_chain_for_hop and MUST be an ordered subsequence of it. A singleton current-actor-only form [N] is syntactically representable under subset disclosure, but this

document also defines explicit actor-only profiles so that such policy remains machine-readable. Omission of act is also permitted under the subset profiles. A future recipient-protected disclosure mechanism MAY define stronger redisclosure from Authorization-Server-retained authoritative workflow state without requiring the presenting actor to learn the broader disclosure, but that is outside this base specification.

8. Profile Selection and Workflow Immutability

This specification uses capability discovery plus explicit profile selection, not interactive profile negotiation.

An actor requesting a token under this specification MUST select exactly one actor_chain_profile value for that request. The Authorization Server MUST either issue a token whose actp equals that requested profile identifier or reject the request.

For a given workflow instance identified by acti, actp is immutable. Accordingly, every accepted chain state within that workflow instance carries the same actp. Any token exchange, cross-domain re-issuance, or Refresh-Exchange that would change actp for that workflow instance MUST be rejected. A current actor MUST reject any returned token whose actp differs from the profile it requested or from the preserved profile state already represented by the inbound token.

Profile switching therefore requires starting a new workflow instance with a new acti, not continuing an existing accepted chain state.

9. Common Validation Procedures

This section gives the short validation checklists that the profile sections reuse. Detailed enforcement rules for actor authentication inputs, proof-key binding, intended-recipient handling, and replay or freshness are collected later in "Common Security and Enforcement Requirements".

Implementations can think of validation in three layers:

* *Layer 1: Admission* -- JWT signature, issuer trust, expiry, audience, intended-recipient checks, and any locally established presenting-actor or current-actor continuity checks required for that processing step.
* *Layer 2: Profile authorization* -- interpretation of the visible nested act according to actp and application of local authorization policy using only the visible chain that the profile exposes.

* *Layer 3: Continuity and audit evidence* -- validation of actc, step proofs, acknowledgments, and retained Authorization Server records.

Recipients that only consume an inbound token MAY apply Layer 3 according to local policy. Current actors that extend a verified workflow, and Authorization Servers that accept verified profile token exchange, MUST validate the inbound actc and any required step proof before extending the workflow.

9.1. Recipient Validation of an Inbound Token

Unless a profile states otherwise, a recipient validating an inbound actor-chain token MUST verify:

* token signature;
* issuer trust;
* profile identifier (actp);
* presence and correct format of profile-required structural claims (actc and any profile-required disclosed act according to actp);
* expiry and replay requirements;
* intended-recipient requirements; and
* any profile-specific disclosed-chain checks.

If a disclosed act is present, a recipient MUST decode the visible chain as VisibleChain(act) and apply the profile-specific checks on that visible chain. If the selected profile requires disclosed current-actor continuity and the recipient establishes a presenting-actor identity for the inbound hop under local policy, the recipient MUST also verify that the outermost act identifies that same presenting actor. If the selected profile requires inline act disclosure, omission of act, or presence of an act that violates the profile’s required disclosure rules, MUST be treated as a profile-validation failure.

A recipient MUST use only the actor-chain information actually disclosed in the inbound artifact for authorization. It MUST treat undisclosed prior actors, and an undisclosed current actor, as unavailable from that artifact.

9.2. Authorization Server Validation of Token Exchange

Unless a profile states otherwise, an Authorization Server validating an actor-chain token exchange MUST verify:

* the inbound token signature and issuer trust;
* the authenticated identity of the current actor;
* intended-recipient semantics for the inbound token;

* the selected profile identifier and any profile-specific step-proof requirements;
* the preserved acti and sub continuity expected for the workflow; and
* any profile-specific disclosed-chain checks on any inbound act.

If the inbound token discloses visible chain state, the Authorization Server MUST decode VisibleChain(act_in) from the inbound token exactly as verified for the current actor and MUST perform only append-only operations on that disclosed visible chain fragment. If the selected profile requires disclosed current-actor continuity, the Authorization Server MUST additionally verify that the outermost act of the inbound token identifies the same actor that authenticated as the current actor and is presenting that inbound hop for exchange. If the selected profile requires inline act disclosure, omission of act, or presence of an act that violates the profile's required disclosure rules, MUST be treated as a profile-validation failure.

This specification does not define validation or handling rules for may_act. Any effect of may_act is determined by RFC 8693, by the specification governing the artifact that carries it, or by local policy. Such use MUST NOT cause an artifact issued under this specification, or any related protocol-visible outcome, to disclose actor-chain information that the selected profile would withhold.

A token-exchange request under this specification MAY additionally carry the RFC 8693 actor_token and actor_token_type parameters. This specification does not define whether or how actor_token contributes actor-related policy input. Any such effect is determined by RFC 8693, by the specification governing that artifact, or by local policy. Such use MUST NOT expand disclosure beyond the selected profile and Authorization Server policy, and MUST NOT alter required acti, actp, or sub continuity.

If neither may_act nor actor_token contributes actor-related policy input, the Authorization Server applies local policy. For this specification, local policy can include trust relationships, authenticated client identity, permitted target contexts, chain-depth limits, profile support, and business or risk controls.

9.3. Current-Actor Validation of a Returned Token

Unless a profile states otherwise, a current actor validating a returned token MUST verify:

* token signature and issuer trust;
* that the returned actp equals the requested profile identifier and any preserved active profile state;

* that the returned acti equals the actor-chain identifier already represented by the inbound token or bootstrap state;
* that the returned sub equals the expected preserved workflow-subject value;
* expiry and replay requirements; and
* any profile-specific disclosed-chain or commitment checks.

If a returned token discloses act, the current actor MUST verify that the disclosed act conforms to the selected profile and Authorization Server policy for that hop. If the selected profile requires inline act disclosure, omission of act, or presence of an act that violates the profile's required disclosure rules, MUST be treated as a profile-validation failure.

For readable full-disclosure profiles, the current actor MUST verify that the returned visible chain equals the full verified chain for that hop. For subset-disclosure profiles, if the returned token discloses act, the current actor MUST verify that the returned visible chain is an ordered subsequence of the exact verified actor-visible chain that the actor signed or otherwise caused to be asserted for that hop. For actor-only profiles, the current actor MUST verify that the returned visible act consists only of the outermost current actor and that this actor is the current actor that requested or caused the hop.

10. Profiles

The profile selection table appears earlier in "Scope and Model". The sections below present the ordinary chain-extending profile flows first. Each profile depends on the common recipient, Authorization Server, and returned-token validation procedures defined earlier, plus the later common security and enforcement rules. Special preserve-state exchanges, metadata, and deeper enforcement details appear later so they do not interrupt the main profile story.

11. Declared Full Disclosure Profile

11.1. Profile Identifier

The profile identifier string for this profile is declared-full. It is used as the actor_chain_profile token request parameter value and as the actp token claim value.

11.2. Objective

The Declared Full Disclosure profile extends token exchange by carrying the full visible nested act chain in every ordinary token and by requiring chain-continuity validation by both the current actor and the issuing Authorization Server at each hop.

11.3. Security Model

This profile provides hop-by-hop visible-chain integrity based on issuer-asserted chain state and continuity checks.

This profile assumes that an actor does not collude with its home Authorization Server.

11.4. Bootstrap

At workflow start, actor A MUST request a token from AS1 with:

* grant_type=client_credentials;
* actor_chain_profile=declared-full; and
* the requested OAuth targeting parameters (audience, resource, or both) sufficient to identify B as the initial target context.

If AS1 accepts the request, AS1 MUST establish the workflow subject according to local policy before issuing T_A. At bootstrap under these base profiles, AS1 MUST choose a stable workflow-subject representation for sub that is appropriate for the selected disclosure policy. AS1 MUST then issue T_A containing at least:

* actp=declared-full
* sub
* act=EncodeVisibleChain([A])
* acti
* jti
* aud=B
* exp

11.5. Hop Processing

When A calls B, A MUST present T_A to B.

B MUST perform recipient validation as described in "Recipient Validation of an Inbound Token".

B MUST decode VisibleChain(T_A.act) and verify that its last actor is A.

If that continuity check fails, B MUST reject the request.

11.6. Token Exchange

To call C, B MUST submit to AS1 at least:

* grant_type=urn:ietf:params:oauth:grant-type:token-exchange;
* actor_chain_profile=declared-full;
* T_A as the RFC 8693 subject_token;
* subject_token_type=urn:ietf:params:oauth:token-type:access_token; and
* the requested OAuth targeting parameters (audience, resource, or both as needed by local policy) sufficient to identify C as the next target context.

B MAY additionally submit RFC 8693 actor_token and actor_token_type parameters subject to the common validation rules in "Authorization Server Validation of Token Exchange".

AS1 MUST perform token-exchange validation as described in "Authorization Server Validation of Token Exchange".

AS1 MUST read the prior visible chain from T_A.act, append B, and issue T_B containing at least:

* actp=declared-full
* sub
* act=EncodeVisibleChain([A,B])
* acti
* jti
* aud=C
* exp

11.7. Returned Token Validation

Upon receipt of T_B, B MUST perform current-actor returned-token validation as described in "Current-Actor Validation of a Returned Token".

B MUST verify that VisibleChain(T_B.act) is exactly the previously verified chain from T_A with B appended.

If that append-only check fails, B MUST reject T_B.

11.8. Next-Hop Validation

Upon receipt of the final B-token, C MUST perform recipient validation as described in "Recipient Validation of an Inbound Token".

C MUST decode VisibleChain(T_B.act) and use that visible chain for authorization decisions.

11.9. Security Result

Under the non-collusion assumption, prior actors MUST NOT be silently inserted, removed, reordered, or altered during token exchange.

12. Declared Subset Disclosure Profile

12.1. Profile Identifier

The profile identifier string for this profile is declared-subset. It is used as the actor_chain_profile token request parameter value and as the actp token claim value.

12.2. Objective

This profile inherits the Declared Full Disclosure profile and changes the visible chain carried across hops: the issuing Authorization Server MAY carry and disclose only a recipient-specific ordered subset of the asserted chain state for the hop, or MAY omit act entirely. Although a singleton current-actor-only visible chain is syntactically representable here, deployments that require actor-only policy to be explicit and machine-readable SHOULD use the Actor-Only profile instead.

12.3. Inheritance and Security Model

Except as modified below, all requirements of the Declared Full Disclosure profile apply.

When this profile discloses act, the visible chain seen by a recipient, obtained from VisibleChain(act), MUST be an ordered subsequence of the asserted chain state for that hop.

A recipient MUST treat undisclosed prior actors as unavailable and MUST NOT infer adjacency, absence, or exact chain length from the disclosed subset alone. If an inbound token under this profile omits act, or discloses act without the current actor, this specification provides no inline current-actor disclosure for that hop. Any additional authorization inputs are determined by local policy and are outside the scope of this specification.

This profile relies on the issuing Authorization Server for continuity of the asserted chain state and for disclosure policy. The Authorization Server MAY retain authoritative workflow chain state richer than the disclosed subset for audit, forensics, legal review, and branch reconstruction. In this base JWT/JWS binding, however, the returned token is visible to the current actor and to the next recipient, so the disclosed act in that token MUST be acceptable for both.

Cross-domain re-issuance preserves only the visible asserted state carried in the inbound token unless a trusted companion mechanism explicitly transfers additional hidden state. This profile does not provide the step-proof-based accountability or cumulative commitment state of the verified profiles.

12.4. Modified Bootstrap and Issuance

At bootstrap, the initial actor MUST request a token with at least:

* grant_type=client_credentials;
* actor_chain_profile=declared-subset; and
* the requested OAuth targeting parameters (audience, resource, or both) sufficient to identify the initial target context.

At bootstrap and at each later exchange, wherever the Declared Full Disclosure profile would issue a token containing a full visible act chain, this profile MUST instead either issue a recipient-specific disclosed visible act chain for the intended recipient or omit act entirely according to local policy. When disclosed, the chain MAY be the singleton current actor if that is what policy permits.

12.5. Modified Hop Processing and Validation

Where the Declared Full Disclosure profile requires presentation or validation of a full visible chain, this profile instead requires presentation and validation of any disclosed subset carried in act, if present.

