Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: 17 November 2026

L. Zhu
S. Currie
Atlassian

16 May 2026

Sender-Constrained Delegation Handle for Asynchronous OAuth 2.0 Identity Chaining

draft-zhu-oauth-async-delegation-00

Abstract

This document defines an extension to OAuth 2.0 Token Exchange [RFC8693] and the OAuth Identity Chaining draft [I-D.ietf-oauth-identity-chaining] for delegated workflows that must continue after the end user has gone offline. When a chained access token expires while the end user is unavailable to re-authenticate (for example, to satisfy a multi-factor authentication or a touch-to-engage gesture), the acting client today has no standards-defined way to obtain a refreshed chained token without either widening privileges, breaking the identity chain, or impersonating the user as a service. This specification introduces the "Delegation Handle": a sender-constrained, audience-locked, scope-frozen JSON Web Token issued by the authorization server alongside a chained access token, that the original acting client MAY present back to the authorization server to refresh the chained access token within strict, server-side policy bounds and without re-prompting the end user.

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 17 November 2026.

Table of Contents

1. Introduction

1.1. The Problem

OAuth 2.0 Token Exchange [RFC8693] and the OAuth Identity Chaining draft [I-D.ietf-oauth-identity-chaining] together define how an acting client (the "actor"; for example, a backend service) can exchange an end user's access token for a new access token that carries both the user's identity (in "sub") and the actor's identity (in "act"). The resulting "chained access token" lets a downstream resource server authorize the request as the user, while the act claim preserves an auditable trail of the actor.

This works well while the end user remains online. When the chained access token nears expiry, the actor can simply re-exchange

a fresh user access token to obtain a new chained access token. If re-authentication is required, the user is present to satisfy it.

For an increasingly common class of "walk-away" asynchronous workloads, however, the user has finished their interactive authentication ceremony (such as multi-factor authentication or a touch-to-engage gesture) and then gone offline, leaving the acting client to continue the delegated work:

* Long-running background jobs initiated by a user and continued after the user signs off (data exports, large file uploads, compliance scans, multi-stage build pipelines).

* Scheduled tasks that the user authorized once and that must execute later on the user's behalf.

* Multi-step orchestrations (workflow engines, AI agent task queues) where each step takes minutes to hours and the user is not in the loop after the initial trigger.

* Asynchronous tool calls made by AI agents on the user's behalf that exceed the lifetime of the originally exchanged access token.

In all of these cases, when the chained access token expires, the acting client cannot obtain a new one through standard mechanisms: re-prompting the offline user is impossible, and unrecoverable when re-authentication requires a fresh user-presence factor (such as MFA or a touch-to-engage gesture) that no stored credential can substitute for. Falling back to the actor's own service-level credentials loses the user identity, typically carries broader privileges than the user holds, and creates a confused-deputy

condition at the downstream resource server -- exactly the least-privilege failure that identity chaining was meant to prevent.

1.2. Why Existing Mechanisms Are Insufficient

Several mechanisms have been considered for this problem; each is insufficient in a way this document addresses:

* OAuth 2.0 refresh tokens [RFC6749] are issued to clients to refresh their own access tokens. They are not defined for the identity-chaining case where the "subject" of refresh is a different principal (the end user) than the entity presenting refresh (the actor). RFC 6749 is silent on how refresh in a chained-identity exchange should validate the actor, scope, and audience constraints peculiar to identity chaining.

* RFC 8693 permits a "refresh_token" response parameter in the token-exchange response (Section 2.2.1 of [RFC8693]) but does not specify how such a refresh token should be shaped, validated, sender-constrained, or audience-locked for the chained-identity case. In particular, RFC 8693 does not require the refresh path to re-run the policy pipeline that

gated the original exchange, nor does it specify how an authorization server should revoke such refresh tokens when the acting client’s credentials are revoked.

* Using the actor’s own client-credentials access token after the chained token expires loses the user identity from the request, violates the least-privilege intent of identity chaining, and creates a confused-deputy condition at the resource server.

* Extending the chained access token’s lifetime to cover the worst-case asynchronous workload (hours to days) makes every stolen token a multi-hour or multi-day compromise window and defeats the purpose of short-lived access tokens.

* Storing the original user access token at the actor and re-presenting it after expiry does not work: the token is expired by construction in the long-running case.

* Storing the user’s primary credential or refresh token at the actor is not acceptable. The actor has not been delegated the user’s authentication factors and must not be able to act as the user without the original delegation having occurred.

The mechanism this document defines specifically targets the case where (a) the original delegation already occurred with the user present and authenticated, (b) the actor needs to continue the delegated work past the chained access token’s lifetime, and (c) the user is no longer available to re-authenticate.

1.3. Overview of the Solution

This document introduces a new token type, the "Delegation Handle" (or "handle"), issued by the authorization server alongside a chained access token in a token-exchange response.

The handle is a JWT [RFC7519] with claims that bind it tightly to the original delegation:

* Its "sub" is the end user, preserving the identity chain across refresh.

* Its "act.sub" (Section 4.1 of [RFC8693]) is the original acting client, sender-constraining it to that client.

* Its "aud" is the acting client itself (not a downstream resource server), so the handle is unusable as a downstream access token

and can only be presented back to the authorization server.

* A new "delegated_aud" claim freezes the downstream audience for which refreshed chained access tokens may be issued.

* The "scope" claim caps the maximum scope; refresh may narrow but never widen.

* A new "refreshes_remaining" claim bounds the number of refreshes the handle authorizes.

* The handle has its own "exp" independent of the chained access token's "exp"; refresh stops at the earlier of either expiry, the refresh cap, the end of the user session, or explicit revocation.

