SPICE
Internet-Draft
Intended status: Standards Track
Expires: 19 September 2026

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

18 March 2026

Cryptographically Verifiable Actor Chains for OAuth 2.0 Token Exchange
draft-mw-spice-actor-chain-03

Abstract

This document defines five 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 defines profile-specific processing for linear multi-hop workflows. The five profiles are: Asserted Chain with Full Disclosure; Asserted Chain with Subset Disclosure; Committed Chain with Full Disclosure; Committed Chain with Subset Disclosure; and Committed Chain with No Chain Disclosure.

These profiles preserve the existing meanings of sub and act, support same-domain and cross-domain delegation, require sender-constrained tokens, and provide different tradeoffs among readable chain-based authorization, cryptographic accountability, auditability, privacy, and long-running workflow support.

Status of This Memo

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

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

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on 19 September 2026.

Table of Contents

1. 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, a 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 model for preserving and validating delegation-path continuity across successive exchanges.

This document defines five interoperable actor-chain profiles for OAuth 2.0 Token Exchange. Across those profiles, ordinary tokens keep the familiar JSON Web Token (JWT) subject and actor model while adding interoperable actor-chain state for later validation, forwarding, and audit. For compactness on the wire, some actor-chain-specific claims use short names; the exact claim set and claim roles are defined later in Common Token Requirements and in the profile sections.

A few recurring artifacts appear throughout the document. An *ordinary token* is the sender-constrained access token issued for presentation to the next hop. Committed profiles additionally use an actor-signed *step proof*, a *bootstrap context* issued by the

Authorization Server (AS) when starting a workflow, and optionally a recipient-signed *hop acknowledgment* (hop_ack). Committed proofs and acknowledgments also bind a *target context*, which is the canonical representation of the next-hop audience together with any other profile-relevant target-selection inputs.

The design separates inline authorization from later proof and audit. Main-body sections focus on the interoperable protocol rules needed to issue, exchange, validate, and consume actor-chain tokens correctly. Implementers primarily interested in interoperable behavior can focus first on Common Basics, Common Validation Procedures, the profile sections, and Appendix A. Special preserve-state exchanges, metadata, and deeper enforcement details are surfaced later so that the normal profile flow can be read without interruption. Readers who want to start with motivation and deployment framing can begin with Appendix H and Appendix D, then return here and to Scope and Model for the implementation-first protocol view. Extended background, problem framing, threat analysis, and operational guidance appear in the appendices.

All profiles assume sender-constrained tokens together with ordinary replay and freshness protections, but the detailed enforcement rules for those mechanisms appear later so they do not interrupt the initial flow narrative.

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

2. 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 authenticated actor that presents a sender-constrained 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 sender-constrained 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 sender-constrained access token issued for presentation to the next hop. It is distinct from step proofs, bootstrap context handles, commitment objects, and receiver acknowledgments.

* *Readable chain*: An ach value carried in an ordinary token and visible to downstream recipients.

* *Actor-visible chain*: The exact ordered actor sequence that the current actor is permitted to know and extend for the next hop. In the Committed Chain with Subset Disclosure profile, this is the exact inbound disclosed ach verified by that actor, with the actor appended when it later acts. In the Committed Chain with No Chain Disclosure profile, bootstrap uses the singleton chain [A]; each non-bootstrap hop uses the exact pair [PresentingActor, CurrentActor].

* *Committed chain state*: The cumulative cryptographic state 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 chain state, 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 committed-profile workflow. It lets the initial actor redeem bound bootstrap state at the token endpoint without carrying that state inline.

* *Workflow identifier (sid)*: 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 inbound token is being presented by the actor that the chain state indicates should be presenting it.

* *Append-only processing*: The rule that a new actor is appended to the prior chain state, 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 refreshes a short-lived transport token without appending the actor chain, changing the active profile, or generating a new step proof.

3. Relationship to RFC 8693 Claims

This specification extends OAuth 2.0 Token Exchange {{!RFC8693}} without changing the existing meanings of sub, act, or may_act.

The following rules apply:

* sub continues to identify the subject of the issued token.
* act MUST identify the current actor represented by the issued token.
* ach, when present, carries the profile-defined ordered actor chain for that artifact. In full-disclosure readable profiles it is the full readable chain to date; in subset-disclosure profiles it is a recipient-specific disclosed subset that ends in the current actor; and in committed step proofs it is the proof-bound actor-visible chain for the hop.

* Nested prior act claims, if present for compatibility or deployment-specific reasons, remain informational only for access-control purposes, consistent with {{!RFC8693}}.
* Tokens issued under this specification MUST include sub and act so that downstream parties can preserve RFC 8693 subject and current-actor semantics while also carrying actor-chain state.
* may_act, when present in an inbound token, MAY be used by the accepting Authorization Server as one input when determining whether the authenticated current actor is authorized to perform token exchange for the requested target context.

Nothing in this specification redefines the delegation and impersonation semantics described in {{!RFC8693}}.

4. Scope and Model

This document specifies a family of profiles for representing and validating actor progression across a linear workflow 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.

The base workflow model is linear:

{::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.

This document defines five profiles:

* *Asserted Chain with Full Disclosure*, which carries a readable ach and relies on AS-asserted chain continuity under a non-collusion assumption;

* *Asserted Chain with Subset Disclosure*, which carries a recipient-specific disclosed subset in readable ach and relies on the issuing AS for both chain continuity and disclosure policy;
* *Committed Chain with Subset Disclosure*, which preserves cumulative committed state and lets the Authorization Server disclose a recipient-specific ordered subset of the actor-visible chain while the current actor signs the exact actor-visible chain it was allowed to see and extend;
* *Committed Chain with Full Disclosure*, which is the committed full-disclosure case in which every actor and downstream recipient sees the full readable chain; and
* *Committed Chain with No Chain Disclosure*, which is the committed no-chain-disclosure case in which ordinary tokens omit ach and downstream authorization is based on the presenting actor only.

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

* the *asserted branch*, rooted at Asserted Chain with Full Disclosure; and
* the *committed branch*, consisting of one common committed-processing section plus three disclosure modes: subset disclosure, full disclosure, and no chain disclosure.

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

* *Asserted Chain with Full Disclosure* for the asserted branch;
* *Common Processing for the Committed Branch* plus the three concise committed profile sections for the committed branch; and
* the concise delta sections for the two subset-disclosure variants.

The same small set of objects recurs throughout the rest of the document: ordinary tokens carry hop-to-hop state, committed profiles add actor-signed step proofs and achc, bootstrap is used only to start a committed-profile workflow, and target_context names the canonical next-hop target that proofs and acknowledgments bind. Keeping those four objects in mind makes the later profile sections much easier to read.

The following table is a quick orientation aid.

Profile	Readable ach in ordinary tokens	achc	Subset disclosure	Next-hop authorization basis	Primary trust/ evidence model
Asserted Chain with Full Disclosure	Full	No	No	Full readable chain	AS-asserted continuity
Asserted Chain with Subset Disclosure	Disclosed subset	No	Yes	Disclosed readable subset	AS-asserted continuity plus AS disclosure policy
Committed Chain with Subset Disclosure	Disclosed subset	Yes	Yes	Disclosed readable subset plus commitment continuity	Actor-signed visible-chain proofs plus recipient-specific disclosure
Committed Chain with Full Disclosure	Full	Yes	No	Full readable chain	Actor-signed visible-chain proofs plus cumulative commitment
Committed Chain with No Chain Disclosure	No	Yes	No	Presenting actor only	Actor-signed visible-chain proofs plus cumulative commitment

Table 1

The committed branch is best read as one common set of bootstrap, proof, commitment, and returned-token rules, followed by its disclosure modes. The special preserve-state exchanges, deeper enforcement requirements, and metadata sections can then be read afterward.

Application logic may branch, fan out, and run in parallel. This document standardizes hop-by-hop actor-chain evidence, not a full call-graph language. If later work needs standardized shared-root branching semantics, a future specification can add them, for example by binding a branch sid to a parent sid during bootstrap or a branch-creation exchange and defining any later merge or branch-selection behavior. Complete call-graph construction is typically an audit or forensic concern rather than an online authorization requirement, and retained Authorization Server records, timestamps, and causal links among presenting actors, current actors, and subsequent actors can often reveal much of the effective call graph even without such an extension, though this base specification alone does not guarantee a complete standardized graph across separate sid values.