12.6. Modified Token Exchange

For this profile, the current actor MUST submit at least:

* grant_type=urn:ietf:params:oauth:grant-type:token-exchange;
* actor_chain_profile=declared-subset;
* the inbound token as the RFC 8693 subject_token;
* subject_token_type=urn:ietf:params:oauth:token-type:access_token; and
* the requested OAuth targeting parameters (audience, resource, or both as needed by local policy) sufficient to identify the next target context.

The current actor MAY additionally submit RFC 8693 actor_token and actor_token_type parameters subject to the common validation rules in "Authorization Server Validation of Token Exchange".

For this profile, the issuing Authorization Server MUST derive the next-hop asserted chain state from accepted workflow state for the inbound hop, append the current actor to that accepted chain state, and then derive any disclosed visible chain for the returned token by selecting an ordered subsequence of the resulting asserted chain state or by omitting act entirely. It MUST NOT insert, reorder, or alter actor identities in any disclosed act.

Because the returned token is visible to the current actor in this base binding, the Authorization Server MUST NOT disclose in that returned token any actor identity that the current actor is not permitted to learn.

If an inbound token under this profile discloses act, the current recipient MUST validate the disclosed subset according to this profile. If a returned token under this profile discloses act, the current actor MUST validate that disclosed subset according to this profile.

Unlike the Declared Full Disclosure profile, the current actor and downstream recipient do not independently validate the hidden undisclosed portion of the prior chain. They validate only the disclosed subset they receive.

12.7. Next-Hop Authorization

A recipient MAY use any disclosed visible chain for authorization decisions.

A recipient MUST use only the disclosed visible chain, if any, for actor-chain authorization and MUST treat undisclosed actors as unavailable. If the token omits act, or discloses act without the current actor, this specification provides no inline current-actor disclosure for that hop. Any additional authorization inputs are determined by local policy and are outside the scope of this specification.

12.8. Security Result

Under the non-collusion assumption, silent insertion, removal, reordering, or alteration of the disclosed chain seen by a recipient is prevented with respect to what the issuing Authorization Server asserted for that recipient.

This profile does not by itself prevent confused-deputy behavior.

13. Declared Actor-Only Disclosure Profile

13.1. Profile Identifier

The profile identifier string for this profile is declared-actor-only. It is used as the actor_chain_profile token request parameter value and as the actp token claim value.

13.2. Objective

This profile inherits the Declared Subset Disclosure profile and fixes the ordinary-token visible act disclosure to the singleton outermost current actor only. Inline act disclosure is therefore mandatory for every ordinary token under this profile. It is intended for workflows in which each hop authorizes only on its immediate upstream actor while the Authorization Server retains richer workflow state for later audit, forensics, or legal review.

13.3. Inheritance and Security Model

Except as modified below, all requirements of the Declared Subset Disclosure profile apply.

The Authorization Server MAY retain authoritative workflow chain state richer than the singleton visible act disclosed in ordinary tokens, but every ordinary token issued under this profile MUST expose only the current actor for that hop. Recipients MUST authorize only on that visible current actor and local policy.

13.4. Modified Bootstrap and Issuance

At bootstrap, the initial actor MUST request a token with at least:

* grant_type=client_credentials;
* actor_chain_profile=declared-actor-only; and
* the requested OAuth targeting parameters (audience, resource, or both) sufficient to identify the initial target context.

At bootstrap and at each later exchange, wherever the Declared Subset Disclosure profile would issue a token containing a disclosed visible act chain, this profile MUST instead issue a token whose visible act contains exactly one ActorChainNode identifying the actor represented by that ordinary token.

13.5. Modified Hop Processing and Validation

Where the Declared Subset Disclosure profile requires presentation or validation of a disclosed visible chain, this profile instead requires validation that the visible act contains only the current actor for this hop.

13.6. Modified Token Exchange

For this profile, the current actor MUST submit at least:

* grant_type=urn:ietf:params:oauth:grant-type:token-exchange;
* actor_chain_profile=declared-actor-only;
* the inbound token as the RFC 8693 subject_token;
* subject_token_type=urn:ietf:params:oauth:token-type:access_token; and
* the requested OAuth targeting parameters (audience, resource, or both as needed by local policy) sufficient to identify the next target context.

For this profile, the issuing Authorization Server MUST validate the inbound visible act as current-actor-only, preserve and extend the accepted workflow state for the authenticated current actor according to local policy, and issue the returned token with a visible act containing exactly one ActorChainNode identifying the actor represented by that returned token. The Authorization Server MUST NOT disclose prior actors inline in ordinary tokens under this profile.

13.7. Next-Hop Authorization

A recipient MUST authorize only on the visible outermost current actor and local policy.

13.8. Security Result

Under the non-collusion assumption, silent alteration of the visible current actor is prevented with respect to what the issuing Authorization Server asserted for that recipient. Prior actors remain available only through Authorization Server records or other out-of-band evidence.

14. Common Processing for the Verified Branch

This section defines the bootstrap, proof, commitment, token-exchange, and returned-token rules shared by the three verified profiles. In this branch, ordinary tokens still carry the hop-to-hop token state, but each chain- extending hop is also backed by an actor-signed step proof and cumulative commitment state:

* Verified Full Disclosure;
* Verified Subset Disclosure; and
* Verified Actor-Only Disclosure.

The profile sections that follow define only the profile-specific meaning of the actor-visible chain for the hop, the ordinary-token disclosure policy, and the corresponding validation and authorization rules.

14.1. Common Parameters

Each verified profile supplies the following profile-specific parameters to the common processing below.

Profile	actp value	Step-proof domain-separation string	Proof-bound actor-visible chain for hop	Visible act in ordinary tokens
Verified Full Disclosure	verified-full	actor-chain-verified-full-step-sig-v1	Full visible chain for the hop after appending the current actor	Full verified visible chain
Verified Subset Disclosure	verified-subset	actor-chain-verified-subset-step-sig-v1	Exact inbound disclosed visible chain verified by the current actor, with that actor appended	Ordered subsequence of the verified actor-visible chain, or omitted act
Verified Actor-Only Disclosure	verified-actor-only	actor-chain-verified-actor-only-step-sig-v1	Exact inbound disclosed visible chain verified by the current actor, with that actor appended	Singleton current actor only

Table 2

The profile identifier, step-proof domain-separation string, and the profile-specific interpretation of act therefore remain aligned across the verified branch.

14.2. Common Bootstrap Context Request

At workflow start, the initial actor MUST request a bootstrap context from the Authorization Server's actor_chain_bootstrap_endpoint using at least:

* grant_type=urn:ietf:params:oauth:grant-type:actor-chain-bootstrap;
* the selected verified profile actor_chain_profile; and
* the requested OAuth targeting parameters (audience, resource, or both) sufficient to identify the initial target context.

The Authorization Server MUST authenticate the initial actor, bind that actor to its ActorID representation, select a commitment hash algorithm, mint a fresh acti, choose a stable workflow-subject representation for sub, derive the bootstrap target context from the requested targeting parameters, and return a bootstrap response containing at least:

* actor_chain_bootstrap_context, an opaque bootstrap handle;
* acti;
* sub;
* halg;
* target_context, the exact canonical bootstrap-bound target context against which the initial step proof will be validated; and
* initial_chain_seed, a base64url value to be used as prev_state for the initial verified step proof.

The initial_chain_seed MUST be generated using a CSPRNG with at least 128 bits of entropy, MUST be unique per workflow instance, and MUST NOT be derived solely from acti or other predictable inputs.

The bootstrap context MUST be integrity-protected by the Authorization Server, bound to the authenticated initial actor, the selected verified profile, the chosen acti, sub, halg, and the bootstrap target context, and be short-lived. It authorizes issuance of initial verified ordinary tokens for that workflow instance while it remains valid.

For a given canonical chosen initial target_context, the Authorization Server MUST treat repeated redemption of the same bootstrap handle as an idempotent retry and MUST return the previously accepted initial state, or an equivalent token representing that same accepted initial state. It MUST NOT issue a second distinct accepted initial state for that same canonical chosen initial target_context.

The Authorization Server MAY accept additional redemptions of the same bootstrap handle for distinct canonical chosen initial target_context values, thereby minting multiple accepted initial successors that share the same acti and initial_chain_seed. If a deployment expects multiple distinct initial successors under the same nominal target, the chosen initial target_context MUST include a unique request_id.

14.3. Common Initial Actor Step Proof and Bootstrap Issuance

After receiving the bootstrap response, the initial actor A MUST choose the initial target context to be used for the first issued token. That chosen initial target context MUST be identical to, or a locally authorized narrowing of, the canonical bootstrap-bound target_context returned in the bootstrap response. Because there are no prior actors at workflow start, the profile-defined actor-visible chain for that initial hop is [A] for all verified profiles. A MUST then sign an actor_chain_step_proof over that initial chain, the exact preserved workflow sub string returned in the bootstrap response, and the chosen initial target context, and redeem the bootstrap context at the token endpoint using at least:

* grant_type=client_credentials;
* the selected verified profile actor_chain_profile;
* actor_chain_bootstrap_context;
* actor_chain_step_proof; and
* the requested OAuth targeting parameters (audience, resource, or both) sufficient to identify that chosen initial target context.

The Authorization Server MUST verify the bootstrap context and verify that the authenticated actor redeeming it is the same actor to which the bootstrap context was issued. It MUST also verify that the requested profile matches the bound bootstrap profile, that the chosen target context is identical to or narrower than the bootstrap-bound target context according to local policy, that the submitted step proof ctx equals the active profile’s domain- separation string, that the submitted step proof sub equals the bound workflow sub string, and that the submitted step proof is otherwise valid. It then creates the initial actc and issues an ordinary token containing at least iss, actp, acti, sub, jti, aud, exp, and actc, and any profile-governed disclosed act.

Because the initial verified hop has no prior visible actors, the exact verified actor-visible chain for that hop is [A]. The returned ordinary token MUST disclose that chain as act=EncodeVisibleChain([A]) for the Verified Full Disclosure and Verified Actor-Only Disclosure profiles. For Verified Subset Disclosure, the Authorization Server MAY either disclose act=EncodeVisibleChain([A]) or omit act entirely according to local policy.

14.4. Common Hop Processing

When a current actor N receives an inbound verified profile token, it MUST:

* validate the inbound token and any required actc;
* determine, under local policy, any presenting-actor identity for the inbound hop needed by the selected profile;
* derive the profile-defined actor-visible chain for the new hop;
* sign actor_chain_step_proof over that visible chain, the preserved workflow sub string for the hop, and the next target context; and
* submit the inbound token plus the step proof in a token exchange request to its home Authorization Server.

14.5. Common Token Exchange

For a chain-extending verified profile token exchange, the current actor MUST submit at least:

* grant_type=urn:ietf:params:oauth:grant-type:token-exchange;
* the selected verified profile actor_chain_profile;
* the inbound token as the RFC 8693 subject_token;
* subject_token_type=urn:ietf:params:oauth:token-type:access_token;
* actor_chain_step_proof; and
* the requested OAuth targeting parameters sufficient to identify the next target context.

The current actor MAY additionally submit RFC 8693 actor_token and actor_token_type parameters subject to the common validation rules in "Authorization Server Validation of Token Exchange".

The Authorization Server MUST validate the inbound token, validate the step proof for the active profile, MUST verify that the step proof ctx equals the active profile’s domain-separation string, MUST verify that the step proof sub equals the preserved workflow sub string for the hop being extended, verify append-only processing of the profile-defined actor-visible chain, update actc, and return an ordinary token whose act representation matches the profile’s disclosure rules for the next recipient. Replay, idempotency, and multiple-successor handling for submitted step proofs are defined later in "Replay and Freshness".

14.6. Common Returned-Token Validation

When validating a returned verified profile token, the current actor MUST perform the common returned-token validation described earlier and MUST also verify:

* that actc is present and valid;
* that the returned token preserves acti, actp, and sub as required;
* that the embedded actc payload preserves the expected acti, actp, and halg values for the active workflow state;
* that actc.prev equals the previously verified prior commitment digest, or the initial_chain_seed for the initial verified hop;
* that actc.step_hash equals b64url(Hash_halg(step_proof_bytes)) computed over the exact compact JWS string submitted as actor_chain_step_proof for that hop; and
* any profile-specific disclosed-chain checks for the active disclosure mode.