The acting client presents the handle to the authorization server's token-exchange endpoint, using the existing RFC 8693 grant, with "subject_token_type" set to a new value identifying it as a delegation handle. The authorization server re-runs its full identity-chaining validation pipeline (policy, scope narrowing, user-permission check, session validity) and, if all checks pass, issues a new chained access token (and optionally a new handle with the refresh counter decremented).

No new authentication primitive is introduced. The handle's security reduces to the strength of the acting client's existing credential (for example, "private_key_jwt" client authentication per Section 9 of [OpenID.Core], DPoP [RFC9449], or mutual TLS [RFC8705]) plus the JWT signature on the handle itself.

1.4. Non-Goals

This document does not:

* Provide a means for an acting client to act on behalf of a user who never delegated to it. The handle can only be issued as part of, or as an extension of, an existing chained delegation.

* Permit the handle to be presented to anyone other than the issuing authorization server. Resource servers MUST reject any token they receive whose "aud" identifies the calling client (this is already implied by standard JWT-access-token verification [RFC9068]).

* Permit widening of scope, change of downstream audience, or

retargeting to a different downstream resource during refresh. The handle is monotonically narrowing.

* Replace user re-authentication for high-assurance operations. Authorization-server policy MAY (and SHOULD, for sensitive audiences) require fresh user authentication context for certain audiences or scopes; refresh of those tokens via the handle will fail.

* Define a federation primitive across trust domains. The handle is issued and consumed within a single authorization server's trust domain.

2. Conventions and 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 uses the OAuth 2.0 terminology of [RFC6749] and [RFC8693]. In particular:

actor: The OAuth client that initiates a token exchange. In identity-chaining terminology, the actor is the principal recorded in the "act" claim of the issued chained access token.

subject: The principal whose identity is asserted in the "sub" claim of the chained access token. In the cases this document addresses, the subject is always an end user.

chained access token:
An OAuth access token issued by RFC 8693 token exchange whose "sub" is an end user and whose "act" records an acting OAuth client. Typically an RFC 9068 JWT access token [RFC9068].

delegation handle (or "handle"):
The new token type defined in this document. A JWT issued by the authorization server alongside a chained access token, that the acting client MAY present back to the authorization server to obtain a refreshed chained access token without prompting the end user.

refresh: The act of presenting a delegation handle to the

authorization server to obtain a new chained access token. Distinct from RFC 6749 refresh-token usage.

sender-constrained:
: A property of a token that ties its usability to a specific cryptographic key or credential held by the intended presenter. In this document, the handle is sender-constrained to the acting client recorded in "act.sub" via the client's existing strong authentication credential (see Section 10.2).

3. Protocol Overview

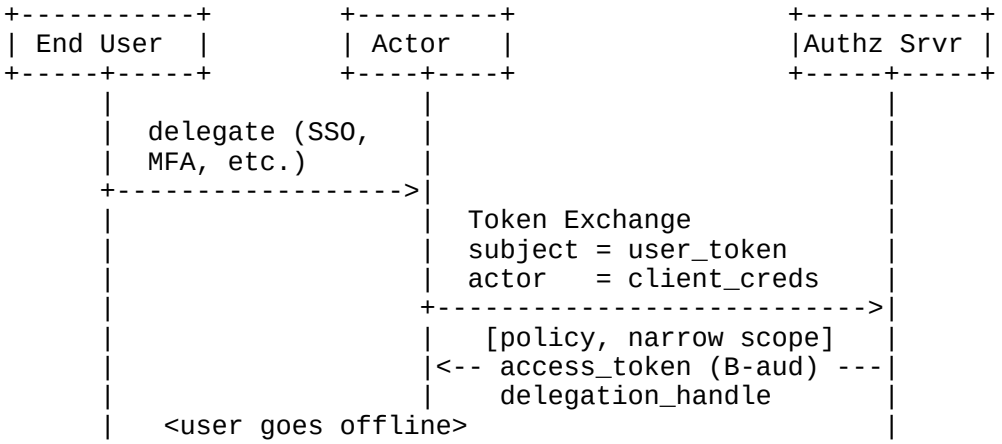
The protocol consists of two flows:

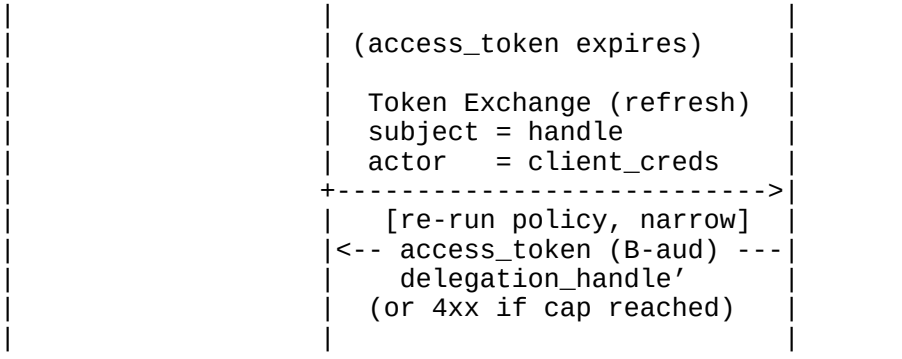
1. *Issuance*: A standard RFC 8693 token exchange, extended so that the authorization server MAY return a delegation handle alongside the chained access token in the response. Issuance of a handle is gated by authorization-server policy.

2. *Refresh*: A subsequent RFC 8693 token exchange in which the acting client presents the handle as the "subject_token" with a new "subject_token_type" URI defined by this document. The authorization server re-runs its identity-chaining validation pipeline and, if all checks pass, returns a new chained access token (and optionally a new handle).