An issued token MAY still carry an aud string or array according to JWT and OAuth conventions, but for any one step it defines one canonical next-hop target_context.

Repeated ActorID values within one linear 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 sid, such as retained tokens, proofs, commitments, and exchange records, can therefore reconstruct the accepted hop sequence, including repeated-actor revisits.

5. Protocol Overview

5.1. 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.

5.2. 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.

6. Common Basics

This section introduces the recurring fields, token contents, and cryptographic objects needed to read the profile flows. More detailed enforcement rules, including sender constraint, proof-key binding, intended-recipient checks, and replay or freshness handling, are collected later in "Common Security and Enforcement Requirements" so that the main profile story can be read first.

6.1. Common Token Requirements

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

Tokens issued under any profile defined by this document:

* MUST be short-lived;
* MUST be sender-constrained to the presenting actor; and
* MUST contain:
  - a profile identifier claim achp;
  - a workflow identifier claim sid;
  - a subject claim sub;
  - a current-actor claim act;
  - a unique token identifier claim jti;
  - an audience value aud; and
  - an expiry value exp.

The token claims used by this document have these roles:

* achp identifies the selected actor-chain profile;
* sub identifies the token subject;
* act identifies the current actor;
* ach, when present, carries the profile-defined ordered actor chain for that artifact; and
* achc, when present, carries cumulative committed chain state for stronger tamper evidence and auditability.

Profiles that preserve readable chain state additionally carry ach.

In full-disclosure readable profiles, ach carries the full readable chain to date. In subset-disclosure profiles, ach carries the recipient-specific disclosed subset that ends in the current actor. In committed step proofs, ach is the proof-bound actor-visible chain for that hop.

Profiles that preserve committed chain state additionally carry achc.

Under the base profiles defined by this document, same-domain token exchange, cross-domain re-issuance, and Refresh-Exchange preserve the inbound sub claim. This document does not define a same-workflow subject-transition mechanism.

6.2. Workflow Identifier

The sid 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 sid 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 achp.

6.3. Target Context Requirements

target_context is the canonical next-hop target value bound into committed-profile step proofs and, when used, into hop_ack. In many deployments it is just aud. Deployments that need finer-grained binding can extend it with other target-selection inputs.

The following normative requirements apply to target_context.

target_context MUST carry the verified audience information exactly in the profile-defined canonical representation. If aud is a string, target_context MAY be that same JSON string or a JSON object that includes an aud member with that same string value. If aud is an array of strings, target_context MUST represent that array exactly, either as that same JSON array value or as a JSON object whose aud member is that exact array.

A deployment MAY additionally include resource identifiers, operation names, tool identifiers, method names, request classes, or other target-selection inputs used by local authorization policy.

If no such additional values are available, target_context is identical to aud.

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 value 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 value after JCS canonicalization. Implementations MUST NOT collapse an audience array to a string, reorder array elements, or otherwise rewrite the verified audience structure before signing or comparing target_context.

6.4. 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.

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

6.5. Actor Identity Representation

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

Each actor identifier 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 authenticated actor identity used during sender-constrained token presentation and token exchange;
* 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 derived from the validated sender-constrained presenter identity under local policy and mapped into the same ActorID representation the issuing Authorization Server uses for proof construction.

Readable-chain profiles carry arrays of ActorID values in ach. Privacy-preserving profiles bind ActorID values only inside step proofs and related evidence. In examples and formulas, [A,B] denotes a readable chain of ActorID values for actors A and B.

6.6. Artifact Typing

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

The following JWT typ values are defined:

* ach-step-proof+jwt
* ach-commitment+jwt
* ach-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.

6.7. 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.

6.8. Commitment Hash Algorithms

Committed-chain profiles use a named hash algorithm for construction of achc.

Commitment hash algorithm identifiers are values from the IANA Named Information Hash Algorithm Registry {{IANA.Hash.Algorithms}}.

Implementations supporting committed-chain profiles MUST implement sha-256. Implementations SHOULD implement sha-384.

Every achc object and every committed-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.

6.9. Commitment Function

Committed profiles use achc 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 achc 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;
* sid: the workflow identifier;
* achp: 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, sid, achp, halg, prev, step_hash}))).

Let prev_digest denote the prior committed-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 achc. 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.

The halg value MUST be a text string naming a hash algorithm from the IANA Named Information Hash Algorithm Registry {{IANA.Hash.Algorithms}}. This specification permits only sha-256 and sha-384 for achc. Hash algorithms with truncated outputs, including truncated sha-256 variants, MUST NOT be used. Other registry values MUST NOT be used with this specification unless a future Standards Track specification updates this document.

When a profile-defined proof input refers to a prior achc, the value incorporated into the proof input MUST be that prior commitment’s verified curr digest string, copied directly from the validated achc 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, sid, achp, prev_digest, step_proof_bytes, halg) = AS-signed commitment object over payload { ctx, iss, sid, achp, halg, prev = prev_digest, step_hash = b64url(Hash_halg(step_proof_bytes)), curr = b64url(Hash_halg(CanonicalEncode({ctx, iss, sid, achp, 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 sid and achp are the workflow and profile values being preserved for that workflow state.

6.10. Common Cryptographic Operations

The committed 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 achp value;
* sid be the stable workflow identifier;
* prev_state be either the returned base64url initial_chain_seed from bootstrap or the verified prior commitment digest string from achc.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;
* TC_next be the canonical target_context for the next hop, often just the next aud value 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.

Committed profiles instantiate the following proof-input template:

visible committed chain template:

{::nomarkdown} <sourcecode type="json"> Sign_N({ "ctx": ds(profile), "sid": sid, "prev": prev_state, "ach": visible_actor_chain_for_hop, "target_context": TC_next }) </sourcecode> {:/nomarkdown}

The domain-separation string ds is profile-specific:

* actor-chain-readable-committed-step-sig-v1 for Committed Chain with Full Disclosure;
* actor-chain-private-committed-step-sig-v1 for Committed Chain with No Chain Disclosure; and
* actor-chain-selectively-disclosed-committed-step-sig-v1 for Committed Chain with Subset Disclosure.

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

The profile-specific meaning of visible_actor_chain_for_hop is:

* for *Committed Chain with Full Disclosure*, the full readable chain for the hop after appending the authenticated current actor;
* for *Committed Chain with Subset Disclosure*, the exact inbound disclosed ach verified by the current actor, with that current actor appended; and
* for *Committed Chain with No Chain Disclosure*, the profile-defined actor-visible chain for the hop: bootstrap uses the singleton [A]; each non-bootstrap hop uses the exact pair [PresentingActor, CurrentActor].

For subset-disclosure committed operation, any readable ach disclosed to the next recipient MUST be derived from the proof-bound visible_actor_chain_for_hop, MUST be an ordered subsequence of it, and MUST include the current actor as its last element. For zero-disclosure operation, ordinary tokens omit the ach claim entirely even though the current actor still signs the profile-defined actor-visible chain for the hop.

7. Profile Selection and Session 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 achp equals that requested profile identifier or reject the request.

For a given accepted chain state identified by sid, achp is immutable. Any token exchange, cross-domain re-issuance, or Refresh-Exchange that would change achp for that accepted chain state MUST be

rejected. A current actor MUST reject any returned token whose achp 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 sid, not continuing an existing accepted chain state.

8. Common Validation Procedures

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

8.1. 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 (achp);
* presence and correct format of profile-required structural claims (ach and/or achc according to achp);
* if the recipient directly relies on achc as evidence, rather than relying on a locally trusted enclosing token issuer, validation of its typ header and JWS signature according to local policy and Appendix A;
* audience and target-context consistency according to local policy;
* expiry;
* sender constraint; and
* replay and freshness state.

8.2. Authorization Server Validation of Token Exchange

Unless a profile states otherwise, an Authorization Server processing a token exchange under this specification MUST verify:

* the inbound subject_token;
* the identity of the current actor;
* replay and freshness constraints;
* intended-recipient semantics for the inbound token, except when actor_chain_refresh=true or actor_chain_cross_domain=true; and
* authorization to act for the requested target context.