Because actc is cumulative commitment state carried inline in verified profile ordinary tokens, every later actor and every Authorization Server that extends the verified workflow relies on that inline actc value. Online validation of an inbound or returned actc proves issuer-signed commitment continuity and internal commitment consistency. It does not by itself prove the semantic meaning of step_hash against the underlying step proof unless the exact step proof bytes and related verification material are also retained or discoverable for later audit. Deployments MAY vary on whether every terminal recipient performs synchronous actc validation at admission time, but chain extension MUST NOT proceed without successful validation of the inbound actc.

15. Verified Full Disclosure Profile

15.1. Profile Identifier

The profile identifier string for this profile is verified-full. It is used as the actor_chain_profile token request parameter value and as the actp token claim value.

15.2. Objective

The Verified Full Disclosure profile is the full-disclosure verified case. It preserves a visible nested act chain in every ordinary token for every actor and downstream recipient while adding per-hop actor-signed step proofs and cumulative commitment state.

15.3. Security Model

Bootstrap, Common Token Exchange, and Common Returned-Token Validation follow the Common Processing for the Verified Branch section. This profile inherits all requirements of that common verified processing and specializes them to the full-disclosure visible case.

This profile preserves visible chain-based authorization and provides stronger accountability and non-repudiation than the Declared Full Disclosure profile.

This profile does not guarantee inline prevention of every invalid token that could be issued by a colluding actor and its home Authorization Server.

The evidentiary value of this profile depends on retention or discoverability of step proofs, exchange records, and associated verification material.

15.4. Profile-Specific Hop Construction and Validation

For a current actor N, let chain_in be the full visible chain verified from the inbound token. If N establishes a presenting actor for the inbound hop under local policy, N MUST verify that the last actor in chain_in is that same presenting actor.

The exact verified actor-visible chain for the hop is the full visible append-only chain:

{::nomarkdown} <sourcecode type="text"> visible_hop_N = chain_in + [N] </sourcecode> {:/nomarkdown}

The Authorization Server MUST issue EncodeVisibleChain(visible_hop_N) as the visible act in the returned token.

A recipient MUST use the full visible chain for authorization decisions.

15.5. Attack Handling

A claim that actor V participated in the chain MUST fail unless a valid step proof for V can be produced and verified against the corresponding prior commitment state and acti.

If an actor is omitted from a later visible chain, that omitted actor MAY prove prior participation by presenting:

* an earlier token showing the prior chain state; and
* the corresponding commitment state and verifiable step proof, or an immutable Authorization Server exchange record.

A denial of participation by actor X MUST fail if a valid step proof for X is available and verifies.

15.6. Security Result

This profile preserves visible chain-based authorization while making tampering materially easier to detect, prove, and audit.

16. Verified Subset Disclosure Profile

16.1. Profile Identifier

The profile identifier string for this profile is verified-subset. It is used as the actor_chain_profile token request parameter value and as the actp token claim value.

16.2. Objective

This profile is the visible subset-disclosure verified case. The issuing Authorization Server MAY disclose only a recipient-specific ordered subset of the verified actor-visible chain, while the current actor signs the exact actor-visible chain that it was allowed to verify and extend for the hop. This preserves stronger continuity and later verifiability without requiring full inline disclosure at every hop.

16.3. Security Model

Bootstrap, Common Token Exchange, and Common Returned-Token Validation follow the Common Processing for the Verified Branch section. This profile inherits all requirements of that common verified processing.

When this profile discloses act, the visible chain seen by a recipient, obtained from VisibleChain(act), MUST be an ordered subsequence of the verified actor-visible chain for that hop.

Step proofs and actc values MUST be computed over the exact actor-visible chain for the hop, not over a hidden canonical full chain that the current actor was not permitted to see.

A recipient MUST treat undisclosed prior actors as unavailable and MUST NOT infer adjacency, absence, exact chain length, or hidden prefixes from the disclosed subset alone.

The Authorization Server MAY retain authoritative workflow chain state richer than the disclosed subset for audit, forensics, legal review, and branch reconstruction. In this base JWT/JWS binding, however, the returned token is visible to the current actor and to the next recipient, so the returned visible act MUST be acceptable for both. Accordingly, a later privileged recipient can learn more than an earlier intermediary only at a hop where every holder of the returned token is permitted to learn that broader disclosure, or by using a future recipient-protected disclosure mechanism outside this base specification.

16.4. Profile-Specific Hop Construction and Validation

For a current actor N, let chain_in be the exact disclosed inbound visible chain that N verified from the inbound token, or the empty chain if the inbound token disclosed no act. If chain_in is non-empty and N establishes a presenting actor for the inbound hop under local policy, N MUST verify that its last actor is that same presenting actor.

The exact verified actor-visible chain for the hop is:

{::nomarkdown} <sourcecode type="text"> visible_hop_N = chain_in + [N] </sourcecode> {:/nomarkdown}

The current actor N MUST append only itself and MUST NOT insert, delete, or reorder prior actors within chain_in.

The Authorization Server MUST verify that the submitted visible chain equals the exact inbound disclosed chain previously verified by N, with N appended. An omitted inbound act corresponds to the empty chain_in value.

Any visible act disclosed to the next recipient in this base binding MUST be derived from the exact verified actor-visible chain for that hop and MUST be an ordered subsequence of it. The singleton [N] is the actor-only disclosure shape, and omission of act is also permitted under this profile.

When validating a returned token, the current actor N MUST additionally verify, if the returned token discloses act, that the disclosed chain, obtained from VisibleChain(act), is an ordered subsequence of the exact verified actor-visible chain that N signed for that hop.

A recipient MAY use the verified disclosed visible chain for authorization decisions, but MUST use only the disclosed subset and MUST treat undisclosed prior actors as unavailable. If the token omits act, or discloses act without the current actor, this specification provides no inline current-actor disclosure for that hop. Any additional authorization inputs are determined by local policy and are outside the scope of this specification.

16.5. Attack Handling

Different recipients MAY receive different valid disclosed subsets derived from the same verified actor-visible chain according to local disclosure policy. That alone does not constitute an integrity failure.

A malicious or compromised Authorization Server could still attempt to issue a disclosed subset inconsistent with the verified actor-visible chain. Such an inconsistency MUST fail if the retained step proof for that hop or an immutable Authorization Server exchange record is later checked.

An actor omitted from a disclosed chain MAY still prove prior participation by presenting the corresponding step proof or immutable Authorization Server exchange record for the verified actor-visible chain for the relevant hop.

When this profile omits act, or discloses only a narrow subset, later reconstruction of the full accepted chain from step proofs alone is not guaranteed. Such reconstruction can require retained Authorization Server records or other authoritative workflow evidence. In those cases, the current actor’s signature provides non-repudiation only over the exact actor-visible chain for that hop and the linked prior commitment digest, not over hidden prefix semantics that the actor was not permitted to verify.

16.6. Security Result

This profile preserves Authorization-Server-validated continuity, cumulative commitment state, and recipient-specific limited visible authorization while avoiding disclosure of hidden prior actors beyond the subset visible in the returned ordinary token.

17. Verified Actor-Only Disclosure Profile

17.1. Profile Identifier

The profile identifier string for this profile is verified-actor-only. It is used as the actor_chain_profile token request parameter value and as the actp token claim value.

17.2. Objective

This profile inherits the common verified processing and keeps the actor-signed step proof and cumulative actc continuity state, while restricting ordinary tokens to the singleton outermost current actor only. It is intended for end-to-end execution pipelines in which each hop authorizes only on the immediate upstream actor even though the Authorization Server retains richer authoritative workflow state for later forensics, legal audit, or branch reconstruction.

17.3. Security Model

Except as modified below, all requirements of the common verified processing section apply.

The current actor signs the exact verified actor-visible chain for the hop, but the ordinary token returned for the next hop MUST disclose only the current actor. Recipients therefore authorize only on the visible current actor plus local policy, while actc and retained step proofs preserve stronger continuity evidence for later review.

17.4. Profile-Specific Hop Construction and Validation

For each hop under this profile, the current actor MUST construct the profile-defined verified actor-visible chain as the exact inbound disclosed visible chain that the actor verified, with that actor appended. The step proof MUST carry that exact actor-visible chain in its act member.

Where Verified Subset Disclosure permits any ordered subsequence of the verified actor-visible chain to appear in the returned ordinary token, this profile instead requires the issuing Authorization Server

to return a visible act containing exactly one ActorChainNode identifying the actor represented by the returned token. The returned token MUST still carry the updated cumulative actc state.

Upon receipt of the returned token, the current actor MUST verify that the returned visible act consists only of the current actor and that the returned actc matches the accepted successor state for the verified hop.

Upon receipt of the token, the next recipient MUST perform recipient validation and authorize only on the visible current actor and local policy. If the recipient establishes a presenting actor for the inbound token under local policy, it MUST also confirm that the visible current actor matches that same presenting actor.

17.5. Attack Handling

If a returned ordinary token under this profile discloses any prior actor inline, the current actor MUST reject it. If a recipient under this profile attempts to infer hidden prior actors from omission, identifier structure, or actc alone, that behavior is out of profile and MUST NOT be treated as a conforming authorization decision.

17.6. Security Result

Under the non-collusion assumption and when step proofs and actc are retained as required by local policy, silent alteration of the current actor in ordinary tokens is prevented and stronger continuity evidence remains available for later forensic or legal review, while inline disclosure remains restricted to the current actor only.

18. Special Preserve-State Exchanges

These sections define the two token-exchange cases that preserve previously accepted chain state rather than appending a new actor. They are easiest to read after the ordinary profile flows.

A token-exchange request under this specification MUST NOT set both actor_chain_cross_domain=true and actor_chain_refresh=true. The Authorization Server MUST reject any request that sets both.

18.1. Cross-Domain Re-Issuance

18.1.1. Request Format

If the next hop does not trust the current Authorization Server directly, the current actor MUST perform a second token exchange at the next domain’s Authorization Server.

A cross-domain re-issuance request MUST include:

* grant_type=urn:ietf:params:oauth:grant-type:token-exchange;
* actor_chain_cross_domain=true;
* actor_chain_profile set to the active profile identifier carried by the inbound token;
* the current inbound actor-chain token as the RFC 8693 subject_token;
* subject_token_type=urn:ietf:params:oauth:token-type:access_token; and
* any requested OAuth targeting parameters (audience, resource, or both) for the local target context to be minted by the re-issuing Authorization Server.

The re-issuing Authorization Server MUST ensure that any locally minted target context is semantically equivalent to, or narrower than, the target context authorized by the inbound token according to local trust policy and audience mapping rules. When the earlier chain-extending hop bound a target_context richer than plain aud, the re-issuing Authorization Server MUST evaluate that equivalence or narrowing against the exact retained or otherwise recoverable prior target_context, not against aud alone. It MUST NOT issue a local token whose target context is broader than, or semantically unrelated to, the target context authorized by the inbound token.

A cross-domain re-issuance request MUST NOT append the chain and MUST NOT submit actor_chain_step_proof, because this exchange preserves rather than extends the accepted chain state. The actor_chain_cross_domain parameter is the explicit wire signal that the request is for preservation and local re-issuance rather than ordinary same-domain chain extension.

18.1.2. Preservation Rules

The cross-domain Authorization Server MUST:

* validate the inbound token signature and issuer trust according to local policy;
* validate the selected actor-chain profile;
* validate the preserved visible-chain structure;
* preserve actp;
* preserve acti;
* preserve actc, if present, exactly as verified;
* continue to represent the same current actor; and
* NOT append the next recipient.

Visible act handling during cross-domain re-issuance is profile-specific:

* for the Full Disclosure profiles and the Actor-Only profiles, the re-issuing Authorization Server MUST preserve the visible act exactly, except that if an inbound visible node omitted iss, the re-issuing Authorization Server MAY materialize that iss explicitly only to preserve the same ActorID semantics;
* for the Subset Disclosure profiles, if the inbound token discloses act, the re-issuing Authorization Server MAY preserve that visible act exactly, disclose any ordered subsequence of that inbound visible chain, or omit act entirely according to local policy; it MUST NOT introduce any actor not present in the inbound visible chain, reorder actors, or use hidden retained state to broaden disclosure; when a returned visible node would otherwise rely on inherited issuer context, the re-issuing Authorization Server MAY materialize the corresponding iss explicitly only to preserve the same ActorID semantics; and
* if the inbound token omits act, the re-issuing Authorization Server MUST NOT synthesize a visible act from hidden retained state.