No new endpoints are introduced. The handle reuses the existing token-exchange endpoint for both issuance and refresh.

The following diagram summarizes the flows:





The dashed line "<user goes offline>" is the load-bearing case:

refresh succeeds without any user-presence requirement, while preserving the identity binding and policy gating that existed at issuance.

4. The Delegation Handle Token

The handle is a signed JWT [RFC7519] using the JWS Compact Serialization [RFC7515].

4.1. JOSE Header

The JOSE header MUST include:

* "alg": the JWS signing algorithm. Authorization servers MUST support "RS256" and SHOULD support an Elliptic Curve algorithm such as "ES256".

* "typ": MUST be set to "dh+jwt" (case-insensitive). This typ value uniquely identifies the token as a delegation handle and prevents cross-type confusion (Section 3.11 of [RFC8725]).

* "kid": SHOULD be present and refer to a key published in the authorization server's JWKS.

4.2. Claims

The handle's JWT body MUST include the following claims:

"iss"
: The authorization server's issuer identifier, matching the "issuer" value published in the OpenID Connect Discovery [OpenID.Discovery] metadata.

"sub"
: The end-user subject identifier, identical to the "sub" of the original "subject_token" presented at handle issuance. This claim MUST identify an end user; an authorization server MUST NOT issue a handle whose "sub" identifies a service principal.

"aud"
: The OAuth client identifier of the acting client. This audience ensures the handle is unusable as a downstream access token: any resource server processing it according to [RFC9068] will reject it because "aud" does not name the resource server.

"azp"
: The OAuth client identifier of the acting client. Same value as "aud" for a handle.

"act"
: An RFC 8693 "act" claim whose "sub" member identifies the original acting client. This claim is the cryptographic anchor of the handle's sender constraint (Section 10.2). The "act" claim MAY be nested when an upstream identity-chained refresh was itself the source of the original exchange.

"delegated_aud"
: A new claim defined by this document. A string (or array of

strings) identifying the downstream resource server audience(s) for which refreshed chained access tokens MAY be issued. This value MUST be a subset of the audiences for which the original chained access token was issued.

"scope"
A space-separated list of OAuth scopes [RFC6749], identifying the maximum scopes for which refreshed chained access tokens MAY be issued. This MUST be a subset of the scopes granted in the original token-exchange response.

"refreshes_remaining"
A new claim defined by this document. A non-negative integer identifying how many further refreshes the authorization server will accept against this handle. When zero, the handle is exhausted and the authorization server MUST refuse further refreshes (returning "invalid_grant").

"exp"
The expiry time of the handle (seconds since the Epoch). The authorization server MUST set "exp" no later than the lifetime of the user session that produced the original delegation

(Section 8.3) and no later than the per-(actor, delegated_aud) maximum lifetime configured in policy (Section 7).

"iat"
The time the handle was issued.

"nbf"
OPTIONAL. The time before which the handle is not valid.

"jti"
A unique identifier for this handle. Used for revocation and replay detection.

The handle’s JWT body MAY include the following claims:

"acr", "amr"
The authentication-context-class and authentication-methods references, carried forward from the original "subject_token". Authorization-server policy MAY require a minimum "acr" on refresh; see Section 10.8.

"cnf"
An RFC 7800 confirmation claim [RFC7800] when DPoP, mTLS, or another proof-of-possession mechanism is used instead of (or in addition to) "act"-based sender constraint. See Section 9.4 and Section 9.5.

The handle’s JWT body MUST NOT include:

* Any claim that grants permission, scope, or audience exceeding that of the original chained access token.

* An "act" chain whose deepest entry identifies an end user (the acting principal must always be an OAuth client).

4.3. Example

An example handle issued to acting client "https://actor.example/" alongside a chained access token for downstream resource server "https://resource.example/" might be (with line breaks for readability):

```
Header (decoded):
  {
    "alg": "RS256",
    "typ": "dh+jwt",
    "kid": "as-2026-q2"
```



```
  }

Payload (decoded):
  {
    "iss": "https://as.example/",
    "sub": "user-1234",
    "aud": "https://actor.example/",
    "azp": "https://actor.example/",
    "act": {
      "sub": "https://actor.example/"
    },
    "delegated_aud": "https://resource.example/",
    "scope": "read:documents write:comments",
    "refreshes_remaining": 8,
    "acr": "urn:mace:incommon:iap:silver",
    "amr": ["pwd", "mfa"],
    "iat": 1747965600,
    "nbf": 1747965595,
    "exp": 1747994400,
    "jti": "01J5MZ9CHA1B2D3E4F5G6H7J8K"
  }
```

5. Issuance During Token Exchange

5.1. Initial Token Exchange Request

The acting client requests a chained access token using a standard RFC 8693 token-exchange request. The acting client MAY signal a preference for receiving a handle by including the OPTIONAL "request_delegation_handle" parameter:

```
POST /oauth2/token HTTP/1.1
Host: as.example
Content-Type: application/x-www-form-urlencoded
Authorization: <client authentication per [RFC8693] Section 2.1>

grant_type=
 urn%3Aietf%3Aparams%3Aoauth%3Agrant-type%3Atoken-exchange
&resource=https%3A%2F%2Fresource.example%2F
&scope=read%3Adocuments+write%3Acomments
&subject_token=...
&subject_token_type=urn%3Aietf%3Aparams%3Aoauth%3Atoken-type%3Ajwt
&actor_token=...
&actor_token_type=urn%3Aietf%3Aparams%3Aoauth%3Atoken-type%3Ajwt
&request_delegation_handle=true
```

The "request_delegation_handle" parameter is a boolean encoded as

the string "true" or "false". Its absence is equivalent to

"false".