For token exchange, sender-constrained validation applies differently to the inbound token and to the current exchange request. The Authorization Server MUST validate the current actor's authentication and any applicable sender-constrained proof for the exchange request itself. It MUST validate the inbound subject_token as an inbound token under this specification, but it MUST NOT require the current actor to produce a fresh sender-constrained proof using the previous actor's private key solely to redeem that inbound subject_token at the token endpoint.

8.3. Current-Actor Validation of a Returned Token

Unless a profile states otherwise, a current actor validating a returned token from token exchange MUST verify the token signature, profile identifier, workflow identifier continuity, subject continuity, current-actor continuity, expiry, that the returned target context corresponds to what was requested for that operation according to local policy, and any profile-specific append-only or commitment checks before presenting that token to the next hop.

The returned sid MUST equal the workflow identifier already in progress, the returned sub claim MUST equal the inbound token's sub value for these base profiles, and the returned act claim MUST continue to identify the same current actor that performed the exchange. An unexpected change in achp is a profile-continuity failure and MUST cause rejection of the returned token.

9. Profiles

The profile selection table appears earlier in "Scope and Model". The sections below present the ordinary chain-extending profile flows first. Special preserve-state exchanges, metadata, and deeper enforcement details appear later so they do not interrupt the main profile story.

10. Asserted Chain with Full Disclosure Profile

10.1. Profile Identifier

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

10.2. Objective

The Asserted Chain with Full Disclosure profile extends token exchange by carrying a readable ach and requiring chain-continuity validation by both the current actor and the issuing Authorization Server at each hop.

10.3. Security Model

This profile provides hop-by-hop readable 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.

10.4. Bootstrap

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

* grant_type=client_credentials;
* actor_chain_profile=asserted-chain-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 set sub either to the authenticated client identity or to an explicitly requested and authorized delegating user identity. AS1 MUST then issue T_A containing at least:

* achp=asserted-chain-full
* sub
* act
* ach=[A]
* sid
* jti
* aud=B
* exp

10.5. 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 extract the verified ach and verify that its last actor is A.

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

10.6. 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=asserted-chain-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.

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

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

* achp=asserted-chain-full
* sub
* act
* ach=[A,B]
* sid
* jti
* aud=C
* exp

10.7. 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 T_B.ach is exactly the previously verified chain from T_A with B appended.

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

10.8. 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 extract the verified ach and use it for authorization decisions.

10.9. Security Result

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

11. Asserted Chain with Subset Disclosure Profile

11.1. Profile Identifier

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

11.2. Objective

This profile inherits the Asserted Chain with Full Disclosure profile and changes the readable 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.

11.3. Inheritance and Security Model

Except as modified below, all requirements of the Asserted Chain with Full Disclosure profile apply.

The disclosed ach seen by a recipient MUST be an ordered subsequence of the asserted chain state for that hop and MUST include the current actor as its last element.

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

This profile relies on the issuing Authorization Server for continuity of the carried-forward asserted subset chain state and for disclosure policy. Actors hidden from the readable chain at one hop are outside the guaranteed carried-forward state of this profile. Cross-domain re-issuance preserves only the verified disclosed subset chain state carried in the inbound token; it does not transfer any hidden full-chain state. An issuing Authorization Server MAY retain richer local audit records, including previously hidden actors; when such local records exist within one issuing Authorization Server, it SHOULD append the current actor to those local records for audit purposes. Such local records are outside the interoperable carried-forward state of this profile and MUST NOT be projected into returned

tokens unless disclosed by policy. Deployments that require hidden full-chain continuity across domains MUST use a committed profile or another trusted state-transfer mechanism. This profile does not provide the step-proof-based accountability or cumulative commitment state of the committed profiles.

11.4. Modified Bootstrap and Issuance

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

* grant_type=client_credentials;
* actor_chain_profile=asserted-chain-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 Asserted Chain with Full Disclosure profile would issue a token containing a readable ach, this profile MUST instead issue a recipient-specific disclosed ach for the intended recipient.

11.5. Modified Hop Processing and Validation

Where the Asserted Chain with Full Disclosure profile requires presentation or validation of a readable ach, this profile instead requires presentation and validation of the disclosed subset chain.

11.6. 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=asserted-chain-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.

For this profile, the issuing Authorization Server MUST derive the next-hop asserted chain state from the inbound readable ach exactly as verified for the current actor. In this profile, that verified inbound readable ach is the authoritative carried-forward asserted subset chain state for the next hop; any actors hidden before the current hop are outside the guaranteed carried-forward state of this profile. The issuing Authorization Server MUST append the current actor to that verified inbound readable chain state and then derive the recipient-specific disclosed subset ach for the returned token by

dropping zero or more prior actors from that resulting chain state. It MUST NOT insert, reorder, or alter actor identities, and it MUST NOT drop the current actor.

The current recipient and the current actor MUST verify that the last disclosed actor is the presenting actor for the inbound token or, for a returned token, the current actor that requested exchange.

Unlike the Asserted Chain with 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.

11.7. Next-Hop Authorization

A recipient MAY use the verified disclosed ach for authorization decisions.

A recipient MUST use only the disclosed ach for authorization and MUST treat undisclosed prior actors as unavailable.

11.8. 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.

12. Common Processing for the Committed Branch

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

* Committed Chain with Subset Disclosure;
* Committed Chain with Full Disclosure; and
* Committed Chain with No Chain Disclosure.

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

12.1. Common Parameters

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

Profile	achp value	init_label(profile)	Step-proof domain-separation string	Proof-bound actor-visible chain for hop	Readable ach in ordinary tokens
Committed Chain with Subset Disclosure	committed-chain-subset	actor-chain-selectively-disclosed-committed-init	actor-chain-selectively-disclosed-committed-step-sig-v1	Exact inbound disclosed ach verified by the current actor, with that actor appended	Ordered subsequence of the proof-bound actor-visible chain, ending in the current actor
Committed Chain with Full Disclosure	committed-chain-full	actor-chain-readable-committed-init	actor-chain-readable-committed-step-sig-v1	Full readable chain for the hop after appending the current actor	Exact proof-bound actor-visible chain
Committed Chain with No Chain Disclosure	committed-chain-no-chain	actor-chain-private-committed-init	actor-chain-private-committed-step-sig-v1	At bootstrap, singleton [A]; thereafter exact pair [PresentingActor, CurrentActor]	Not present

Table 2

The step-proof domain-separation strings above are intentionally distinct. Although the committed-branch step-proof payload members are structurally aligned, the signed payload does not carry achp or another explicit proof-mode identifier, and the profile-specific interpretation of ach therefore remains bound by ds. The bootstrap-init labels and domain-separation strings are stable protocol constants and are not required to track later editorial changes to the human-readable profile names.

12.2. Common Bootstrap Context Request

Authorization Servers supporting committed profiles SHOULD publish an actor_chain_bootstrap_endpoint metadata value naming the endpoint used to mint bootstrap context for initial actors. Authorization Servers supporting this bootstrap flow SHOULD also advertise urn:ietf:params:oauth:grant-type:actor-chain-bootstrap in the standard OAuth grant_types_supported metadata.

At workflow start, actor A MUST send an authenticated HTTPS POST to that endpoint using application/x-www-form-urlencoded with:

* actor_chain_profile set to one of the committed profile identifiers above;
* audience=B; and
* any other local inputs needed to derive the intended target_context, such as a resource identifier, tool name, or operation class when local policy distinguishes among them.

From those inputs, the Authorization Server MUST derive the exact canonical bootstrap-authorized target_context for the first hop. The Authorization Server MUST select halg for the workflow according to local policy and the supported values advertised in Authorization Server metadata.

The Authorization Server MUST generate:

* sid;
* initial_chain_seed; and
* actor_chain_bootstrap_context.

The halg value in the bootstrap context MUST be either sha-256 or sha-384 and MUST remain fixed for the lifetime of the workflow instance.