18.1.3. Subject Handling

For top-level sub, the re-issuing Authorization Server SHOULD preserve the exact inbound value when doing so preserves the same underlying subject semantics and does not broaden disclosure. If exact preservation would change subject semantics under the new issuer namespace or would disclose more than the inbound token disclosed, the re-issuing Authorization Server MAY translate sub into a local alias that denotes the same underlying subject and MUST retain an audit binding between the old and new subject representations. If the re-issuing Authorization Server cannot establish same-subject semantic continuity without broader disclosure, it MUST reject the request. Once such a local alias is accepted in cross-domain re-issuance, that returned sub value becomes the preserved workflow-subject representation for later same-domain validation within the new issuer domain.

This specification cryptographically binds same-domain workflow-subject continuity through verified step proofs. It does not cryptographically bind cross-domain sub alias continuity in preserved actc; cross-domain subject alias continuity therefore remains a matter of Authorization-Server policy and retained audit evidence in this version.

Future companion specifications MAY define privacy-preserving Authorization Server-to-Authorization Server transfer of additional hidden workflow state. Such mechanisms are outside this base specification. This document's cross- domain rules preserve only the visible state allowed by policy together with any preserved actc state defined here.

18.1.4. ActorID Namespace Handling

Each visible act entry uses ActorID semantics over (iss, sub). If an inbound act node omitted iss, the re-issuing Authorization Server MUST preserve the same ActorID semantics by emitting an explicit iss equal to the inbound token's issuer together with the same actor sub, rather than relying on the new local token issuer as an implicit namespace. Internal canonicalization for proof, comparison, and ordered-subsequence evaluation in this document therefore always uses fully materialized ActorID pairs.

The cross-domain Authorization Server MAY mint a new local jti, apply a new local expiry, change token format or envelope, and add local trust or policy claims. If cross-domain re-issuance narrows or locally rewrites the target context, retained step proofs and preserved actc continue to reflect the target context that was bound during the original chain-extending hop, not the narrower or rewritten token audience issued by the re-issuing Authorization Server.

A recipient or current actor in the new domain that trusts the re-issuing Authorization Server MAY rely on that enclosing token signature as attestation that any preserved foreign actc was validated and carried forward unchanged. Such a recipient need not independently validate a foreign Authorization Server's JWS signature on the preserved actc unless local policy or audit requires it.

18.1.5. Returned-Token Validation

When validating a token returned by cross-domain re-issuance, the current actor does not recompute a new commitment object from a new step proof. Instead, it MUST verify the token signature and MUST verify that preserved chain-state fields, including actp, acti, and actc, preserve the same accepted chain state as the inbound token except where this specification explicitly permits cross-domain re-issuance changes such as local sub aliasing under the semantic-equivalence rule, local jti, local exp, token format or envelope, approved local trust and policy claims, or explicit iss materialization in act solely to preserve the same ActorID semantics when an inbound node had omitted iss. For Full Disclosure and Actor-Only profiles, any returned visible act MUST preserve the inbound

visible act exactly except for such permitted iss materialization. For Subset Disclosure profiles, any returned visible act, if present, MUST be an ordered subsequence of the inbound visible chain and MUST NOT introduce any actor not visible in the inbound token.

18.2. Refresh-Exchange

A current actor MAY use token exchange to refresh a short-lived transport token without appending the actor chain or regenerating a step proof.

18.2.1. Request Format

A Refresh-Exchange request MUST include:

* grant_type=urn:ietf:params:oauth:grant-type:token-exchange;
* actor_chain_refresh=true;
* actor_chain_profile set to the active profile identifier carried by the inbound token;
* the current inbound actor-chain token as the RFC 8693 subject_token;
* subject_token_type=urn:ietf:params:oauth:token-type:access_token;
* the same authenticated current actor that is continuing that accepted chain state; and
* any requested OAuth targeting parameters (audience, resource, or both). If omitted, the requested target context is the same as the inbound token’s target context as represented by aud and any locally retained target-selection context associated with that accepted token state.

A Refresh-Exchange request MUST NOT include actor_chain_step_proof, because Refresh-Exchange preserves rather than extends the accepted chain state.

A Refresh-Exchange request MUST NOT broaden the active profile, represented actor identity, visible chain state visible to the current actor, commitment state, or target context. The requested target context MUST be identical to, or a locally authorized narrowing of, the target context already represented by the inbound token and any associated retained target-selection state according to local policy. Such narrowing MUST preserve the same recipient identity; it MUST NOT retarget the token to an audience or resource server not already present in the inbound token’s canonical target_context.

18.2.2. Processing Rules

When processing Refresh-Exchange, the Authorization Server MUST:

* validate the inbound token and the identity of the current actor;
* verify that the requested profile identifier exactly matches the inbound token's actp;
* verify intended-recipient semantics as applicable;
* verify that the request does not append the chain, alter preserved chain state, broaden target context, or change recipient identity; and
* issue a replacement token with a new jti and refreshed exp.

For Refresh-Exchange, the Authorization Server MUST preserve acti, actp, sub, act, if present, and actc, if present, exactly as verified for the current actor. A new step proof MUST NOT be required, and a new commitment object MUST NOT be created. If Refresh-Exchange narrows the target context, retained step proofs and preserved actc continue to reflect the target context that was bound during the original chain-extending hop, not the narrower refreshed target context.

This specification does not define or require any particular token-binding, presenter-binding, or key-transition mechanism for Refresh-Exchange. If a deployment changes such transport or authentication properties during refresh, that handling is governed by local policy and any companion specifications rather than by this document. Historical step proofs remain bound to the keys used when those proofs were created and MUST be verified against those historical bindings, not against later local authentication material.

A recipient or coordinating component MUST treat a token obtained by Refresh-Exchange as representing the same accepted chain state as the inbound token from which it was refreshed. If local policy records a presenter-binding or key-transition event, later verifiers rely on those local records or other retained evidence for that event itself.

18.2.3. Returned-Token Validation

When validating a token returned by Refresh-Exchange, the current actor does not recompute a new commitment object from a new step proof. Instead, it MUST verify the token signature and MUST verify that preserved chain-state fields, including actp, acti, sub, act, and actc, are unchanged from the inbound token except where this specification explicitly permits refresh- specific changes such as jti, exp, or locally managed transport metadata.

19. Optional Receiver Acknowledgment Extension

A recipient MAY produce a receiver acknowledgment artifact, called hop_ack, for an inbound actor-chain token. This OPTIONAL extension does not alter chain progression semantics.

A valid hop_ack proves that the recipient accepted responsibility for the identified hop, bound to the actor-chain identifier, the identified inbound hop state, recipient, target context, and the acknowledged inbound token instance via inbound_jti. For verified profiles, the stronger workflow-level correlation anchors are acti together with the inbound commitment digest for that hop; inbound_jti remains the token-instance trace field. If the deployment establishes a presenter ActorID for the acknowledged hop under local policy, hop_ack MAY additionally bind that presenter. For declared profiles, the inbound hop state is the verified visible act from the inbound token when that profile disclosed act on the acknowledged hop. For verified profiles, that inbound hop state is the verified inbound commitment digest extracted from the inbound token’s actc.curr.

A recipient can issue a valid hop_ack only if it can either deterministically derive or receive the exact canonical target_context value for the acknowledged hop. When target_context extends beyond plain aud, the caller or a coordinating component MUST communicate that exact canonical JSON value to the recipient by an integrity-protected application mechanism before expecting a matching hop_ack.

A hop_ack is ordinarily returned to the presenting actor or to a coordinating component acting for that hop. It MAY later be archived, forwarded, or made available to an audit service or other authorized dispute-resolution component under local policy.

hop_ack MUST NOT by itself append the recipient to the actor chain.

A recipient MUST NOT emit hop_ack with status accepted until it has either:

* completed the requested operation; or
* durably recorded sufficient state to recover, retry, or otherwise honor the accepted request according to local reliability policy.

A deployment MAY require hop_ack for selected hops, including terminal hops. When hop_ack is required by policy, the calling actor and any coordinating component MUST treat that hop as not accepted unless a valid hop_ack is received and verified.

Deployments that rely on hop_ack for later audit or dispute resolution SHOULD ensure that the caller side and the recipient side each retain the artifact, or records sufficient to validate and contextualize it, for the applicable audit period. For example, a downstream agent might bill the presenting agent only for work that it accepted. In that case, the recipient-signed hop_ack helps both sides later prove exactly which hop, target context, and accepted inbound artifact are in scope.

hop_ack does not by itself prove successful completion or correctness of the requested operation.

Recipients are not required to issue hop_ack for rejected, malformed, abusive, unauthorized, or rate-limited requests. Absence of hop_ack is sufficient to prevent proof of acceptance.

When a deployment needs hop_ack to acknowledge multiple distinct operations performed under the same inbound token and the same nominal target, it MUST include a request-unique request_id inside target_context.

The acknowledgment payload MUST include at least:

* ctx = actor-chain-hop-ack-v1;
* acti;
* actp;
* jti, a unique identifier for the hop_ack JWT itself;
* inbound_jti, copied from the acknowledged inbound token;
* OPTIONAL presenter, the presenting actor’s ActorID when established under local policy for that hop;
* recipient, the acknowledging recipient’s ActorID;
* target_context;
* iat, a JWT NumericDate recording when the acknowledgment was issued;
* exp, a short-lived JWT NumericDate;
* for declared profiles, OPTIONAL inbound visible act when that profile disclosed act on the acknowledged hop;
* for verified profiles, inbound_commitment, the verified inbound commitment digest copied directly from the inbound token’s actc.curr; and
* ack, whose value MUST be accepted.

A hop_ack MUST be signed by the recipient using JWS.

19.1. Receiver Acknowledgment Validation

A caller or coordinating component that receives hop_ack and relies on it for acceptance processing MUST verify at least:

* the JWS signature using the recipient identity and keying material expected by local trust policy;
* the JWS protected header contains typ=act-hop-ack+jwt;
* ctx=actor-chain-hop-ack-v1;
* acti equals the actor-chain identifier of the inbound token for which acknowledgment is being evaluated;
* actp equals the active profile of that inbound token;
* jti is unique for the acknowledgment artifact under local replay policy;
* inbound_jti equals the jti of the inbound token that was actually sent to the recipient;
* if presenter is present, it equals the presenting actor established under local policy for the acknowledged hop;
* recipient equals the recipient from which acknowledgment is expected;
* target_context equals the exact canonical target context that was requested, communicated, or deterministically derived for the acknowledged hop;
* iat is present and acceptable under local clock policy;
* exp has not expired;
* for declared profiles, if act is present, the carried act equals the inbound visible act for the acknowledged hop and MUST NOT disclose more than that acknowledged hop disclosed;
* for verified profiles, inbound_commitment equals the verified inbound commitment digest copied from the inbound token’s actc.curr; and
* the ack member is present and its value equals accepted.

When the inbound token being acknowledged was obtained by cross-domain re-issuance or Refresh-Exchange, the target_context compared here is the exact canonical value for that acknowledged presentation. Any preserved step proofs and actc from an earlier chain-extending hop continue to reflect the target context of that earlier hop, not a later locally rewritten audience, unless those values are identical.

20. Common Security and Enforcement Requirements

This section collects enforcement requirements that all profiles rely on but that need not be read before the main profile flows. Implementations still MUST satisfy these requirements even when they are consulted later in a first reading pass.

20.1. Actor Authentication and Presenter Binding

This specification does not define or require any particular actor-authentication, presenter-binding, or token-binding mechanism. Authorization Servers and recipients MAY establish current-actor or presenting-actor identity using any locally trusted method. Recipient authorization policy based on such inputs is outside the scope of this specification.