The authorization server MAY issue a handle even if "request_delegation_handle" is absent or "false", subject to policy; conversely, the authorization server MAY decline to issue a handle even if "request_delegation_handle=true" was requested, without failing the underlying token exchange.

5.2. Token Exchange Response Extensions

When the authorization server issues a delegation handle, the token-exchange response (defined in Section 2.2.1 of [RFC8693]) MUST include the following additional parameters:

"delegation_handle"
: REQUIRED. The handle JWT as a string.

"delegation_handle_expires_in"
: RECOMMENDED. The lifetime of the handle in seconds. The acting client SHOULD treat this as advisory; the authoritative expiry is the "exp" claim within the handle.

The acting client MUST treat the value of "delegation_handle" as opaque if it does not implement this document. Clients that do implement this document MAY inspect the handle's claims (notably "refreshes_remaining" and "exp") to make scheduling decisions, but MUST NOT take any authorization decision based on the handle's claims (the authorization server is authoritative on every refresh).

5.3. Authorization Server Behaviour

On receiving a token-exchange request, the authorization server:

1. Performs the validation pipeline required by [RFC8693] and any applicable identity-chaining or local-policy rules.

2. Determines, by consulting its policy (Section 7), whether a handle MAY be issued for the requested (actor, delegated audience, scope, "acr") tuple. Policy MAY require explicit opt-in for the actor, the audience, or both.

3. If a handle is to be issued, MUST mint it with:

    * "sub" identical to the "sub" of the issued chained access token.

    * "aud" and "azp" identifying the authenticated client (the actor of this exchange).

- "act.sub" identifying the authenticated client.
- "delegated_aud" identifying the audience of the issued chained access token (or a subset thereof, if policy narrows it).
- "scope" no broader than the scope granted on the chained access token.
- "refreshes_remaining" set to the per-policy maximum (or less).
- "exp" set to the earlier of (a) "iat" plus the per-policy maximum handle lifetime, (b) the end of the user session, (c) any other policy-imposed cap.

4. MUST include "delegation_handle" and "delegation_handle_expires_in" in the response.

5. MUST emit an audit event identifying the handle ("jti", "sub", "act.sub", "delegated_aud", "scope", and the policy version that authorized issuance).

5.4. Example

Continuing the example of Section 4.3:

```
HTTP/1.1 200 OK
Content-Type: application/json
Cache-Control: no-store

{
  "access_token": "eyJhbGciOiJSUzI1NiIs...",
  "issued_token_type":
      "urn:ietf:params:oauth:token-type:access_token",
  "token_type": "Bearer",
  "expires_in": 3600,
  "scope": "read:documents write:comments",
  "delegation_handle": "eyJhbGciOiJSUzI1NiIs...",
  "delegation_handle_expires_in": 28800
}
```

Here the chained access token lives for 3600 seconds (one hour) and the handle lives for 28800 seconds (eight hours), permitting

the acting client to refresh the chained access token up to "refreshes_remaining" times during the eight-hour window.

6. Refreshing a Chained Access Token

6.1. Refresh Request

To refresh, the acting client sends a token-exchange request to the same token endpoint, with:

- Client authentication identifying the actor (the same client identified in "act.sub" of the handle).

* "grant_type" set to "urn:ietf:params:oauth:grant-type:token-exchange".

* "subject_token" set to the handle JWT.

* "subject_token_type" set to "urn:ietf:params:oauth:token-type:delegation-handle" (defined in Section 12.3).

* "resource" or "audience" identifying a member of the handle's "delegated_aud".

* "scope" identifying a subset of the handle's "scope".

* OPTIONAL "actor_token" if a token-form actor assertion is desired; when omitted, the authenticated client identity is authoritative.

* OPTIONAL "request_delegation_handle=true" to request that a new handle be returned alongside the new chained access token.

Example:

```
POST /oauth2/token HTTP/1.1
Host: as.example
Content-Type: application/x-www-form-urlencoded
Authorization: <client authentication>

grant_type=
 urn%3Aietf%3Aparams%3Aoauth%3Agrant-type%3Atoken-exchange
&subject_token=eyJhbGciOiJSUzI1NiIs...
&subject_token_type=
 urn%3Aietf%3Aparams%3Aoauth%3Atoken-type%3Adelegation-handle
&resource=https%3A%2F%2Fresource.example%2F
&scope=read%3Adocuments
```



```
&request_delegation_handle=true
```

6.2. Authorization Server Validation Pipeline

On receiving a refresh request, the authorization server MUST:

1. Authenticate the client and confirm that the client identifier matches the value of "act.sub" in the handle. If they differ, return "invalid_grant".

2. Verify the handle's JWS signature using a key in its own JWKS, and verify "iss", "aud" == the calling client, and the "typ" header == "dh+jwt". If any check fails, return "invalid_grant".

3. Verify the handle is not in the handle-revocation list and has not exceeded its "exp" or had its "refreshes_remaining" reach zero. Otherwise, return "invalid_grant".

4. Verify the requested "resource"/"audience" is present in "delegated_aud". Otherwise, return "invalid_target" (Section 2 of [RFC8707]).

5. Verify the requested "scope" is a subset of the handle's "scope". Otherwise, return "invalid_scope".

6. Verify the end user's session is still valid according to authorization-server policy (for example, that the session has not been revoked, and that any session-idle or absolute timeout has not been exceeded). Otherwise, return "invalid_grant".