Let init_label(profile) denote the profile-specific bootstrap init label from the table above. The Authorization Server MUST derive raw bootstrap-seed bytes as:

{::nomarkdown} <sourcecode type="text">
Hash_halg(CanonicalEncode([init_label(profile), sid])) </sourcecode>
{:/nomarkdown}

For this bootstrap-seed derivation, CanonicalEncode is JCS {{!RFC8785}} applied to the two-element ordered JSON array [init_label(profile), sid]. The initial_chain_seed value carried on the wire is the base64url encoding of those raw bootstrap-seed bytes.

The Authorization Server MUST return a bootstrap response containing at least:

* actor_chain_bootstrap_context;
* sid;
* halg;
* initial_chain_seed (the base64url-encoded bootstrap seed string);
* target_context=TC_B, where TC_B is the exact canonical bootstrap-authorized target context for the first hop;
* aud corresponding to that exact canonical bootstrap-authorized target_context=TC_B; and
* a short expiry.

The actor_chain_bootstrap_context value is an opaque single-use handle. The Authorization Server MUST bind that handle to bootstrap state containing at least:

* the selected committed profile;
* sid;
* halg;
* the returned base64url initial_chain_seed;
* the exact canonical target_context=TC_B;
* the corresponding aud;
* the expiry; and
* the requesting actor or authenticated client according to local policy.

The handle MUST be single use and MUST be rejected after expiry or successful use at the token endpoint, except that an exact replay of a previously accepted bootstrap token request using the same handle and the same compact JWS step proof MUST be honored as an idempotent retry within a short retention window sufficient for ordinary transport retries. For such an idempotent retry, the Authorization Server MUST return the previously accepted bootstrap successor state, or an equivalent token representing that same accepted state, and MUST NOT treat the retry as a fresh second use of the handle. The bound state MUST be sufficient for the Authorization Server to reconstruct exactly the same canonical target_context value that the actor is expected to sign at bootstrap. During that retry-retention window, the Authorization Server SHOULD retain the exact previously issued bootstrap response or otherwise ensure that any retried response carries the same accepted chain state. Recomputing a retried response with probabilistic signatures can change wire bytes even when the decoded accepted state is equivalent.

12.3. Common Initial Actor Step Proof and Bootstrap Issuance

At bootstrap, the initial actor A uses the singleton actor-visible chain [A]. Let TC_B denote the exact canonical target_context value returned in the bootstrap response and bound to the bootstrap context for next recipient B. Let ds(profile) denote the profile-specific domain-separation string from the table above for committed step proofs. A MUST compute:

{::nomarkdown} <sourcecode type="json"> chain_sig_A = Sign_A({ "ctx": ds(profile), "sid": sid, "prev": initial_chain_seed, "ach": [A], "target_context": TC_B }) </sourcecode> {:/nomarkdown}

using canonical encoding. If A retries the same bootstrap hop after an uncertain transport failure, A MUST reuse the same compact JWS step proof rather than regenerating a different proof for that same attempted successor state.

A MUST submit an OAuth token request to the token endpoint using grant_type=urn:ietf:params:oauth:grant-type:actor-chain-bootstrap and containing at least:

* grant_type=urn:ietf:params:oauth:grant-type:actor-chain-bootstrap;
* actor_chain_profile set to the selected committed profile;
* actor_chain_step_proof=chain_sig_A; and
* actor_chain_bootstrap_context set to the bootstrap handle previously returned by actor_chain_bootstrap_endpoint.

Because the exact bootstrap-authorized target_context is already returned by the bootstrap endpoint and bound to the bootstrap context handle, the bootstrap token request need not repeat audience, resource, or other targeting parameters. If the client does repeat such targeting parameters, they MUST be semantically equivalent to the bound bootstrap-authorized target_context and MUST NOT broaden it.

The Authorization Server MUST verify:

* grant_type=urn:ietf:params:oauth:grant-type:actor-chain-bootstrap;
* the selected committed profile matches the profile bound to the bootstrap state;
* the submitted actor_chain_bootstrap_context handle and the bound bootstrap state;
* the identity of A;
* that the submitted proof’s JWS protected header contains typ=ach-step-proof+jwt;
* the validity of chain_sig_A;

* that the submitted proof binds the same sid as the bootstrap state;
* that the submitted proof uses ctx=ds(profile) for the selected profile;
* that the submitted proof binds prev=initial_chain_seed from the bootstrap response;
* that the submitted proof binds the exact singleton actor-visible chain [A];
* that the submitted proof binds the exact canonical bootstrap-authorized target_context=TC_B; and
* if the request repeats targeting parameters, that they are semantically equivalent to the bound bootstrap-authorized target_context.

Before issuing T_A, the Authorization Server MUST establish the workflow subject according to local policy. At bootstrap under these base profiles, the Authorization Server MUST set sub either to the authenticated client identity or to an explicitly requested and authorized delegating user identity. The resulting workflow subject is the one later same-workflow exchanges preserve.

If verification succeeds, the Authorization Server MUST compute:

{::nomarkdown} <sourcecode type="text"> achc = Commit_AS(as_issuer_id, sid, profile, initial_chain_seed, chain_sig_A, halg) </sourcecode> {:/nomarkdown}

The Authorization Server MUST use the same profile and sid values that it just verified.

and issue T_A containing at least:

* achp equal to the selected committed profile identifier;
* sub;
* act;
* achc;
* sid;
* jti;
* aud corresponding to that exact canonical bootstrap-authorized target_context=TC_B;
* exp; and
* any profile-defined readable ach.

12.4. Common Hop Processing

When an actor presents an inbound token under one of the committed profiles, the receiving actor MUST verify:

* token signature;
* issuer trust;
* profile identifier (achp);
* audience;
* expiry;
* sender constraint; and
* replay and freshness state.

The receiving actor MUST extract sid and achc, and MUST then apply the profile-specific continuity checks to determine the presenter continuity inputs and, where applicable, the exact inbound actor-visible chain that it is allowed to extend.

12.5. Common Token Exchange

To call the next recipient, the current actor N MUST set profile to the immutable achp value extracted from the inbound token and MUST set prior_commitment_digest to the verified curr value extracted from the inbound token’s achc. The current actor and the issuing Authorization Server MUST preserve that profile value for the exchange.

The current actor N MUST construct the exact profile-defined visible_hop_N and MUST compute:

{::nomarkdown} <sourcecode type="json"> chain_sig_N = Sign_N({ "ctx": ds(profile), "sid": sid, "prev": prior_commitment_digest, "ach": visible_hop_N, "target_context": TC_next }) </sourcecode> {:/nomarkdown}

using canonical encoding. If N retries the same hop after an uncertain transport failure, N MUST reuse the same compact JWS step proof rather than regenerating a different proof for that same attempted successor state.

The current actor N MUST submit to the issuing Authorization Server:

* grant_type=urn:ietf:params:oauth:grant-type:token-exchange;
* actor_chain_profile set to that same immutable committed 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=chain_sig_N; and
* the requested OAuth targeting parameters (audience, resource, or both as needed by local policy) sufficient to identify TC_next.

The Authorization Server MUST verify:

* *Standard OAuth and token validation:*

- the inbound token signature, issuer trust, and expiry;
- the identity of N;
- replay and freshness constraints;
- sender constraint on the inbound token as applicable;
- that the requested actor_chain_profile matches the immutable achp of the inbound token;
- that N was an intended recipient of the inbound subject_token, except when actor_chain_refresh=true or actor_chain_cross_domain=true; and
- that N is authorized to act for the requested target context.

* *Committed-profile proof and commitment validation:*
  - that the inbound achc object is a valid commitment JWS under Appendix A, including JWS signature validation and a locally permitted halg, before the Authorization Server relies on any extracted curr or halg value;
  - that the validated inbound achc object’s sid and achp exactly match the inbound top-level token’s sid and achp;
  - that the Authorization Server extracts the inbound halg value from that validated achc object before computing any new commitment;
  - that the submitted proof’s JWS protected header contains typ=ach-step-proof+jwt;
  - that the Authorization Server decodes the submitted proof payload to extract target_context, verifies that the extracted value is semantically consistent with the requested OAuth targeting parameters and local policy, and uses that extracted canonical value as TC_next when validating the proof;
  - that the submitted proof uses ctx=ds(profile) for the selected profile;
  - that the submitted proof binds the same sid;
  - that the submitted proof binds the same prior commitment;
  - that the submitted proof binds the reconstructed exact visible_hop_N; and
  - that the submitted proof binds the requested target_context=TC_next.