20.2. Actor and Recipient Proof Keys

For verified profiles and for hop_ack, any signature used as a profile-defined proof MUST be generated with an asymmetric key whose verification material is trusted under local policy for the actor or recipient identity represented in that artifact.

For a verified profile step proof, the ActorID represented in the proof and the key used to sign the proof MUST be bound to the same actor identity under local trust policy. The Authorization Server MUST verify the proof using trusted verification material for that actor identity before accepting the proof.

For hop_ack, the recipient ActorID and the key used to sign the acknowledgment MUST likewise be bound to the same recipient identity under local trust policy. If presenter is included in a hop_ack, that value MUST be established under local policy and MUST NOT expand disclosure beyond the selected profile.

Shared client secrets MUST NOT be the sole basis for independently verifiable step proofs or receiver acknowledgments.

Deployments that rely on later verification of archived step proofs or acknowledgments MUST retain, or be able to recover, the verification material and identity-binding records needed to validate those signatures during the applicable audit period. Deployments that claim verified-profile auditability beyond Authorization-Server-only trust SHOULD also retain, or be able to recover, the exact compact JWS step-proof artifacts and their associated workflow context for the applicable audit period, because commitment digests alone do not prove which actor signed which hop.

20.3. Intended Recipient Validation

When a current actor submits an inbound token as a subject_token in token exchange, the accepting Authorization Server MUST normally verify that the authenticated current actor was an intended recipient of that inbound token according to local audience, resource, or equivalent validation rules. For actor_chain_refresh=true and actor_chain_cross_domain=true, this intended- recipient check does not apply, because the current actor is legitimately redeeming a token it holds as the presenter in order to refresh or preserve previously established chain state.

Possession of an inbound token alone is insufficient.

20.4. Replay and Freshness

Recipients and Authorization Servers MUST enforce replay and freshness checks on inbound tokens according to local policy.

For profiles that use actor-signed step proofs, the accepting Authorization Server:

* MUST detect replay of a previously accepted step proof within its replay-retention window;
* MUST treat an exact replay of a previously accepted compact-JWS step proof for the same authenticated actor and same prior state as an idempotent retry, not as a distinct successor;
* MUST, for such an idempotent retry, return the previously accepted successor state, or an equivalent token representing that same accepted successor state, while any required retry record is retained; and
* SHOULD, during that retry-retention window, retain the exact previously issued response or otherwise ensure that a retried response carries the same accepted chain state, because recomputing with probabilistic signatures can change wire bytes even when the decoded accepted state is equivalent; and
* MUST reject a different attempted successor for the same (acti, prior_state, target_context) tuple unless local policy explicitly authorizes replacement or supersession; this base specification does not standardize how multiple accepted successors that share earlier history are correlated or later merged. Deployments that expect multiple distinct same-target successors SHOULD distinguish them by including a unique request_id in target_context.

21. Authorization Server Metadata

Actor-chain capability discovery uses OAuth 2.0 Authorization Server Metadata {{!RFC8414}}. This specification does not define a new discovery endpoint. Clients retrieve Authorization Server metadata from the RFC 8414 well-known metadata endpoint derived from the issuer, verify that the returned issuer matches the configured issuer, and then process the actor-chain-specific metadata values defined below.

An Authorization Server supporting this specification SHOULD publish metadata describing its supported actor-chain capabilities.

This specification defines the following Authorization Server metadata values:

* actor_chain_bootstrap_endpoint: URL of the Authorization Server endpoint used to mint verified profile bootstrap context for initial actors;
* actor_chain_profiles_supported: array of supported actor-chain profile identifiers. Each value MUST be the exact identifier string used both as the actor_chain_profile token request parameter value and as the actp token claim value;
* actor_chain_commitment_hashes_supported: array of supported commitment hash algorithm identifiers;
* actor_chain_receiver_ack_supported: boolean indicating whether the Authorization Server supports processing and policy for hop_ack;
* actor_chain_refresh_supported: boolean indicating whether the Authorization Server supports Refresh-Exchange processing under this specification; and
* actor_chain_cross_domain_supported: boolean indicating whether the Authorization Server supports cross-domain re-issuance processing under this specification.

Client behavior is:

1. obtain or configure the Authorization Server issuer;
2. fetch RFC 8414 metadata from the corresponding well-known endpoint;
3. verify the returned issuer exactly matches the configured issuer;
4. check actor_chain_profiles_supported and any other needed actor-chain capability fields; and
5. fail closed if the required profile or capability is absent, unless local policy explicitly allows fallback to plain RFC 8693 behavior.

If omitted, clients MUST NOT assume support for any actor-chain profile beyond out-of-band agreement.

22. Error Handling

Token exchange errors in this specification build on OAuth 2.0 and OAuth 2.0 Token Exchange.

An Authorization Server processing a token exchange request applies the following mapping:

OAuth error code	Triggering condition
invalid_request	Malformed or missing profile-defined parameters, malformed bootstrap context, malformed ActorID values, malformed commitment objects, unsupported profile bindings, both actor_chain_cross_domain=true and actor_chain_refresh=true, or structurally malformed inline act
invalid_target	The requested audience, canonical target context, or recipient is not permitted or not supported, or a Refresh-Exchange attempts to retarget to a different recipient
invalid_grant	The subject_token fails validation, the intended-recipient check fails, continuity fails at token exchange, replay or freshness checks fail, actor_chain_step_proof verification fails, bootstrap-context reuse that is neither an idempotent retry nor an authorized distinct initial successor is detected, required inline act disclosure is absent, profile-disclosure rules fail, or the submitted prior state is inconsistent with the claimed profile state

Table 3

Recipients and Authorization Servers MUST return protocol-appropriate error signals for authentication, authorization, profile-validation, and continuity failures. When the selected profile requires inline act disclosure for an artifact, omission of act, or presence of an act value that fails that profile’s disclosure rules, MUST be treated as a profile-validation failure.

In HTTP deployments, this typically maps to 400-series status codes and OAuth-appropriate error values. In non-HTTP deployments, functionally equivalent protocol-native error signaling MUST be used.

Error responses and logs MUST NOT disclose undisclosed prior actors, full step proofs, canonical proof inputs, or other sensitive proof material unless the deployment explicitly requires such disclosure for diagnostics.

23. Part II. Security, Privacy, Deployment, and Rationale

24. Motivating Real-World Examples

24.1. Full Disclosure: Emergency Production Change in Critical Infrastructure

A safety-critical production environment requires a hotfix during an incident. The workflow is:

* A = on-call engineer;
* B = incident commander;
* C = security approver;
* D = deployment service; and
* E = runtime control plane.

Every downstream hop must see the complete visible approval path before allowing the change to proceed. The deployment service and runtime control plane both need to verify inline that the on-call engineer initiated the action, the incident commander approved it, and the security approver signed off. This is the motivating case for the Full Disclosure profiles: every hop needs the same full visible lineage for normal online authorization.

24.2. Subset Disclosure: Regulated M&A Review

A large acquisition review passes through multiple highly sensitive business functions. The workflow is:

* A = market-intelligence team;
* B = product-strategy review;
* C = internal legal review;
* D = antitrust counsel; and
* E = Chief Executive Officer.

Intermediate actors are intentionally compartmentalized and may be forbidden from learning who else contributed to the decision. The CEO, however, may need a richer disclosed chain before approving the transaction. This is the motivating case for the Subset Disclosure

profiles: the Authorization Server retains authoritative workflow state and discloses to each recipient only the portion of the chain that recipient is allowed to learn.

24.3. Actor-Only Disclosure: Bank Wire-Payment Processing Pipeline

A bank processes high-value wire payments through a sequence of narrowly scoped control services. The workflow is:

* A = payment-initiation service;
* B = sanctions-screening service;
* C = fraud-scoring service;
* D = treasury and limit-check service;
* E = payment-release service; and
* F = SWIFT or bank-connector service.

Each stage authorizes only on the immediately preceding actor for that hop. The sanctions service needs to know only that the payment-initiation service invoked it; the fraud service needs to know only that sanctions invoked it; and so on. Revealing the entire upstream pipeline to every stage adds no value to the local authorization decision and unnecessarily exposes internal control topology. This is the motivating case for the Actor-Only profiles: every hop sees only the current actor inline, while richer workflow state may still be retained by the Authorization Server for forensic review, legal audit, or incident review.

25. Security Considerations

A fuller threat discussion appears in Appendix I. This section keeps only the security considerations that directly affect interoperable processing or likely implementation choices.

25.1. Actor Authentication and Presenter Binding Are Deployment-Specific

This specification assumes that Authorization Servers can determine the current actor for an exchange and that recipients MAY establish the presenting actor for a hop under local policy when needed. The mechanisms that provide those inputs are intentionally outside the scope of this specification.

25.2. Canonicalization Errors Break Interoperability and Proof Validity

Any ambiguity in canonical serialization, actor identity representation, target representation, or proof payload encoding can cause false verification failures or inconsistent commitment values across implementations.

25.3. Readable Chain Does Not Prevent Payload Abuse

A valid visible act does not imply that the application-layer request content is safe, correct, or policy-conformant. Recipients MUST apply local payload validation and authorization.

25.4. Verified Profiles Depend on Proof Retention

The evidentiary benefits of the verified profiles depend on retention or recoverability of step proofs, exchange records, and relevant verification material. Without such retention, the profiles still provide structured commitment state, but post hoc provability and non-repudiation are materially weakened.

Authorization Servers supporting verified profiles SHOULD retain proof state, exchange records, authoritative workflow chain state, and the historical verification material needed for later verification for at least the maximum validity period of the longest-lived relevant token plus a deployment-configured audit window. Retention policies SHOULD also account for later verification during or after key rotation.

25.5. Subset-Disclosure Profiles Reveal Only a Verified Subset

Recipients using the subset-disclosure profiles can authorize based only on the disclosed visible chain subset that they verify. They MUST treat undisclosed prior actors as unavailable and MUST NOT infer adjacency, absence, or exact chain length from the disclosed subset alone.

In this base JWT/JWS binding, the returned ordinary token is visible to the current actor and to the next recipient. Therefore, a returned subset-disclosure token cannot safely reveal an actor identity that the current actor is not permitted to learn. A future recipient-protected disclosure mechanism MAY relax that limitation, but it is outside this base specification.

A singleton visible chain containing only the current actor is a valid subset-disclosure outcome. In that case, downstream authorization is current- actor-only even though the underlying profile remains one of the subset profiles unless actp explicitly selects an actor-only profile.

25.6. Cross-Domain Re-Issuance Must Preserve Chain State

A cross-domain Authorization Server that re-issues a local token for the next recipient MUST preserve the relevant visible chain state unchanged. For sub, it MAY translate the representation only when it preserves the same underlying subject semantics and does not broaden disclosure.

25.7. Branch Reconstruction Is an Audit Concern

This specification allows an Authorization Server to issue multiple successor tokens from one prior accepted state, but it does not standardize merge or sibling-discovery semantics across branches. Reconstructing a branched call graph is therefore a forensic or legal-audit concern rather than a normal online authorization requirement.

Retained Authorization Server records, timestamps, commitment state, and causal links among presenting actors, current actors, and subsequent actors can often reveal much of the effective call graph, but this base specification alone does not guarantee a complete standardized graph across all branches.

This specification does not standardize per-actor causal timestamps inside nested act. Deployments that need branch and time reconstruction SHOULD rely on retained Authorization Server records, ordinary token or proof timestamps, commitment state, and parent-child causal linkage across accepted successor tokens rather than embedding chronology fields in visible act entries.

25.8. Intended Recipient Checks Reduce Confused-Deputy Risk

Accepting Authorization Servers MUST ensure that the authenticated current actor was an intended recipient of the inbound subject_token. This reduces a class of deputy and repurposing attacks, though it does not eliminate all confused-deputy scenarios.

25.9. Chain Depth

Authorization Servers SHOULD enforce a configurable maximum chain depth. A RECOMMENDED default is 10 entries. Relying Parties MAY enforce stricter limits.

25.10. Key Management

Actors SHOULD use short-lived keys and/or hardware-protected keys. Deployments that require long-term auditability MUST retain, or make durably discoverable, the historical verification material needed to validate archived step proofs and receiver acknowledgments after key rotation.