7. Re-execute the full identity-chaining policy pipeline for the (actor, delegated audience, scope, user-permissions) tuple as though this were an initial exchange (Section 7). In particular, the authorization server MUST re-check the end user's current permissions on the downstream audience; if the user's permissions have narrowed, the granted scope MUST be narrowed to the intersection or the request MUST fail with "invalid_scope".

8. Mint a new chained access token with the granted scope and authoritative-server-chosen expiry.

9. If "request_delegation_handle=true", mint a new handle with "refreshes_remaining" decremented by at least one and all other claims unchanged or narrowed. The new handle's "exp" MUST NOT exceed the original handle's "exp". The previous handle's "jti" MUST be added to the handle-revocation list to prevent re-use (see Section 8.1).

10. Emit an audit event linking the previous handle's "jti", the new handle's "jti" (if any), the issued chained access token's "jti", and the policy decisions made.

The "refreshes_remaining" counter MAY be decremented by more than one if policy considers the refresh "expensive" (for example, a refresh that triggered a fresh upstream permissions lookup).

6.3. Refresh Response

The refresh response is a standard RFC 8693 token-exchange response with the additions defined in Section 5.2. When a new handle is issued, the response includes "delegation_handle" and "delegation_handle_expires_in"; when not issued, those parameters are absent.

The previous handle MUST NOT be presented again; its "jti" is in the revocation list as of step 9 of Section 6.2.

6.4. Termination

Refresh ends when any of the following occur:

* "refreshes_remaining" reaches zero;

* the handle's "exp" passes;

* the user session is revoked or expires (Section 8.3);

* the handle's "jti" is added to the handle-revocation list

(Section 8.1);

* the acting client's credentials are revoked (Section 8.2);

* authorization-server policy changes such that the issued chain is no longer permitted (Section 7).

In all cases, the authorization server MUST return "invalid_grant" with no further detail (per [RFC6749] Section 5.2) and SHOULD log the reason for operational diagnosis.

6.5. Example

Refresh response:

```
HTTP/1.1 200 OK
Content-Type: application/json
Cache-Control: no-store

{
  "access_token": "eyJhbGciOiJSUzI1NiIs...",
  "issued_token_type":
      "urn:ietf:params:oauth:token-type:access_token",
  "token_type": "Bearer",
  "expires_in": 3600,
  "scope": "read:documents",
  "delegation_handle": "eyJhbGciOiJSUzI1NiIs...",
  "delegation_handle_expires_in": 25200
}
```

Note that "scope" has narrowed from the original "read:documents write:comments" to "read:documents" (the acting client requested only the narrower scope), and the new handle's advertised "expires_in" reflects the remaining window of the original handle's "exp" (which the new handle inherits).

7. Authorization Server Policy

Issuance and refresh of delegation handles are governed by authorization-server policy. This document does not standardize policy format, but specifies what policy MUST and SHOULD parameterize:

* Per-(actor, delegated audience) opt-in for handle issuance. The default SHOULD be that handles are not issued unless policy explicitly permits it for the (actor, delegated audience) pair.

* Maximum handle lifetime ("max_handle_ttl_seconds") per (actor, delegated audience). This SHOULD be capped at the maximum user session lifetime.

* Maximum number of refreshes ("max_refreshes_per_handle") per (actor, delegated audience).

* Minimum acceptable "acr" for handle issuance and for each

refresh. Policy MAY require fresh user authentication (an "acr" that is achievable only with the user present) for

certain audiences or scopes; refresh against such policy will fail and force the actor back to a fresh user-driven delegation.

* Whether and how the user is informed at delegation time that a handle will be issued (consent UX, opt-in checkbox, etc.). This document takes no position on whether end-user consent is required in addition to operator policy; it is a policy decision.

* Audit-event sinks and retention.

Policy MUST be evaluated on every refresh, not only at issuance. A policy change that disallows a previously allowed (actor, delegated audience, scope) tuple MUST cause subsequent refreshes against an outstanding handle for that tuple to fail.

8. Revocation

8.1. Handle-Level Revocation

Authorization servers MUST support revocation of individual handles by their "jti". The revocation endpoint MAY be the existing OAuth 2.0 token revocation endpoint [RFC7009]; the "token_type_hint" value "delegation_handle" (registered in Section 12.4) MAY be used.

On accepting a revocation, the authorization server MUST add the handle's "jti" to a handle-revocation set consulted by step 3 of Section 6.2. Handle-revocation entries MAY be aged out once the handle's "exp" has passed.

Authorization servers MAY also support bulk revocation by:

* user identifier (revokes all handles where "sub" = user);

* acting client identifier (revokes all handles where "act.sub" = client);

* user-session identifier (when sessions carry a stable id).

Bulk revocation is RECOMMENDED for the incident-response cases described in Section 8.2.

8.2. Cascading Revocation via Client Credential Revocation

Because the handle is sender-constrained via the acting client's

credentials (Section 10.2), revocation of the acting client's

credentials (for example, removal of its public key from the authorization server's client registry) immediately invalidates every outstanding handle for which that client is the actor: any refresh attempt fails at step 1 of Section 6.2 because client authentication fails.

This cascade is achieved by the signature-check / key-check pipe on every refresh, not by an explicit registry walk, and so is immediate and complete for the credential type concerned.

8.3. User Session Revocation

When the end user's session is revoked, all outstanding handles for that user MUST become un-refreshable within one cache TTL of the user-session check.

Authorization servers SHOULD evaluate the user-session check on every refresh against a fresh-enough projection of session state (typically a cache with a TTL of seconds), and SHOULD subscribe to session-revocation events from the authentication subsystem to invalidate the cache eagerly when possible.