If verification succeeds, the Authorization Server MUST compute:

{::nomarkdown} <sourcecode type="text"> achc = Commit_AS(as_issuer_id, sid, profile, prior_commitment_digest, chain_sig_N, halg) </sourcecode> {:/nomarkdown}

The Authorization Server MUST use the same verified profile, sid, and halg values when computing the new commitment object.

and issue T_N containing at least:

* achp equal to the selected committed profile identifier;

* sub;
* act;
* achc;
* sid;
* jti;
* aud corresponding to the requested and verified TC_next;
* exp; and
* any profile-defined readable ach.

Under the base profiles defined by this document, the Authorization Server MUST preserve the inbound token's sub claim in T_N. A same-workflow subject transition is outside these base profiles and MUST start a new workflow instance with a new sid.

12.6. Common Returned-Token Validation

Upon receipt of the returned token for hop N, the current actor N MUST verify the token signature, profile fields, workflow identifier continuity, and current-actor continuity.

The current actor N MUST validate the returned achc according to Appendix A and MUST verify that its decoded payload is bound to the exact step proof submitted for that hop, including the expected iss value of the issuing Authorization Server for that commitment object, the expected sid, achp, halg, prev, and step_hash values derived from prior_commitment_digest and chain_sig_N. The returned top-level token sid MUST equal the expected workflow sid, the returned top-level sub claim MUST equal the expected inbound sub value for these base profiles, and the returned top-level act claim MUST continue to identify actor N.

Any additional profile-specific readable-chain or disclosure checks are defined in the profile sections below.

13. Committed Chain with Subset Disclosure Profile

13.1. Profile Identifier

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

13.2. Objective

This profile is the readable subset-disclosure committed case. The issuing Authorization Server MAY disclose only a recipient-specific ordered subset of the proof-bound actor-visible chain, while the current actor signs the exact actor-visible chain that it was allowed to verify and extend for the hop.

13.3. Security Model

This profile inherits all requirements of the common committed-processing section.

The disclosed ach seen by a recipient MUST be an ordered subsequence of the proof-bound actor-visible chain for that hop and MUST include the current actor as its last element.

Step proofs and achc 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.

This profile preserves current-actor continuity and cumulative committed state for the chain state that the current actor was allowed to verify and extend. It does not require a current actor to learn hidden prior actors in order to continue the workflow.

13.4. Profile-Specific Hop Construction and Validation

For a current actor N, let ach_in be the exact disclosed inbound ach that N verified from the inbound token. N MUST verify that the last actor in ach_in is the verified presenting actor of the inbound token.

The exact proof-bound actor-visible chain for the hop is:

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

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

The Authorization Server MUST verify that the submitted visible chain equals the exact inbound disclosed chain previously verified by N, with N appended.

Any readable ach disclosed to the next recipient MUST be derived from the exact proof-bound actor-visible chain for that hop, MUST be an ordered subsequence of it, and MUST end in N.

When validating a returned token, the current actor N MUST additionally verify:

* that the returned readable ach, or an equivalent presentation-derived ach when an agreed optional encoding is in use, has N as its last actor; and
* that the disclosed chain is an ordered subsequence of the exact proof-bound actor-visible chain that N signed for that hop.

A recipient MAY use the verified disclosed ach for authorization decisions, but MUST use only the disclosed subset and MUST treat undisclosed prior actors as unavailable.

13.5. Attack Handling

Different recipients MAY receive different valid disclosed subsets derived from the same proof-bound 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 proof-bound 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 proof-bound actor-visible chain for the relevant hop.

13.6. Security Result

This profile preserves current-actor continuity, cumulative committed state, and recipient-specific limited readable authorization while avoiding disclosure of hidden prior actors to an acting intermediary that was not permitted to see them.

14. Committed Chain with Full Disclosure Profile

14.1. Profile Identifier

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

14.2. Objective

The Committed Chain with Full Disclosure profile is the full-disclosure readable committed case. It preserves a readable ach for every actor and downstream recipient while adding per-hop actor-signed step proofs and cumulative committed state.

14.3. Security Model

This profile inherits all requirements of the common committed-processing section and specializes that common processing to the full-disclosure readable case.

This profile preserves readable chain-based authorization and provides stronger accountability and non-repudiation than the Asserted Chain with 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.

14.4. Profile-Specific Hop Construction and Validation

For a current actor N, let ach_in be the full readable ach verified from the inbound token. N MUST verify that the last actor in ach_in is the verified presenting actor of the inbound token.

The exact proof-bound actor-visible chain for the hop is the full readable append-only chain:

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

The Authorization Server MUST issue the full visible_hop_N as the readable ach in the returned token.

When validating a returned token, the current actor N MUST additionally verify that the returned readable ach is exactly the full proof-bound actor-visible chain that N signed for that hop.

A recipient MUST use the full readable ach for authorization decisions.

14.5. 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 committed state and sid.

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

* an earlier token showing the prior chain state; and
* the corresponding committed 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.

14.6. Security Result

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

15. Committed Chain with No Chain Disclosure Profile

15.1. Profile Identifier

The profile identifier string for this profile is committed-chain-no-chain. It is used as the actor_chain_profile token request parameter value and as the achp token claim value.

15.2. Objective

This profile is the no-chain-disclosure committed case. It removes the ach claim from ordinary tokens, leaving only cumulative committed state and the verified presenting actor visible at the next hop.

15.3. Security Model

This profile inherits all requirements of the common committed-processing section and specializes that common processing to the no-chain-disclosure case.

This profile preserves sender-constrained current-actor continuity and cumulative committed state, but recipients of ordinary tokens see only an opaque commitment object and not a readable prior-actor path.

This profile does not preserve readable prior-actor authorization at downstream hops. Prior-actor integrity is ordinarily verifiable only by the issuing Authorization Server or an auditor with access to retained step proofs or exchange records.

15.4. Profile-Specific Hop Construction and Validation

For a current actor N, let P be the verified presenting actor of the inbound token. Because ordinary tokens omit the ach claim, the current actor MUST determine P from either a validated act claim or a validated sender-constrained presenter identity bound by local trust policy to the same ActorID representation used for proof construction. For non-bootstrap hops, the exact proof-bound actor-visible chain for the hop is:

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

At bootstrap, this profile instead uses the common committed-bootstrap rule, under which the initial actor signs the singleton actor-visible chain [A]. For non-bootstrap hops, the Authorization Server MUST verify that the submitted visible chain equals exactly [PresentingActor, CurrentActor] for that hop.

Tokens issued under this profile MUST contain achp, sub, act, achc, sid, jti, aud, and exp, and MUST NOT contain an ach claim.

When validating a returned token, the current actor N MUST additionally verify that the returned token does not contain an ach claim.

A downstream recipient MUST use the verified presenting actor, not prior actors, for authorization decisions.

A downstream recipient MUST NOT infer the identities or number of prior actors from achc alone.

15.5. Attack Handling

The committed-profile attack-handling properties still apply, but omission, insertion, or reordering of prior actors will ordinarily be detected only by the issuing Authorization Server or by later audit, not inline by downstream recipients receiving ordinary tokens.

15.6. Security Result

This profile reduces ordinary-token disclosure and token size while preserving per-hop continuation proofs at the acting hop and cumulative committed state across hops.

16. 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.

16.1. Cross-Domain Re-Issuance

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. It MUST NOT issue a local token whose target context is broader than, or semantically unrelated to, the audience 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.

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 chain-state structure;
* preserve achp;
* preserve sid;
* preserve sub;
* preserve ach, if present, exactly as verified for the current actor, without broadening, narrowing, or otherwise rewriting the verified disclosed or readable chain state;
* preserve achc, if present, exactly as verified;
* continue to represent the same current actor; and
* NOT append the next recipient.