26. Privacy Considerations

This section keeps the privacy requirements that affect protocol behavior. Additional trust-boundary and operational notes appear in Appendix J.

Readable-chain profiles disclose prior actors to downstream recipients. Deployments that do not require full visible prior-actor authorization SHOULD consider one of the subset-disclosure profiles.

The stable actor-chain identifier acti correlates all accepted hops within one workflow instance. Accordingly, acti MUST be opaque and MUST NOT encode actor identity, profile selection, business semantics, or target meaning.

Even in the privacy-preserving profiles, the Authorization Server processing token exchange observes the authenticated current actor and any retained chain-related state. Accordingly, these profiles reduce ordinary-token disclosure but do not hide prior actors from the issuing Authorization Server.

Deployments concerned with minimization SHOULD consider:

* pairwise or pseudonymous actor identifiers;
* workflow-local or pairwise sub aliases;
* omission of auxiliary claims unless receiving policy depends on them; and
* the subset-disclosure profiles when partial visible-chain disclosure is sufficient.

26.1. Subset Disclosure Extensions

This specification defines subset-disclosure semantics for the Declared Subset Disclosure profile and the Verified Subset Disclosure profile. In both profiles, if act is disclosed, the recipient-visible act is a profile-defined ordered subsequence of the actor chain for that hop, carried as an ordinary visible act claim containing only the disclosed subset. A subset profile MAY also omit act entirely.

When a subset profile discloses act, that representation is the interoperable base-wire format for the disclosed subset.

Deployments MAY additionally use an optional selective-disclosure or recipient-protected encoding technique by agreement, including Selective Disclosure JWT (SD-JWT) {{!RFC9901}}, a future COSE/CBOR companion binding, or an encrypted envelope, but only as an auxiliary overlay. Such an overlay MUST NOT replace any required visible subset act representation in the interoperable base-wire format; it MAY only add an equivalent presentation form whose disclosed value matches the same recipient-visible act and does not change any required validation result.

This specification defines the following actor-chain-specific constraints on such use:

* for the Declared Subset Disclosure profile, any disclosed visible chain MUST be an ordered subsequence of the asserted chain state for that hop;
* for the Verified Subset Disclosure profile, any disclosed visible chain MUST be an ordered subsequence of the verified actor-visible chain for that hop;
* if the selected profile uses step proofs or chain commitments, those artifacts remain bound to the verified hop progression, not to a later disclosed subset;
* a verifier MUST treat undisclosed information as unavailable and MUST require disclosure of any information needed for authorization; and
* an encoding used with a Full Disclosure profile MUST reveal the full readable chain required by that profile to the recipient before authorization.

27. Appendix A. JWT Binding (Normative)

This appendix defines the JWT and JWS wire representation for profile-defined ActorID values, visible act structures, step proofs, receiver acknowledgments, and commitment objects.

27.1. ActorID in JWT

An ActorID is a JSON object with exactly two members:

* iss: a string containing the issuer identifier; and
* sub: a string containing the subject identifier.

The object MUST be serialized using JCS {{!RFC8785}} whenever it is included in profile-defined proof or commitment inputs.

When actp is present, the visible act structure in a JWT is an ActorChainNode. Newly issued profile-defined nodes MUST contain explicit iss and sub, and MAY contain a nested act member whose value is the immediately prior visible ActorChainNode. Validators MUST also be able to normalize a validated inbound node that omits iss and inherits the enclosing issuer context.

27.2. Step Proof in JWT

The actor_chain_step_proof token request parameter value MUST be a compact JWS string {{!RFC7515}}. The JWS protected header MUST contain typ=act-step-proof+jwt. The JWS payload MUST be the UTF-8 encoding of a JCS-serialized JSON object.

For all profiles in the verified branch, the payload MUST contain:

* ctx;
* acti;
* prev;
* sub;
* target_context; and
* act.

When this payload is used as commitment input through step_hash, the step_proof_bytes value is the ASCII byte sequence of the exact compact JWS serialization of the proof artifact.

The ctx member value MUST equal the profile-specific step-proof domain-separation string ds(profile) defined in Common Processing for the Verified Branch. The prev member MUST be the base64url string value of the prior commitment digest or bootstrap seed, copied directly from the verified inbound actc.curr or bootstrap response, respectively. The sub member MUST be the exact preserved workflow sub string for the same-domain hop being extended. The act member MUST be an ActorChainNode structure and denotes the profile-defined verified actor-visible chain for the hop. For the Verified Subset Disclosure profile, the returned ordinary-token act MAY disclose any ordered subsequence of that verified chain that disclosure policy permits, or MAY omit act entirely when that profile and local policy permit omission. For the Verified Actor-Only Disclosure profile, the returned ordinary-token act MUST contain only the outermost current actor. The target_context member MUST conform to the representation defined in Target Context Requirements.

The JWS algorithm MUST be an asymmetric algorithm. The none algorithm MUST NOT be used. The JWS verification key MUST be trusted under local policy for the ActorID represented in the proof.

27.3. Receiver Acknowledgment in JWT

A hop_ack, when used in a JWT deployment, MUST be a compact JWS string {{!RFC7515}}. The JWS protected header MUST contain typ=act-hop-ack+jwt. The JWS payload MUST be the UTF-8 encoding of a JCS-serialized JSON object with at least these members:

* ctx;
* acti;
* actp;
* jti;
* inbound_jti;
* iat;
* exp;
* target_context;
* OPTIONAL presenter;
* recipient;
* for declared profiles, OPTIONAL act as permitted by the selected profile’s disclosure rules for the acknowledged inbound hop;
* for verified profiles, inbound_commitment; and
* ack.

The ctx member value MUST equal actor-chain-hop-ack-v1. The recipient member MUST be an ActorID object. If presenter is present, it MUST be an ActorID object established under local policy for that hop. The ack member MUST have the value accepted. The jti member MUST uniquely identify the hop_ack JWT itself. The inbound_jti member MUST carry the jti value from the acknowledged inbound token. The iat and exp members MUST be JWT NumericDate values, and exp SHOULD be short-lived according to local policy. The target_context member MUST conform to the representation defined in Target Context Requirements. For declared profiles, the act member, when present, MUST carry the verified inbound visible act value and MUST NOT disclose more than the selected profile allowed on the acknowledged inbound hop. For verified profiles, the inbound_commitment member MUST be copied directly from the verified inbound commitment digest extracted from the inbound token’s actc.curr. The JWS signer MUST be the recipient, and the verification key MUST be trusted under local policy for that recipient ActorID.

27.4. Commitment Object in JWT

The actc claim value MUST be a compact JWS string {{!RFC7515}}. The JWS protected header MUST contain typ=act-commitment+jwt.

The JWS payload MUST be the UTF-8 encoding of a JCS-serialized JSON object with exactly these members:

* ctx;
* iss;
* acti;
* actp;
* halg;
* prev;
* step_hash; and
* curr.

The meaning of those members is defined in the main text. actc is a commitment-ledger artifact, not an access token, and therefore does not use access-token lifetime claims such as exp.

28. Appendix B. Compact End-to-End Examples (Informative)

28.1. Example 1: Declared Full Disclosure in One Domain

Assume A, B, and C are governed by AS1.

1. A requests a token for B under the Declared Full Disclosure profile.
2. AS1 issues T_A with act=EncodeVisibleChain([A]) and aud=B.
3. A calls B and presents T_A.
4. B validates T_A, verifies continuity, and exchanges T_A at AS1 for a token to C.
5. AS1 authenticates B, verifies that B was an intended recipient of the inbound token, appends B, and issues T_B with act=EncodeVisibleChain([A,B]) and aud=C.
6. B validates that the returned visible chain is exactly the prior chain plus B.
7. B presents T_B to C.
8. C validates the token and authorizes based on the visible chain [A,B].

28.2. Example 2: Declared Subset Disclosure

Assume A, B, and C use the Declared Subset Disclosure profile and accept the issuing AS as the trust anchor for disclosure policy.

1. A requests a token for B under the Declared Subset Disclosure profile.
2. AS1 issues T_A with a recipient-specific disclosed visible act intended for B, or omits act entirely according to local policy.
3. A calls B and presents T_A.

4. B validates the token and uses only the disclosed visible chain, if any, for actor-chain authorization. If act is omitted, this specification provides no inline actor-chain input beyond what is disclosed; any additional authorization inputs come from local policy.
5. B exchanges T_A at AS1 for a token to C.
6. AS1 appends B to the accepted workflow state for that hop, applies disclosure policy for C, and issues T_B with a recipient-specific disclosed visible act, or with no act, according to local policy.
7. B presents T_B to C.
8. C validates the token and authorizes only on the disclosed visible chain, if any. If act is omitted or does not disclose B, this specification provides no inline current- actor disclosure for that hop, and C treats undisclosed actors as unavailable.

28.3. Example 2b: Declared Actor-Only Disclosure

Assume A, B, and C use the Declared Actor-Only Disclosure profile.

1. A requests a token for B under the Declared Actor-Only Disclosure profile.
2. AS1 issues T_A with act=EncodeVisibleChain([A]) and aud=B.
3. A calls B and presents T_A.
4. B validates that the visible act identifies only the current actor A and authorizes only on that actor and local policy.
5. B exchanges T_A at AS1 for a token to C.
6. AS1 issues T_B with act=EncodeVisibleChain([B]) and aud=C.
7. B presents T_B to C.
8. C validates the token and authorizes only on the visible current actor B and local policy.

28.4. Example 3: Verified Full Disclosure Across Two Domains

Assume A and B are governed by AS1, while C is governed by AS2.

1. A obtains bootstrap context from AS1, signs chain_sig_A, and receives T_A with act=EncodeVisibleChain([A]) and actc.
2. A calls B with T_A.
3. B validates T_A, constructs [A,B], signs chain_sig_B, and exchanges T_A at AS1 for a token to C.
4. AS1 verifies chain_sig_B submitted as actor_chain_step_proof, updates the commitment, and issues T_B with act=EncodeVisibleChain([A,B]) and aud=C.
5. Because C does not trust AS1 directly, B performs a second exchange at AS2.

6. AS2 preserves actp, acti, act=EncodeVisibleChain([A,B]), and actc, and issues a local token trusted by C that still represents B.
7. C validates the local token, sees the visible chain [A,B], and authorizes accordingly.

28.5. Example 4: Verified Actor-Only Disclosure

Assume A, B, and C use the Verified Actor-Only Disclosure profile.

1. A obtains bootstrap context, signs chain_sig_A over visible chain [A], and receives T_A with act=EncodeVisibleChain([A]) and actc.
2. A calls B with T_A.
3. B validates T_A, verifies that A is the presenter, constructs the verified actor-visible chain [A,B], signs chain_sig_B, and exchanges T_A at its home AS to obtain T_B for C.
4. T_B contains the updated actc and visible act=EncodeVisibleChain([B]).
5. B presents T_B to C.
6. C validates the token and authorizes based only on the disclosed current actor B and local policy. C MUST NOT infer prior-actor identity or count from undisclosed information or from actc alone.

28.6. Example 5: Verified Subset Disclosure

Assume A, B, and C use the Verified Subset Disclosure profile.

1. A obtains bootstrap context, signs chain_sig_A, and receives T_A with a recipient-specific disclosed visible act, or with no act, plus actc intended for B according to local policy.
2. A calls B and presents T_A.
3. B validates the token and uses only the disclosed visible chain, if any, for actor-chain authorization. If act is omitted, this specification provides no inline actor-chain input beyond what is disclosed; any additional authorization inputs come from local policy.
4. B signs chain_sig_B over the exact actor-visible chain that B verified on the inbound hop, with B appended, and exchanges T_A at its home AS alongside chain_sig_B as actor_chain_step_proof to obtain T_B for C.
5. AS1 verifies that submitted chain state, applies disclosure policy for C, and issues T_B with a recipient-specific disclosed visible act, or with no act, and updated actc.
6. B presents T_B to C.

7. C validates the token and authorizes only on the disclosed visible chain, if any. If act is omitted or does not disclose B, this specification provides no inline current- actor disclosure for that hop, and C treats undisclosed actors as unavailable.
8. If later audit is needed, the verified actor-visible chain for the hop can be reconstructed from retained step proofs together with exchange records and, when subset disclosure omitted act, any retained Authorization-Server workflow records needed to supply hidden continuity context.