9. Relationship to Other Specifications

9.1. OAuth 2.0 Refresh Tokens (RFC 6749)

The handle is not an RFC 6749 refresh token. RFC 6749 refresh tokens are typically opaque, are issued to and authorize refresh of access tokens for the same OAuth client / user pair as the original grant, and have no standardized claim model.

The handle is a JWT with a defined claim model that explicitly carries the chained-identity binding ("sub" = user, "act.sub" = actor). It is also presented at the token-exchange endpoint (not the refresh-token endpoint) and uses the RFC 8693 grant.

An authorization server MAY internally implement the handle on top of its existing refresh-token infrastructure; the wire format and validation rules in this document are the normative contract.

9.2. RFC 8693 refresh_token Response Parameter

Section 2.2.1 of [RFC8693] permits a "refresh_token" parameter in the token-exchange response but does not specify how the refresh token should be shaped or validated for the chained-identity case. This document provides one concrete shape and validation

model.

An authorization server implementing this document SHOULD use "delegation_handle" rather than "refresh_token" in token-exchange responses for chained-identity flows, so that clients can distinguish a handle (with the validation properties defined here) from an opaque RFC 6749-style refresh token.

9.3. OAuth Identity Chaining

This document is designed to interoperate with [I-D.ietf-oauth-identity-chaining]. The handle preserves the

"act" claim chain across refresh; the "sub" remains the original user; the "delegated_aud" identifies a member of the audiences that the identity-chaining mechanism originally permitted.

Where the identity-chaining draft defines policy hooks at the authorization server, this document extends those hooks with the handle-specific policy parameters listed in Section 7.

9.4. Demonstrating Proof of Possession (DPoP)

When DPoP [RFC9449] is in use for client authentication or token-sender-constraint, the handle MAY additionally carry a "cnf" claim binding it to the same key. In that case, refresh MUST also require a valid DPoP proof for the request, and the authorization server MUST verify the DPoP proof in addition to the checks in Section 6.2.

The two sender-constraint mechanisms ("act"-claim binding plus client authentication, and "cnf"-claim binding plus DPoP) MAY be combined; when combined, both MUST pass.

9.5. Mutual TLS Client Authentication (RFC 8705)

When mTLS [RFC8705] is the client authentication mechanism, the handle MAY carry a "cnf" claim with the client certificate thumbprint. Refresh MUST be performed over a mutually authenticated TLS connection whose client certificate matches.

10. Security Considerations

10.1. Threat Model

The principal threats this document addresses are:

T1. *Actor compromise leading to indefinite user-scoped action.* Without bounded refresh and rapid revocation, a compromised actor could continue to act as the user indefinitely once any handle has been issued. See Section 10.6 and Section 10.7.

T2. *Handle theft.* An attacker who obtains a handle but not the acting client's credentials must not be able to use the handle. See Section 10.2.

T3. *Cross-audience confusion.* An attacker who obtains a handle (with "aud" = actor) must not be able to present it as a downstream access token. See Section 10.12.

T4. *Scope or audience widening.* An attacker who obtains a handle must not be able to acquire chained access tokens for audiences or scopes broader than those the user originally delegated. See Section 10.3 and Section 10.4.

T5. *Acting against a user whose session has been revoked.* See Section 10.5.

T6. *Acting under stale authentication context.* An attacker who obtains a handle minted when the user satisfied weak

authentication must not thereby gain refresh against audiences requiring stronger authentication. See Section 10.8.

10.2. Sender Constraint

The handle MUST be sender-constrained to the acting client recorded in "act.sub". This document defines three mechanisms, any one of which is sufficient:

1. *Client-authentication binding* (REQUIRED baseline). The authorization server requires strong client authentication (such as "private_key_jwt" per Section 9 of [OpenID.Core] or "tls_client_auth" per Section 2.1 of [RFC8705]) on refresh, and verifies that the authenticated client identifier matches "act.sub". Possession of the handle alone is insufficient; the attacker also needs the client’s authentication credential.

2. *DPoP-key binding* (OPTIONAL, see Section 9.4). The handle includes a "cnf" claim per [RFC9449], and refresh requires a DPoP proof.

3. *mTLS binding* (OPTIONAL, see Section 9.5).

Bearer-only handles (with no sender constraint) are FORBIDDEN.

An authorization server MUST NOT issue a handle if the original exchange was authenticated by a method that does not satisfy at least one of these mechanisms. For example, "client_secret_post" (Section 2.3.1 of [RFC6749]) MUST NOT be used for handle issuance unless combined with one of the proof-of-possession mechanisms above; a stolen "client_secret" would otherwise be sufficient to refresh every outstanding handle for that client.

10.3. Audience Lock

"delegated_aud" is single-valued in the common case and is set at issuance. Refresh MUST verify that the requested "resource"/"audience" is a member of "delegated_aud".

An array-valued "delegated_aud" is permitted but SHOULD be used sparingly: it widens the blast radius of handle theft from one audience to all listed audiences.

10.4. Scope Monotonicity

"scope" in the handle MUST be the maximum. Refresh narrows or preserves scope; it never widens. This MUST be enforced at step 5 of Section 6.2.

In addition, the user-permission check at step 7 of Section 6.2 means that if the user’s permissions on the downstream audience have narrowed since handle issuance (for example, the user lost a project role), the next refresh narrows the granted scope accordingly or fails.

10.5. User-Session Binding

The handle MUST be invalidated when the underlying user session ends. See Section 8.3.

Authorization servers MUST set the handle's "exp" no later than the projected end of the user session (as known at issuance). For long-lived sessions, this still leaves the handle subject to the policy-imposed "max_handle_ttl_seconds".