The cross-domain Authorization Server MUST validate any preserved achc JWS before carrying it forward. That validation includes using the preserved commitment object's own signer identity to resolve the original Authorization Server's verification key material. If the inbound act claim omitted iss, the re-issuing Authorization Server MUST preserve the same ActorID semantics by emitting an explicit act.iss equal to the inbound token's issuer together with the same act.sub value, rather than relying on the new local token issuer as an implicit namespace. Because the token subject is interpreted for ActorID purposes as { "iss": token.iss, "sub": token.sub }, the re-issuing Authorization Server MUST ensure that preserving the inbound sub value under the new enclosing token issuer would still denote the same subject under local federation or identifier-mapping policy. If preserving the same sub bytes under the new issuer would change subject semantics, the re-issuing Authorization Server MUST reject cross-domain re-issuance rather than silently reinterpret that subject under the new local issuer namespace. 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. It MUST NOT alter the verified preserved chain state. If cross-domain re-issuance narrows or locally rewrites the target context, retained step proofs and preserved achc 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 achc was validated and carried forward unchanged. Such a recipient need not independently validate a foreign Authorization Server's JWS signature on the preserved achc unless local policy or audit requires it.

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 achp, sid, sub, ach, and achc, are unchanged from the inbound token except where this specification explicitly permits cross-domain re-issuance changes such as local jti, local exp, token format or envelope, or approved local trust and policy claims.

16.2. 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.

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 represented by that token; 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.

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, readable or disclosed chain state visible to the current actor, committed chain state, or target context. The requested target context MUST be identical to, or narrower than, the target context already represented by the inbound token according to local policy.

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 achp;
* verify sender constraint and intended-recipient semantics as applicable;

* verify that the request does not append the chain, alter preserved chain state, or broaden target context; and
* issue a replacement token with a new jti and refreshed exp.

For Refresh-Exchange, the Authorization Server MUST preserve sid, achp, sub, ach, and achc, 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 achc continue to reflect the target context that was bound during the original chain-extending hop, not the narrower refreshed token audience.

A Refresh-Exchange MAY rotate the sender-constrained presentation key only if the actor provides a key-transition proof that binds the new presentation key to the same sid and ActorID under local policy, and the Authorization Server verifies and records that proof. Such proof MAY be satisfied by continuity mechanisms provided by the sender-constrained binding in use or by another locally trusted proof-of-possession transition method. Otherwise, the sender-constrained key binding MUST be preserved. Historical step proofs remain bound to the keys used when those proofs were created and MUST be verified against those historical bindings, not against a later rotated key.

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 a sender-constrained key transition occurred, recipients still validate historical step proofs against the keys bound when those proofs were produced and rely on Authorization Server records or other retained evidence for the key-transition event itself.

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 achp, sid, sub, ach, and achc, are unchanged from the inbound token except where this specification explicitly permits refresh-specific changes such as jti, exp, or approved sender-constrained key-transition metadata.

17. 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 workflow identifier, the identified inbound hop state, presenting actor, recipient, target context, and

the acknowledged inbound token instance via inbound_jti. For asserted-chain profiles, that inbound hop state is the verified readable ach from the inbound token. For committed-chain profiles, that inbound hop state is the verified prior commitment digest extracted from the inbound token's achc.

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.

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.

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 target_context, it MUST include an operation-unique request identifier inside target_context or by a profile-defined extension that is covered by the recipient's hop_ack JWT signature.

The acknowledgment payload MUST include at least:

* ctx = actor-chain-hop-ack-v1;
* sid;
* achp;
* jti, a unique identifier for the hop_ack JWT itself;
* inbound_jti, copied from the acknowledged inbound token;

* presenting actor ActorID;
* recipient ActorID;
* target_context;
* exp, a short-lived JWT NumericDate;
* for asserted-chain profiles, inbound readable ach;
* for committed-chain profiles, prev, the verified prior commitment digest copied directly from the inbound token's achc.curr; and
* ack, whose value MUST be accepted.

A hop_ack MUST be signed by the recipient using JWS.

17.1. 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=ach-hop-ack+jwt;
* ctx=actor-chain-hop-ack-v1;
* sid equals the workflow identifier of the inbound token for which acknowledgment is being evaluated;
* achp 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;
* presenter equals the presenting actor (that is, the actor who performed token exchange for the acknowledged hop) represented by that inbound token;
* 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;
* exp has not expired;
* for asserted-chain profiles, the carried ach equals the inbound readable ach for the acknowledged hop; and
* for committed-chain profiles, prev equals the verified prior commitment digest copied from the inbound token's achc.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 achc 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.

18. 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.

18.1. Sender Constraint

A token issued under any profile in this document MUST be sender-constrained to the actor represented by that token.

A recipient or Authorization Server validating such a token MUST verify the applicable sender-constrained proof before accepting the token.

Failure of sender-constrained validation MUST cause rejection.

18.2. Actor and Recipient Proof Keys

For committed-chain profiles and for hop_ack, any signature used as a profile-defined proof MUST be generated with an asymmetric key bound to the authenticated actor or recipient identity by local trust policy.

For a committed-profile step proof, the ActorID represented in the proof, the key used to sign the proof, and the sender-constrained key used to present the corresponding token MUST all be bound to the same actor identity. When the same key is not reused for both functions, the Authorization Server MUST validate an explicit local binding between the proof-signing key and the sender-constrained presentation key before accepting the proof.

For hop_ack, the recipient ActorID, the key used to sign the acknowledgment, and any sender-constrained key used by that recipient for the protected interaction MUST likewise be bound to the same recipient identity.

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

A deployment SHOULD reuse the same asymmetric key material used for sender-constrained token presentation, or another asymmetric key that is cryptographically bound to the same actor identity.

18.3. 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.

18.4. 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 (sid, 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.

19. Authorization Server Metadata

An Authorization Server supporting this specification SHOULD publish metadata describing 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 committed-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 achp 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; and
* 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.

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

20. 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, or unsupported profile bindings
invalid_target	The requested audience, target context, or recipient is not permitted or not supported
invalid_grant	The subject_token fails validation, sender-constrained verification fails, the intended-recipient check fails, continuity fails at token exchange, replay or freshness checks fail, actor_chain_step_proof verification fails, 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.

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.

21. 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.

21.1. Sender-Constrained Enforcement is Foundational

The security of these profiles depends strongly on sender-constrained token enforcement. If a token can be replayed by an attacker that is not the bound actor, continuity checks become materially weaker.

21.2. 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.

21.3. Readable Chain Does Not Prevent Payload Abuse

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

21.4. Committed Profiles Depend on Proof Retention

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

Authorization Servers supporting committed profiles SHOULD retain proof state, exchange records, 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.

21.5. Committed Chain with No Chain Disclosure Removes Inline Prior-Actor Visibility

Recipients using the Committed Chain with No Chain Disclosure profile can validate the presenting actor and preserved commitment continuity, but cannot authorize based on readable prior-actor membership or order from the ordinary token alone.

21.6. Subset-Disclosure Profiles Reveal Only a Verified Subset

Recipients using the Asserted Chain with Subset Disclosure profile or the Committed Chain with Subset Disclosure profile can authorize based only on the disclosed ach 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.

For the Committed Chain with Subset Disclosure profile, the disclosed subset to a recipient MUST be derived from the actor-signed actor-visible chain for that hop. A malicious or compromised issuing

Authorization Server can still attempt to issue a subset inconsistent with that proof-bound chain, so retained step proofs and exchange records remain important for later verification and audit.

This specification intentionally avoids requiring an acting intermediary to learn a hidden full-chain prefix merely to continue the workflow. Deployments that need later reconstruction of a hidden prefix beyond what each actor signed MUST rely on retained Authorization Server state and audit records.

21.7. 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 chain state unchanged. For committed subset-disclosure operation, this includes the chain state visible to the current actor and any disclosed subset carried forward for the next hop. Any such re-issuance MUST continue to represent the current actor and MUST NOT append the recipient.

21.8. Residual Risks and Out-of-Scope Behavior

These profiles do not by themselves make application payloads safe or policy-conformant, and they do not by themselves prevent confused-deputy behavior.

The asserted profiles rely on the issuing Authorization Server for the asserted chain state that they carry forward. In the subset-disclosure asserted profile, a current actor or downstream recipient validates only the disclosed subset it receives and does not independently validate undisclosed prior actors.