29. Appendix C. Future Considerations (Informative)

29.1. Terminal Receipts and Result Attestations

This specification defines special handling for the first actor in order to initialize chain state. It does not define corresponding terminal-hop semantics for a final recipient that performs work locally and does not extend the chain further.

Future work MAY define:

* a terminal receipt proving that the recipient accepted the request;
* an execution attestation proving that the recipient executed a specific operation; and
* a result attestation binding an outcome or result digest to the final commitment state.

29.2. Bootstrap Receipts and Portable Initial-State Evidence

This specification does not define a bootstrap-signed receipt artifact. Later audit of bootstrap processing therefore relies on Authorization Server records, the bootstrap response, the initial step proof, and the first issued ordinary token.

Future work MAY define a portable bootstrap receipt or bootstrap attestation artifact if deployments need independently portable evidence of workflow initialization outside Authorization Server logs.

29.3. Receiver Acceptance and Unsolicited Victim Mitigation

This specification deliberately does not append a recipient merely because that recipient was contacted. It also defines an OPTIONAL hop_ack extension that lets a recipient prove accepted responsibility for a hop.

However, this specification still does not by itself prevent a malicious actor from sending a validly issued token to an unsolicited victim service. Future work MAY define stronger receiver-driven protections, including:

* stronger result attestations for completed terminal work;
* a challenge-response model for high-risk terminal hops; and
* recipient-issued nonces or capabilities that MUST be bound into the final accepted hop.

29.4. Subset Disclosure Extensions

This document now defines baseline Declared Subset Disclosure and Verified Subset Disclosure profiles at the actor-chain semantics layer. Future work MAY define stronger or more standardized subset-disclosure encodings and verification techniques, including Selective Disclosure JWT (SD-JWT) {{!RFC9901}}, a future COSE/CBOR companion binding, recipient-bound disclosure artifacts, zero-knowledge proofs over the canonical full chain, or richer verifier-assisted consistency checks against retained proof state.

Any future encoding or presentation profile MUST preserve the disclosure semantics of the selected base profile. In particular, a Full Disclosure profile still requires full visible-chain disclosure to the recipient, while subset-disclosure outcomes that reveal only the current actor MUST NOT expose hidden actor entries to recipients of ordinary tokens merely as digests or selectively revealable placeholders.

29.5. Branching and Fan-Out

This specification defines one visible path per issued token and does not standardize merge or sibling-discovery semantics across multiple descendants that share earlier workflow history.

An Authorization Server MAY nevertheless mint multiple accepted successor tokens from one prior accepted state. Such branching is represented across multiple tokens, not inside one token’s nested act structure. Later reconstruction of the resulting call graph is primarily a forensic or legal- audit concern.

Future work MAY define explicit branch identifiers, parent-child workflow correlation, tree-structured commitment verification, inclusion proofs, partial disclosure across branches, and later merge behavior. Such future work could also help correlate related *WHO*, *WHAT*, and *HOW* evidence across companion Actor Chain, Intent Chain {{!I-D.draft-mw-spice-intent-chain}}, and Inference Chain {{!I-D.draft-mw-spice-inference-chain}} deployments.

29.6. Evidence Discovery and Governance Interoperability

Proof-bound profiles derive much of their value from later verification of step proofs and exchange records. Future work MAY standardize interoperable evidence discovery, retention, and verification-material publication.

Any such specification should define, at minimum, evidence object typing, authorization and privacy controls for cross-domain retrieval, stable lookup keys such as jti or acti, error handling, and retention expectations.

30. Appendix D. Design Rationale and Relation to Other Work (Informative)

This document complements {{!RFC8693}} by defining chain-aware token-exchange profiles. It also fits alongside the broader SPICE service-to-service and attestation work {{!I-D.ietf-spice-s2s-protocol}} {{!I-D.draft-mw-spice-transitive-attestation}} and composes with companion SPICE provenance work: Actor Chain addresses *WHO* acted, Intent Chain {{!I-D.draft-mw-spice-intent-chain}} addresses *WHAT* was produced or transformed, and Inference Chain {{!I-D.draft-mw-spice-inference-chain}} addresses *HOW* a result was computed.

This specification defines six profiles instead of one deployment mode so that implementations can choose among full visible chain-based authorization, trust-first partial disclosure, explicit actor-only operation, stronger commitment-state accountability, recipient-specific commitment-backed partial disclosure, and verified actor-only disclosure without changing the core progression model.

The base specification remains linear. Branching, richer disclosure mechanisms, and evidence-discovery protocols remain future work because they require additional identifiers, validation rules, and interoperability work.

31. Appendix E. Implementation Conformance Checklist (Informative)

An implementation is conformant only if it correctly implements the profile it claims to support and all common requirements on which that profile depends.

At a minimum, implementers should verify that they have addressed the following:

Requirement	Draft section reference	Implemented [ ]
Stable generation and preservation of acti, using a CSPRNG with at least 122 bits of entropy (for example, standard UUIDv4 or stronger generation)	Actor-Chain Identifier (acti)	[ ]
Local policy for actor authentication or presenter binding, if relied upon	Actor Authentication and Presenter Binding	[ ]
Exact ActorID equality over (iss, sub)	Actor Identity Representation	[ ]
Canonical serialization for all proof and commitment inputs	Canonicalization; Target Context Requirements; Appendix F	[ ]
Intended-recipient validation during token exchange	Intended Recipient Validation	[ ]
Replay and freshness handling for tokens and step proofs	Replay and Freshness	[ ]
Exact append-only checks for full-disclosure profiles	Declared Full Disclosure Profile; Verified Full Disclosure Profile	[ ]
Correct ordered-subsequence validation for subset-disclosure profiles	Declared Subset Disclosure Profile; Verified Subset Disclosure Profile	[ ]
Current-actor-only validation for actor-only profiles	Declared Actor-Only Disclosure Profile; Verified Actor-Only Disclosure Profile	[ ]
Exact commitment verification for verified	Commitment Function; Verified Full	[ ]

profiles	Disclosure Profile; Verified Subset Disclosure Profile; Verified Actor-Only Disclosure Profile	
Proof-key binding between ActorID and proof signer under local trust policy	Actor and Recipient Proof Keys	[ ]
Non-broadening Refresh-Exchange processing, if supported	Refresh-Exchange	[ ]
Correct binding and one-time redemption of bootstrap context for verified workflows	Common Bootstrap Context Request; Common Initial Actor Step Proof and Bootstrap Issuance	[ ]
Policy for when hop_ack is optional or required	Optional Receiver Acknowledgment Extension	[ ]
Preserve-state handling for cross-domain re-issuance and Refresh-Exchange, if supported	Cross-Domain Re-Issuance; Refresh-Exchange	[ ]
Privacy-preserving handling of logs and error messages	Error Handling; Privacy Considerations	[ ]

Table 4

32. Appendix F. Canonicalization Test Vectors (Informative)

The following illustrative vectors are intended to reduce interoperability failures caused by divergent canonicalization. They are not exhaustive, but they provide concrete byte-for-byte examples for common JWT/JCS ActorID and target_context inputs.

32.1. JWT / JCS ActorID Example

Input object:

{::nomarkdown} <sourcecode type="json">
{"iss":"https://as.example","sub":"svc:planner"
(https://as.example","sub":"svc:planner")} </sourcecode>
{:/nomarkdown}

JCS serialization (UTF-8 bytes rendered as hex):

{::nomarkdown} <sourcecode type="text">
7b22697373223a2268747470733a2f2f61732e6578616d706c65222c22737562223a227376633a706c616e
6e6572227d
</sourcecode> {:/nomarkdown}

SHA-256 over those bytes:

{::nomarkdown} <sourcecode type="text">
7a14a23707a3a723fd6437a4a0037cc974150e2d1b63f4d64c6022196a57b69f
</sourcecode> {:/nomarkdown}

32.2. JWT / JCS target_context Example

Input object:

{::nomarkdown} <sourcecode type="json">
{"aud":"https://api.example","method":"invoke","resource":"calendar.read"
(https://api.example","method":"invoke","resource":"calendar.read")}
</sourcecode> {:/nomarkdown}

JCS serialization (UTF-8 bytes rendered as hex):

{::nomarkdown} <sourcecode type="text">
7b22617564223a2268747470733a2f2f6170692e6578616d706c65222c226d6574686f64223a22696e766f
6b65222c227265736f75726365223a2263616c656e6461722e72656164227d
</sourcecode> {:/nomarkdown}

SHA-256 over those bytes:

{::nomarkdown} <sourcecode type="text">
911427869c76f397e096279057dd1396fe2eda1ac9e313b357d9cecc44aa811e
</sourcecode> {:/nomarkdown}

33. Appendix G. Illustrative Wire-Format Example (Informative)

This appendix shows one abbreviated decoded JWT payload together with one abbreviated decoded actc JWS payload. The values are illustrative and signatures are omitted for readability.

33.1. Decoded Access Token Payload Example

{::nomarkdown} <sourcecode type="json"> { "iss": "https://as.example" (https://as.example"), "sub": "svc:planner", "act": { "iss": "https://as.example" (https://as.example"), "sub": "svc:planner", "act": { "iss": "https://as.example" (https://as.example"), "sub": "svc:orchestrator" } }, "aud": "https://api.example" (https://api.example"), "exp": 1760000000, "jti": "2b2b6f0d3f0f4d7a8c4c3c4f9e9b1a10", "acti": "6cb5f0c14ab84718a69d96d31d95f3c4", "actp": "verified-full", "actc": "<compact JWS string>" } </sourcecode> {:/nomarkdown}

33.2. Decoded actc JWS Example

Protected header:

{::nomarkdown} <sourcecode type="json"> {"alg":"ES256","typ":"act-commitment+jwt"} </sourcecode> {:/nomarkdown}

Payload:

{::nomarkdown} <sourcecode type="json"> { "ctx": "actor-chain-commitment-v1", "iss": "https://as.example" (https://as.example"), "acti": "6cb5f0c14ab84718a69d96d31d95f3c4", "actp": "verified-full", "halg": "sha-256", "prev": "SGlnaGx5SWxsdXN0cmF0aXZlUHJldkRpZ2VzdA", "step_hash": "z7mq8c0u9b2C0X5Q2m4Y1q3r7n6s5t4u3v2w1x0y9z8", "curr": "Vb8mR6b2vS5h6S8Y6j5X4r3w2q1p0n9m8l7k6j5h4g3" } </sourcecode> {:/nomarkdown}

On the wire, the actc claim carries the usual compact-JWS form:

{::nomarkdown} <sourcecode type="text"> BASE64URL(protected-header) "." BASE64URL(payload) "." BASE64URL(signature) </sourcecode> {:/nomarkdown}

34. Appendix H. Problem Statement and Deployment Context (Informative)

defines the top-level act claim for the current actor and allows nested prior actors. However, prior nested act claims are informational only for access-control decisions. In multi-hop systems, especially service-to-service and agentic systems, that is not sufficient.

Consider:

{::nomarkdown} <artwork type="ascii-art"> User -> Orchestrator -> Planner -> Tool Agent -> Data API </artwork> {:/nomarkdown}

By the time the request reaches the Data API, the immediate caller may or may not be visible in the token, and the upstream delegation path is not standardized as a policy input and is not bound across successive token exchanges in a way that can be independently validated or audited. This creates several concrete gaps:

* downstream policy cannot reliably evaluate the full delegation path;
* cross-exchange continuity is not standardized;
* tampering by an actor and its home AS is not uniformly addressed;
* forensic verification of per-hop participation is not standardized; and
* ordinary tokens may disclose more visible-chain information than some deployments are willing to reveal.

35. Appendix I. Threat Model (Informative)

This specification defines a multi-hop, multi-actor delegation model across one or more trust domains. The security properties provided depend on the selected profile, the trust relationship among participating Authorization Servers, and the availability of step proofs or exchange records where relied upon.