10.6. Refresh Cap and Handle Lifetime Cap

Two independent caps bound the exposure window:

* "refreshes_remaining" caps the number of chained access tokens

derivable from a single handle.

* "exp" caps the wall-clock window during which the handle may be exercised.

Operators SHOULD set both to the smallest values that meet the operational requirement. Typical values for an eight-hour asynchronous workload might be eight refreshes over eight hours (roughly one refresh per hour); higher rates may be appropriate for shorter tasks with more aggressive turnover.

10.7. Cascading Revocation

The cascading revocation property of Section 8.2 is crucial: incident-response procedures for actor compromise SHOULD revoke the actor's credential at the authorization server, which instantly invalidates all outstanding handles for that actor without any per-handle bookkeeping.

10.8. Step-Up Authentication Preservation

The handle's "acr" / "amr" claims MUST be carried forward from the original chained access token, and authorization-server policy MAY require a minimum "acr" for refresh.

For high-assurance audiences, policy SHOULD require an "acr" only achievable with the user present (for example, requiring a fresh MFA assertion). Refresh against such audiences will fail deliberately, forcing the actor to obtain a fresh user-driven delegation.

10.9. Comparison to Refresh Token Theft

The handle's security posture is approximately equal to that of a sender-constrained refresh token (such as one bound via DPoP or mTLS), with two additions:

* The handle carries the chained-identity claim model in the clear, so any introspection, audit, or anomaly-detection system can reason about which user and which actor the handle authorizes.

* The handle's refresh path re-runs the identity-chaining policy pipeline on every refresh, not just at issuance, narrowing automatically when user permissions or operator policy change.

Conventional bearer-only refresh tokens are NOT a substitute, and

authorization servers MUST NOT implement this document with bearer-only handles (Section 10.2).

10.10. Handle Storage at the Client

Acting clients MUST treat the handle as a high-value secret. Storage SHOULD be encrypted at rest, with access limited to the process(es) that initiate refresh. Clients SHOULD prefer secret-manager-backed storage (cloud KMS, Hardware Security Modules, or equivalent) over plaintext on disk.

Clients SHOULD scope each handle to the smallest unit of work that may need refresh (for example, one per long-running task) rather than reusing a single handle across unrelated tasks, to minimize blast radius from storage compromise.

10.11. Authorization Server Compromise

Authorization-server compromise is out of scope for this document, but a compromise of the authorization server’s signing keys would permit forgery of handles and of chained access tokens alike. Operators SHOULD rotate signing keys per established practice for OIDC discovery (see [OpenID.Discovery]).

Compromise of the authorization server’s handle-revocation list storage MAY permit replay of previously revoked handles until the handle’s natural expiry; operators SHOULD treat the revocation list as integrity-critical.

10.12. Cross-Audience Confusion

A handle has "aud" = the acting client, not the downstream resource server. A resource server implementing RFC 9068 verification will reject a handle presented as a bearer token, because "aud" does not name the resource server.

However, resource-server implementations that fail to verify "aud" strictly (for example, that accept any well-signed token from the issuer) are vulnerable. Implementations conforming to [RFC9068] are safe. Operators MUST ensure all resource-server token verifiers check "aud" against the resource server’s own identifier and SHOULD additionally check the "typ" header is either "at+jwt" [RFC9068] or another value explicitly accepted for that resource (and not "dh+jwt").

11. Privacy Considerations

The handle carries the same user identifier ("sub") as the

chained access token it accompanies, with identical privacy characteristics. No new categories of personal data are introduced.

Because the handle MAY be retained at the actor for hours, an operator considering whether to enable handles for a given actor SHOULD consider whether the actor's storage and access controls are appropriate for retention of a user-identifying token for that period. Per-user opt-out, per-task consent prompts, and maximum handle lifetimes are policy levers an operator may use to manage this.

Audit logs generated by handle issuance and refresh contain user identifiers and SHOULD be subject to the same retention and access controls as other identity-bearing audit data.

12. IANA Considerations

12.1. Media Type Registration

IANA is requested to register the following media type per [RFC6838]:

Type name: application
Subtype name: dh+jwt
Required parameters: N/A
Optional parameters: N/A
Encoding considerations: binary; JWT values are encoded as a series of base64url-encoded values separated by period characters.
Security considerations: See Section 10 of this document and [RFC8725].
Interoperability considerations: N/A
Published specification: This document.
Applications that use this media type: Applications using OAuth 2.0 Token Exchange [RFC8693] that implement the asynchronous delegation handle defined in this document.
Fragment identifier considerations: N/A
Additional information:
Magic number(s): N/A
File extension(s): N/A
Macintosh file type code(s): N/A
Person and email address to contact for further information:
The authors of this document.
Intended usage: COMMON
Restrictions on usage: N/A

Author: L. Zhu, S. Currie
Change controller: IETF

12.2. JOSE Header Parameter Values

IANA is requested to add the following entry to the "JSON Web Signature and Encryption Type Values" sub-registry of the "JSON Object Signing and Encryption (JOSE)" registry:

"typ" Header Parameter Value: dh+jwt
Abbreviation for MIME Type: application/dh+jwt
Change Controller: IETF

Reference: This document

12.3. OAuth URI Registry: subject_token_type

IANA is requested to add the following entry to the "OAuth URI" registry established by [RFC6755]:

URN: urn:ietf:params:oauth:token-type:delegation-handle
Common Name: Delegation Handle Token Type
Change Controller: IETF
Specification Document(s): This document