The committed subset-disclosure and no-chain-disclosure profiles reduce what downstream recipients can authorize inline from ordinary tokens alone. Hidden-prefix reconstruction or later proof of what an actor saw can depend on retained Authorization Server state, exchange records, and proofs.

The committed full-disclosure profile does not by itself provide privacy minimization, and the committed profiles do not standardize merge or branch-selection semantics across parallel work that shares earlier workflow history. Deployments that need interoperable shared-root branching behavior MUST use an extension or companion protocol that defines it explicitly. Deployments MAY still correlate related linear flows out of band by local policy.

21.9. 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.

21.10. Chain Depth

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

21.11. 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.

22. 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 readable prior-actor authorization SHOULD consider the Committed Chain with No Chain Disclosure profile or one of the subset-disclosure profiles.

The stable workflow identifier sid correlates all accepted hops within one workflow instance. Accordingly, sid 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;
* omission of auxiliary claims unless receiving policy depends on them; and

* the Asserted Chain with Subset Disclosure profile or the Committed Chain with Subset Disclosure profile when partial readable-chain disclosure is sufficient.

22.1. Subset Disclosure and Optional Encodings

This specification defines subset-disclosure semantics for the Asserted Chain with Subset Disclosure profile and the Committed Chain with Subset Disclosure profile. In both profiles, the recipient-visible ach is a profile-defined ordered subsequence of the actor chain for that hop, carried as an ordinary readable ach claim containing only the disclosed subset.

This representation is the interoperable base-wire format for the subset profiles.

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

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

* for the Asserted Chain with Subset Disclosure profile, the disclosed ach MUST be an ordered subsequence of the asserted chain state for that hop;
* for the Committed Chain with Subset Disclosure profile, the disclosed ach MUST be an ordered subsequence of the proof-bound actor-visible chain for that hop;
* in both subset profiles, the disclosed ach MUST include the current actor as its last element;
* if the selected profile uses step proofs or chain commitments, those artifacts remain bound to the proof-bound 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;
* an encoding used with a Full Disclosure profile MUST reveal the full readable chain required by that profile to the recipient before authorization; and

* an encoding used with the Committed Chain with No Chain Disclosure profile MUST NOT expose hidden actor entries to recipients of ordinary tokens merely in digested or hidden or selectively revealable form.

23. Appendix A. JWT Binding (Normative)

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

23.1. 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.

The ach claim, when present in a JWT, is a JSON array of ActorID objects.

23.2. Step Proof in JWT

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

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

* ctx;
* sid;
* prev;
* target_context; and
* ach.

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 Committed Branch. The prev member MUST be the base64url string value of the prior commitment digest or bootstrap seed, copied directly from the verified inbound achc.curr or bootstrap response,

respectively. The ach member MUST be a JSON array of ActorID objects and denotes the profile-defined proof-bound actor-visible chain for the hop. For the Committed Chain with No Chain Disclosure profile, that proof-payload ach is internal proof input only and MUST NOT be copied into a readable token ach claim for ordinary tokens issued under that profile. The target_context member value MUST carry the verified aud value exactly. If aud is a string, target_context MAY be that same JSON string or a JSON object that includes an aud member with that same string. If aud is an array of strings, target_context MUST represent that array exactly, either as that same JSON array value or as a JSON object whose aud member is that exact array. Additional target-selection members used by local policy MAY be included only in the JSON-object form. Before any proof input is hashed or signed, target_context MUST be canonicalized using JCS exactly once as part of the enclosing payload object; verifiers MUST reproduce the same JCS bytes when validating the proof.

The JWS algorithm MUST be an asymmetric algorithm. The none algorithm MUST NOT be used. The JWS verification key MUST be bound to the same ActorID as the sender-constrained presentation key for the corresponding actor.

23.3. Receiver Acknowledgment in JWT

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

* ctx;
* sid;
* achp;
* jti;
* inbound_jti;
* exp;
* target_context;
* presenter;
* recipient;
* for asserted-chain profiles, ach;
* for committed-chain profiles, prev; and
* ack.

The ctx member value MUST equal actor-chain-hop-ack-v1. The presenter and recipient members MUST be ActorID objects. 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 exp member MUST be a JWT NumericDate and SHOULD be short-lived according to

local policy. The target_context member MUST follow the same representation rules defined for step proofs. For asserted-chain profiles, the ach member MUST carry the verified readable inbound ach value. For committed-chain profiles, the prev member MUST be copied directly from the verified prior commitment digest extracted from the inbound token's achc.curr. The JWS signer MUST be the recipient, and the verification key MUST be bound to the same recipient ActorID as any sender-constrained presentation key used for the protected interaction.

23.4. Commitment Object in JWT

The achc claim value MUST be a compact JWS string. The JWS protected header MUST contain typ=ach-commitment+jwt.

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

* ctx;
* iss;
* sid;
* achp;
* halg;
* prev;
* step_hash; and
* curr.

The ctx member value MUST equal actor-chain-commitment-v1. The iss member MUST identify the Authorization Server that signed the achc object. The halg member MUST be either sha-256 or sha-384. The members prev, step_hash, and curr MUST be the base64url encodings of raw hash bytes. The curr member is the current commitment digest for that accepted hop. Later chain-extending step proofs copy the verified inbound achc.curr value into their prev member, and committed-profile hop_ack copies that same inbound digest into its prev member for the acknowledged token.

The JWS payload signer MUST be the Authorization Server identified by iss. An Authorization Server that issues or re-issues a token using a preserved achc MUST validate that JWS signature, use iss to resolve the appropriate signer trust context when the commitment originated at another domain, and verify that halg is locally permitted before relying on the object. A current actor or downstream recipient that receives a token from a locally trusted issuing Authorization Server MAY rely on that enclosing token signature as attestation that any preserved foreign achc was validated, and need not independently validate a foreign Authorization Server's JWS signature on achc unless local policy or audit requires it. Where the verifier does validate the achc object itself, it MUST then validate that curr equals:

{::nomarkdown} <sourcecode type="text"> b64url(Hash_halg(JCS({ctx, iss, sid, achp, halg, prev, step_hash}))) </sourcecode> {:/nomarkdown}

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

24.1. Example 1: Asserted Chain with Full Disclosure in One Domain

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

1. A requests a token for B under the Asserted Chain with Full Disclosure profile.
2. AS1 issues T_A with ach=[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 ach=[A,B] and aud=C.
6. B validates that the returned 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 readable chain [A,B].

24.2. Example 2: Asserted Chain with Subset Disclosure

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

1. A requests a token for B under the Asserted Chain with Subset Disclosure profile.

2. AS1 issues T_A with a recipient-specific disclosed ach intended for B.
3. A calls B and presents T_A.
4. B validates the token and uses only the disclosed chain for authorization.
5. B exchanges T_A at AS1 for a token to C.
6. AS1 appends B to the inbound disclosed chain state it verified from T_A, applies disclosure policy for C, and issues T_B with a recipient-specific disclosed ach.
7. B presents T_B to C.
8. C validates the token, confirms that B is the last disclosed actor, and authorizes based only on the disclosed chain.

24.3. Example 3: Committed Chain with 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 ach=[A] and achc.
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, updates the commitment, and issues T_B with ach=[A,B] and aud=C.
5. Because C does not trust AS1 directly, B performs a second exchange at AS2.
6. AS2 preserves achp, sid, ach=[A,B], and achc, and issues a local token trusted by C that still represents B.
7. C validates the local token, sees the readable chain [A,B], and authorizes accordingly.

24.4. Example 4: Committed Chain with No Chain Disclosure

Assume A, B, and C use the Committed Chain with No Chain Disclosure profile.

1. A obtains bootstrap context, signs chain_sig_A over visible chain [A], and receives T_A with achc, but no ach claim.
2. A calls B with T_A.
3. B validates T_A, verifies that A is the presenter, constructs the profile-defined 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 achc, but no readable chain.
5. B presents T_B to C.
6. C validates the token and authorizes based on the verified presenting actor B and local policy. C MUST NOT infer prior-actor identity or count from the commitment alone.