35.1. Assets

The protocol seeks to protect the following assets:

* continuity of the delegation path;
* integrity of prior-actor ordering and membership;
* continuity of the actor represented as current for each hop when such continuity is disclosed or otherwise established under local policy;
* binding of each hop to the intended target;
* resistance to replay of previously accepted hop state;
* audit evidence for later investigation and proof; and
* minimization of prior-actor disclosure where privacy-preserving profiles are used.

35.2. Adversaries

Relevant adversaries include:

* an external attacker that steals or replays a token;
* a malicious actor attempting to insert, omit, reorder, or repurpose hop state;
* a malicious actor colluding with its home Authorization Server;
* a malicious downstream recipient attempting to over-interpret or misuse an inbound token;

* an untrusted or compromised upstream Authorization Server in a multi-domain path; and
* an unsolicited victim service reached by a validly issued token without having agreed to participate.

35.3. Assumptions

This specification assumes:

* verifiers can validate token signatures and issuer trust;
* the authenticated actor identity used in token exchange is bound under local trust policy to the actor identity represented in profile-defined proofs; and
* deployments that rely on later proof verification retain, or can discover, the verification material needed to validate archived step proofs and exchange records.

35.4. Security Goals

The protocol aims to provide the following properties:

* in the Declared Full Disclosure profile, silent insertion, removal, reordering, or modification of prior actors is prevented under the assumption that an actor does not collude with its home Authorization Server;
* in the Declared Subset Disclosure profile, ordinary tokens reveal only a visible ordered subset of actors selected by the Authorization Server, and authorization is limited to that disclosed subset;
* in the Verified Subset Disclosure profile, each accepted hop is bound to an actor-signed proof over the exact actor-visible chain for that hop and to cumulative commitment state, while ordinary tokens reveal only an ordered subset of that actor-visible chain selected by the Authorization Server;
* in the Verified Full Disclosure profile, the actor-visible chain equals the full visible chain at each hop, preserving full visible authorization while improving detectability, provability, and non-repudiation; and
* in the Verified Actor-Only Disclosure profile, ordinary tokens MUST disclose only the current actor while preserving current-actor continuity where that continuity is disclosed or otherwise established under local policy, together with cumulative commitment state for later verification.

35.5. Non-Goals

This specification does not by itself provide:

* integrity or safety guarantees for application payload content;
* complete prevention of confused-deputy behavior;
* concealment of prior actors from the Authorization Server that processes token exchange;
* standardized merge or branch-selection semantics across branched work; or
* universal inline prevention of every invalid token that could be issued by a colluding actor and its home Authorization Server.

35.6. Residual Risks

Even when all checks succeed, a valid token chain does not imply that the requested downstream action is authorized by local business policy. Recipients MUST evaluate authorization using only the actor-chain information actually disclosed to them by the artifact together with token subject, intended target, and local policy.

Deployments that depend on independently verifiable provenance for high-risk operations SHOULD require synchronous validation of commitment-linked proof state or otherwise treat the issuing Authorization Server as the sole trust anchor.

In the Verified Subset Disclosure and Verified Actor-Only Disclosure profiles, a current actor signs only the exact actor-visible chain available at that hop. Those profiles therefore provide non-repudiation over the signed visible chain and linked commitment state, not over hidden prefix semantics against a rogue or colluding Authorization Server.

36. Appendix J. Trust Boundaries and Audit Guidance (Informative)

The trust model and visible-disclosure properties of the six profiles are defined in the main specification text and Appendix I. This appendix focuses on operational retention and forensic guidance rather than restating those profile summaries.

Authorization Servers supporting these profiles SHOULD retain records keyed by acti and jti.

For verified profiles, the retention period SHOULD be at least the maximum validity period of the longest-lived relevant token plus a deployment-configured audit window, and it SHOULD remain sufficient to validate historical proofs across key rotation.

For verified profiles, such records SHOULD include:

* prior token reference;

* authenticated actor identity accepted for the exchange or proof-validation step;
* step proof reference or value;
* issued token reference;
* commitment state;
* requested audience or target context; and
* timestamps.

For subset-disclosure profiles, retained records SHOULD also allow reconstruction of the verified actor-visible chain for each hop and the disclosed subset issued for each recipient. Collecting all such accepted hop evidence for one acti, including retained tokens, proofs, commitments, and exchange records, can reconstruct the accepted hop sequence, including repeated-actor revisits, and can often reveal much of the effective call graph, but this specification does not by itself yield a complete standardized graph across related branches. If a deployment also relies on a hidden full-chain prefix not signed by every acting intermediary, the Authorization Server SHOULD retain the additional state needed to reconstruct that hidden prefix for later audit.

Actors SHOULD also retain local records sufficient to support replay detection, incident investigation, and later proof of participation.

37. Appendix K. Design Decisions (Informative)

This appendix records key design decisions made in the core specification so that future revisions can preserve the underlying interoperability and security rationale even if the document is later split.

37.1. Why visible chain state is carried in nested act

The interoperable JWT form uses nested act as the single authoritative inline carrier for visible actor-chain state. This avoids maintaining separate readable chain claims and prevents dual-truth bugs in which two different claims could describe different visible histories.

37.2. Why top-level sub remains the workflow subject

Top-level sub identifies the workflow subject of the token rather than the current actor. This keeps ordinary token semantics aligned with RFC 8693 and allows the current actor to remain visible in act. Replacing sub with the current actor or with a constant marker would make token subject semantics much less clear for recipients and auditors.

37.3. Why privacy-sensitive profiles use stable subject aliases

Subset-disclosure operation can hide actors only if other visible fields do not reveal the hidden subject indirectly. The AS therefore chooses a stable workflow-subject representation at bootstrap, such as a pairwise or workflow-local alias, and preserves it for the workflow.

37.4. Why subset disclosure is recipient-specific and AS-driven

Different recipients in the same workflow can have different need-to-know. The AS is therefore the policy decision and enforcement point for disclosure and may issue a narrower or broader visible nested act to different recipients, so long as each returned token remains consistent with the active profile and any disclosed act is a permitted profile-conformant disclosure of the accepted chain state for that hop.

37.5. Why actor-only is a separate profile even though subset can express it

Actor-only disclosure can be expressed as a special case of subset disclosure, but this document gives it distinct profile identifiers so that the intended policy remains explicit and machine-readable to current and future implementations. A token carrying actp=declared-actor-only or actp=verified-actor-only therefore tells every conforming actor and recipient that ordinary tokens in that workflow MUST expose only the outermost current actor inline, even though the Authorization Server may retain richer hidden workflow state for later audit or forensics.

37.6. Why branching is represented across tokens, not inside one token

Each token carries one visible path in nested act. Branching is represented by issuing multiple successor tokens that share a prior workflow state. This keeps token syntax simple while still supporting forensic and legal-audit reconstruction of a broader call graph from retained AS records and related artifacts.

37.7. Why actc uses parent-pointer commitments instead of Merkle trees

The base commitment mechanism is hop-oriented: each accepted successor step binds exactly one parent commitment state and one new verified hop. Allowing multiple successors to share the same parent forms a hash-linked workflow graph without requiring Merkle-tree ordering rules, sibling proof transmission, or merge semantics. Future work can add Merkle-style aggregation if deployments need compact set commitments across many branches.

37.8. Why actp and actc use short claim names

These claims appear in every token issued under this specification, and actc also travels in later verified hops. The document therefore uses short claim names to limit repeated per-hop overhead on the wire. Their meanings are fixed only by this specification, and the main body defines those semantics before IANA registration.

37.9. Why RFC 8414 metadata discovery is reused

The specification reuses the standard RFC 8414 authorization-server metadata endpoint for discovery. This avoids inventing a new discovery API and keeps capability negotiation in the well-known OAuth metadata channel that implementations already use.

37.10. Privacy goals and limits

Subset disclosure limits per-hop visibility, but it does not hide information from the issuing AS or from colluding parties that pool tokens, proofs, acknowledgments, and logs. The design therefore treats complete branch and call graph reconstruction primarily as a forensics or legal-audit concern rather than an online authorization requirement.

38. IANA Considerations

This specification does not create a new hash-algorithm registry. actc uses hash algorithm names from the IANA Named Information Hash Algorithm Registry {{IANA.Hash.Algorithms}}, subject to the algorithm restrictions defined in this document.

38.1. JSON Web Token Claims Registration

This document requests registration of the following claims in the "JSON Web Token Claims" registry established by {{!RFC7519}}:

Claim Name	Claim Description	Change Controller	Specification Document(s)
actc	Proof-bound chain state binding accepted hop progression for the active profile.	IETF	[this document]
acti	Actor-chain identifier preserved across accepted hops.	IETF	[this document]
actp	Actor-chain profile identifier for the issued token.	IETF	[this document]

Table 5

38.2. Media Type Registration

This document requests registration of the following media types in the "Media Types" registry established by {{!RFC6838}}:

Media Type	Media Subtype Name	Required Parameters	Optional Parameters	Encoding Considerations	Security Considerations	Interoperability Considerations	Published Specification	Applications that use this media type	Fragment Identifier	Additional Information	Contact Author/Change Controller	Intended Usage	Restrictions on Usage
application	act-step-proof+jwt	N/A	N/A	binary	see [this document]	N/A	[this document]	OAuth 2.0 Token Exchange actor-chain step proofs	N/A	Magic Number(s): N/A; File Extension(s): N/A; Macintosh File Type Code(s): N/A	IETF	COMMON	N/A
application	act-commitment+jwt	N/A	N/A	binary	see [this document]	N/A	[this document]	OAuth 2.0 Token Exchange actor-chain commitments	N/A	Magic Number(s): N/A; File Extension(s): N/A; Macintosh File Type	IETF	COMMON	N/A

							Code(s): N/A					
application	act-hop-ack+jwt	N/A	N/A	binary	see [this document]	N/A	[this document]	IETF	IETF	COMMON	N/A	Exchange actor-chain receiver acknowledgments	OAuth 2.0 Token	N/A	Magic Number(s): N/A; File Extension(s): N/A; Macintosh File Type Code(s): N/A	IETF				

Table 6

38.3. OAuth URI Registration

This document requests registration of the following value in the "OAuth URI" registry established by {{!RFC6749}}:

URI	Description	Change Controller	Specification Document(s)
urn:ietf:params:oauth:grant-type:actor-chain-bootstrap	OAuth grant type for the initial verified profile bootstrap token request.	IETF	[this document]

Table 7

38.4. OAuth Authorization Server Metadata Registration

This document requests registration of the following metadata names in the "OAuth Authorization Server Metadata" registry established by {{!RFC8414}}:

Metadata Name	Metadata Description	Change Controller	Specification Document(s)
actor_chain_bootstrap_endpoint	Endpoint used to mint bootstrap context for verified profile initial actors.	IETF	[this document]
actor_chain_profiles_supported	Supported actor-chain profile identifiers.	IETF	[this document]
actor_chain_commitment_hashes_supported	Supported commitment hash algorithm identifiers.	IETF	[this document]
actor_chain_receiver_ack_supported	Indicates support for receiver acknowledgments (hop_ack) under this specification.	IETF	[this document]
actor_chain_refresh_supported	Indicates support for Refresh-Exchange under this specification.	IETF	[this document]
actor_chain_cross_domain_supported	Indicates support for cross-domain re-issuance under this specification.	IETF	[this document]

Table 8

38.5. OAuth Parameter Registration

This document requests registration of the following parameter names in the relevant OAuth parameter registry:

Parameter Name	Parameter Usage Location	Change Controller	Specification Document(s)
actor_chain_profile	actor-chain bootstrap endpoint request; OAuth token endpoint request	IETF	[this document]
actor_chain_bootstrap_context	actor-chain bootstrap endpoint response; OAuth token endpoint request	IETF	[this document]
actor_chain_step_proof	OAuth token endpoint request	IETF	[this document]
actor_chain_refresh	OAuth token endpoint request	IETF	[this document]
actor_chain_cross_domain	OAuth token endpoint request	IETF	[this document]

Table 9

Authors' Addresses

A Prasad
Oracle
Email: a.prasad@oracle.com

Ram Krishnan
JPMorgan Chase & Co
Email: ramkri123@gmail.com

Diego R. Lopez
Telefonica
Email: diego.r.lopez@telefonica.com

Srinivasa Addepalli
Aryaka
Email: srinivasa.addepalli@aryaka.com