12.4. OAuth Token Exchange Response Parameters

IANA is requested to add the following entries to the "OAuth Parameters" registry established by [RFC6749] (sub-registry "OAuth Parameters" with usage location "token response"):

Parameter name: delegation_handle
Parameter usage location: token response
Change Controller: IETF
Specification Document(s): This document

Parameter name: delegation_handle_expires_in
Parameter usage location: token response
Change Controller: IETF
Specification Document(s): This document

Parameter name: request_delegation_handle
Parameter usage location: token request
Change Controller: IETF
Specification Document(s): This document

IANA is also requested to add the following entry to the "OAuth Token Type Hints" sub-registry established by [RFC7009]:

Hint Value: delegation_handle
Change Controller: IETF
Specification Document(s): This document

12.5. JSON Web Token Claims Registry

IANA is requested to add the following entries to the "JSON Web Token Claims" registry established by [RFC7519]:

Claim Name: delegated_aud
Claim Description: Downstream audience(s) for which refreshed chained access tokens may be issued via this delegation handle.
Change Controller: IETF
Reference: This document

Claim Name: refreshes_remaining
Claim Description: Non-negative count of remaining refreshes a delegation handle authorizes.
Change Controller: IETF
Reference: This document

13. References

13.1. Normative References

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

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

[RFC6755] Campbell, B. and H. Tschofenig, "An IETF URN Sub-Namespace for OAuth", RFC 6755, DOI 10.17487/RFC6755, October 2012, <https://www.rfc-editor.org/info/rfc6755>.

[RFC6838] Freed, N., Klensin, J., and T. Hansen, "Media Type Specifications and Registration Procedures", BCP 13, RFC 6838, DOI 10.17487/RFC6838, January 2013, <https://www.rfc-editor.org/info/rfc6838>.

[RFC7009] Lodderstedt, T., Ed., Dronia, S., and M. Scurtescu, "OAuth 2.0 Token Revocation", RFC 7009, DOI 10.17487/RFC7009, August 2013,

<https://www.rfc-editor.org/info/rfc7009>.

[RFC7515] Jones, M., Bradley, J., and N. Sakimura, "JSON Web Signature (JWS)", RFC 7515, DOI 10.17487/RFC7515, May 2015, <https://www.rfc-editor.org/info/rfc7515>.

[RFC7519] Jones, M., Bradley, J., and N. Sakimura, "JSON Web Token (JWT)", RFC 7519, DOI 10.17487/RFC7519, May 2015, <https://www.rfc-editor.org/info/rfc7519>.

[RFC7800] Jones, M., Bradley, J., and H. Tschofenig, "Proof-of-Possession Key Semantics for JSON Web Tokens (JWTs)", RFC 7800, DOI 10.17487/RFC7800, April 2016, <https://www.rfc-editor.org/info/rfc7800>.

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

[RFC8693] Jones, M., Nadalin, A., Campbell, B., Bradley, J., and C. Mortimore, "OAuth 2.0 Token Exchange", RFC 8693, DOI 10.17487/RFC8693, January 2020, <https://www.rfc-editor.org/info/rfc8693>.

[RFC8705] Campbell, B., Bradley, J., Sakimura, N., and T. Lodderstedt, "OAuth 2.0 Mutual-TLS Client Authentication and Certificate-Bound Access Tokens", RFC 8705, DOI 10.17487/RFC8705, February 2020, <https://www.rfc-editor.org/info/rfc8705>.

[RFC8707] Campbell, B., Bradley, J., and H. Tschofenig, "Resource Indicators for OAuth 2.0", RFC 8707, DOI 10.17487/RFC8707, February 2020,

            <https://www.rfc-editor.org/info/rfc8707>.

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

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

   [RFC9449] Fett, D., Campbell, B., Bradley, J., Lodderstedt, T.,

              Jones, M., and D. Waite, "OAuth 2.0 Demonstrating Proof
              of Possession (DPoP)", RFC 9449,
              DOI 10.17487/RFC9449, September 2023,
              <https://www.rfc-editor.org/info/rfc9449>.

13.2. Informative References

   [I-D.ietf-oauth-identity-chaining]
              Schwenkschuster, A., Kasselman, P., Tschofenig, H.,
              Saakian, A., and B. Campbell, "OAuth Identity and
              Authorization Chaining Across Domains", Work in
              Progress, Internet-Draft, draft-ietf-oauth-identity-
              chaining-latest, <https://datatracker.ietf.org/doc/
              draft-ietf-oauth-identity-chaining/>.

   [OpenID.Core]
              Sakimura, N., Bradley, J., Jones, M., de Medeiros, B.,
              and C. Mortimore, "OpenID Connect Core 1.0
              incorporating errata set 2", December 2023,
              <https://openid.net/specs/openid-connect-core-1_0.html>.

   [OpenID.Discovery]
              Sakimura, N., Bradley, J., Jones, M., and E. Jay,
              "OpenID Connect Discovery 1.0 incorporating errata set
              2", December 2023,
              <https://openid.net/specs/openid-connect-
              discovery-1_0.html>.

Acknowledgments

   The mechanism in this document grew out of operational experience
   with identity-chained service-to-service authorization at
   Atlassian. The authors thank the Atlassian Platform Security and
   Atlassian Token Service teams for review and pressure-testing of
   earlier drafts, in particular for surfacing the asynchronous
   workload class that motivates this work.

   Thanks also to participants in informal discussions of the
   problem space prior to this document’s publication; specific
   acknowledgments will be added in a future revision.

Authors’ Addresses

   Larry Zhu
   Atlassian

Email: lzhu3@atlassian.com

Sam Currie
Atlassian
Email: sam@atlassian.com