24.5. Example 5: Committed Chain with Subset Disclosure

Assume A, B, and C use the Committed Chain with Subset Disclosure profile.

1. A obtains bootstrap context, signs chain_sig_A, and receives T_A with a recipient-specific disclosed ach and achc intended for B.
2. A calls B and presents T_A.
3. B validates the token and uses only the disclosed chain for authorization.
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 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 ach and updated achc.
6. B presents T_B to C.
7. C validates the token, confirms that B is the last disclosed actor, and authorizes based only on the disclosed chain.
8. If later audit is needed, the proof-bound actor-visible chain for the hop can be reconstructed from retained step proofs and exchange records.

25. Appendix C. Future Considerations (Informative)

25.1. Terminal Recipient Handling

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 committed state.

25.2. 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.

25.3. Subset Disclosure and Optional Encodings

This document now defines baseline Asserted Chain with Subset Disclosure and Committed Chain with 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 readable-chain disclosure to the recipient, while Committed Chain with No Chain Disclosure MUST NOT expose hidden actor entries to recipients of ordinary tokens merely as digests or selectively revealable placeholders.

25.4. Branching and Fan-Out

This specification defines linear per-step evidence and does not standardize merge or branch-selection semantics across multiple descendants that share earlier workflow history.

A future branching profile could add explicit branch identifiers or parent-child workflow correlation, for example by binding a branch sid to a parent sid, and could define tree-structured commitment verification, inclusion proofs, partial disclosure, and any 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.

Those semantics remain out of scope for this base specification.

25.5. Evidence Discovery and Governance Interoperability

Committed 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 sid, error handling, and retention expectations.

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

This document complements {{!RFC8693}} by defining chain-aware token-exchange profiles. It also 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 five profiles instead of one deployment mode so that implementations can choose among full readable chain-based authorization, trust-first partial disclosure, stronger committed-state accountability, recipient-specific committed partial disclosure, and reduced ordinary-token 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.

27. 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 sid, using a CSPRNG with at least 122 bits of entropy (for example, standard UUIDv4 or stronger generation)	Workflow Identifier (sid)	[ ]
Sender-constrained validation for every inbound token	Sender Constraint	[ ]
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 readable-chain profiles	Asserted Chain with Full Disclosure Profile; Committed Chain with Full Disclosure Profile	[ ]
Exact commitment verification for committed profiles	Commitment Function; Committed Chain with Full Disclosure Profile	[ ]
Proof-key binding between ActorID, proof signer, and sender-constrained presentation key	Actor and Recipient Proof Keys	[ ]

Non-broadening Refresh-Exchange processing, if supported	Refresh-Exchange	[ ]
Policy for when hop_ack is optional or required	Optional Receiver Acknowledgment Extension	[ ]
Privacy-preserving handling of logs and error messages	Error Handling; Privacy Considerations	[ ]

Table 4

28. 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.

28.1. 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}

28.2. 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}

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

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

29.1. 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"},
"aud": "https://api.example" (https://api.example"), "exp":
1760000000, "jti": "2b2b6f0d3f0f4d7a8c4c3c4f9e9b1a10", "sid":
"6cb5f0c14ab84718a69d96d31d95f3c4", "achp": "committed-chain-full",
"ach": [{"iss": "https://as.example" (https://as.example"), "sub":
"svc:orchestrator"}, {"iss": "https://as.example"
(https://as.example"), "sub": "svc:planner"}], "achc": "<compact JWS
string>" } </sourcecode> {:/nomarkdown}

29.2. Decoded achc JWS Example

Protected header:

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

Payload:

{::nomarkdown} <sourcecode type="json"> { "ctx": "actor-chain-
commitment-v1", "iss": "https://as.example" (https://as.example"),
"sid": "6cb5f0c14ab84718a69d96d31d95f3c4", "achp": "committed-chain-

full", "halg": "sha-256", "prev":
"SGlnaGx5SWxsdXN0cmF0aXZlUHJldkRpZ2VzdA", "step_hash":
"z7mq8c0u9b2C0X5Q2m4Y1q3r7n6s5t4u3v2w1x0y9z8", "curr":
"Vb8mR6b2vS5h6S8Y6j5X4r3w2q1p0n9m8l7k6j5h4g3" } </sourcecode>
{:/nomarkdown}

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

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

30. 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 be visible, but 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 prior-actor information than some deployments are willing to reveal.

31. 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 correctness of sender-constrained token enforcement, the trust relationship among participating Authorization Servers, and the availability of step proofs or exchange records where relied upon.

31.1. Assets

The protocol seeks to protect the following assets:

* continuity of the delegation path;
* integrity of prior-actor ordering and membership;
* continuity of the presenting actor;
* 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.

31.2. 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.

31.3. Assumptions

This specification assumes:

* verifiers can validate token signatures and issuer trust;
* sender-constrained enforcement is correctly implemented;
* the authenticated actor identity used in token exchange is bound 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.

31.4. Security Goals

The protocol aims to provide the following properties:

* in the Asserted Chain with 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 Asserted Chain with Subset Disclosure profile, ordinary tokens reveal only an ordered subset of prior actors selected by the Authorization Server, and authorization is limited to that disclosed subset;
* in the Committed Chain with 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 committed state, while ordinary tokens reveal only an ordered subset of that actor-visible chain selected by the Authorization Server;
* in the Committed Chain with Full Disclosure profile, the actor-visible chain equals the full readable chain at each hop, preserving full readable authorization while improving detectability, provability, and non-repudiation; and
* in the Committed Chain with No Chain Disclosure profile, ordinary tokens omit the ach claim while preserving presenting-actor continuity and cumulative committed state for later verification.

31.5. 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.

31.6. 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 the verified presenting actor, token subject, intended target, and local policy.

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

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

This specification provides different assurances depending on the selected profile:

* *Asserted Chain with Full Disclosure*: the issuing AS signature and chain assertion are the primary trust anchor.
* *Asserted Chain with Subset Disclosure*: the issuing AS signature, chain assertion, and disclosure policy are the primary trust anchors.
* *Committed Chain with Subset Disclosure*: the issuing Authorization Server reveals only a policy-selected ordered subset of the actor-visible chain to each recipient, while committed state and actor-signed proofs continue to support later verification of what each actor was shown and extended.
* *Committed Chain with Full Disclosure*: every actor and downstream recipient sees the full readable chain, so the actor-visible chain equals the full chain at each hop.
* *Committed Chain with No Chain Disclosure*: ordinary tokens omit the ach claim, while committed state and actor-signed proofs still bind the profile-defined actor-visible chain available at the acting hop and permit stronger accountability and later verification.

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

For committed 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 committed profiles, such records SHOULD include:

* prior token reference;
* authenticated actor identity;
* step proof reference or value;
* issued token reference;
* committed chain state;
* requested audience or target context; and
* timestamps.

For subset-disclosure profiles, retained records SHOULD also allow reconstruction of the proof-bound actor-visible chain for each hop and the disclosed subset issued for each recipient. Collecting all such accepted hop evidence for one sid, 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.

33. IANA Considerations

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

33.1. 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)
ach	Profile-defined ordered array of actor identity entries carried in the artifact.	IETF	[this document]
achc	Committed chain state binding accepted hop progression for the active profile.	IETF	[this document]
achp	Actor-chain profile identifier for the issued token.	IETF	[this document]

Table 5

33.2. Media Type Registration

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

Media Type	Media Subtype	Required Parameters	Optional Parameters	Encoding Considerations	Security Considerations	Interoperability Considerations	Published Specification	Author/Change Controller	Restrictions on Usage
application	ach-step-proof+jwt	N/A	N/A	binary	see [this document]	N/A	[this document]	IETF	IETF	COMMON	N/A
		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					
application	ach-commitment+jwt	N/A	N/A	binary	see [this document]	N/A	[this document]	IETF	IETF	COMMON	N/A
		OAuth 2.0 Token Exchange actor-chain commitments	N/A	Magic Number(s): N/A; File Extension(s): N/A;					

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

Table 6

33.3. 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 committed-profile bootstrap token request.	IETF	[this document]

Table 7

33.4. 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 committed-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

33.5. 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	OAuth token endpoint request	IETF	[this document]
actor_chain_bootstrap_context	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