oauth
Internet-Draft
Intended status: Standards Track
Expires: 12 November 2026

A. Schwenkschuster
P. Kasselmann
Defakto Security
K. Burgin
MITRE
M. Jenkins
NSA-CCSS
B. Campbell
Ping Identity
A. Parecki
Okta
11 May 2026

# OAuth Identity and Authorization Chaining Across Domains
draft-ietf-oauth-identity-chaining-12

## Abstract

This specification defines a mechanism to preserve identity and authorization information across trust domains that use the OAuth 2.0 Framework.

## Discussion Venues

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

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

Source for this draft and an issue tracker can be found at https://github.com/oauth-wg/oauth-identity-chaining.

## Status of This Memo

Table of Contents

## 1. Introduction

Applications often require access to resources that are distributed across multiple trust domains where each trust domain has its own OAuth 2.0 authorization server. A request may transverse multiple resource servers in multiple trust domains before completing. All protected resources involved in such a request need to know on whose behalf the request was originally initiated (i.e. the user), what authorization was granted and optionally which other resource servers were called prior to making an authorization decision. This information needs to be preserved, even when a request crosses one or more trust domains. This document refers to this as "chaining" and defines a common pattern for combining OAuth 2.0 Token Exchange [RFC8693] and the JSON Web Token (JWT) Profile for OAuth 2.0 Client Authentication and Authorization Grants [RFC7523] to access resources across multiple trust domains while preserving identity and authorization information.

### 1.1. Requirements Language

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

## 2. Identity and Authorization Chaining Across Domains

This specification describes a combination of OAuth 2.0 Token Exchange [RFC8693] and JWT Profile for OAuth 2.0 Client Authentication and Authorization Grants [RFC7523] to achieve identity and authorization chaining across domains.

A client in trust domain A that needs to access a resource server in trust domain B requests a JWT authorization grant from the authorization server for trust domain A using a profile of OAuth 2.0 Token Exchange [RFC8693]. The client in trust domain A then presents the received grant as an assertion to the authorization server in trust domain B, using the JWT authorization grant feature of [RFC7523], to obtain an access token for the protected resource in trust domain B.

In some deployments, the client in trust domain A may obtain a JWT authorization grant using a proprietary API or interface other than the OAuth 2.0 Token Exchange protocol [RFC8693]. The details of such an interface are out of scope for this document but an alternative means of acquiring the JWT authorization grant is not precluded by this document. A JWT authorization grant, regardless of how it was obtained, MUST be used to request an access token from the authorization server in trust domain B as described in Section 2.4 of this document.

### 2.1. Overview

The identity and authorization chaining flow outlined below describes how a combination of OAuth 2.0 Token Exchange [RFC8693] and JWT Profile for OAuth 2.0 Client Authentication and Authorization Grants [RFC7523] are used to address the use cases identified. Conceptually, this is an exchange within the first domain that produces a JWT authorization grant intended for use in acquiring an access token from the second domain.

```
  +-------------+         +--------+         +-------------+ +---------+
  |Authorization|         | Client |         |Authorization| |Protected|
  |Server       |         | Trust  |         |Server       | |Resource |
  |Trust        |         |Domain A|         |Trust        | |Trust    |
  |Domain A     |         |        |         |Domain B     | |Domain B |
  +-------------+         +--------+         +-------------+ +---------+
         |                    |                     |             |
         |                    |----+                |             |
         |      (1) discover  |    |                |             |
         |      Authorization |<---+                |             |
         |      Server        |                     |             |
         |      Trust Domain B|                     |             |
         |                    |                     |             |
         | (2) exchange token |                     |             |
         |   [RFC 8693]       |                     |             |
         |<-------------------|                     |             |
         |                    |                     |             |
         | (3) <authorization |                     |             |
         |       grant JWT>   |                     |             |
         | - - - - - - - - - >|                     |             |
         |                    |                     |             |
         |                    | (4) present         |             |
         |                    | authorization grant |             |
         |                    | [RFC 7523]          |             |
         |                    | ------------------->|             |
         |                    |                     |             |
         |                    | (5) <access token>  |             |
         |                    | <- - - - - - - - - -|             |
         |                    |                     |             |
         |                    |               (6) access          |
         |                    | --------------------------------->|
         |                    |                     |             |
```

Figure 1: Identity and Authorization Chaining Flow

The flow illustrated in Figure 1 shows the steps the client in trust domain A needs to perform to access a protected resource in trust domain B. In this flow, the client is in possession of a token that an authorization server will accept as part of a token exchange flow as defined in Token Exchange (Section 2.3). How the client obtained this token is out of scope of this specification. The client has a way to discover the authorization server in domain B and a trust relationship exists between domain A and domain B. It includes the following:

1. The client in trust domain A discovers the location of the authorization server of trust domain B. See Authorization Server Discovery (Section 2.2).

2. The client in trust domain A exchanges a token it has in its possession with the authorization server in trust domain A for a JWT authorization grant that can be used at the authorization server in trust domain B. See Token Exchange (Section 2.3).

3. The authorization server of trust domain A processes the request and returns a JWT authorization grant that the client can use with the authorization server of trust domain B. This requires a trust relationship between the authorization servers in trust domain A and trust domain B. Such a trust relationship typically manifests as the exchange of key material, whereby the authorization server in domain B trusts the public key(s) of domain A, which are used to verify JWT authorization grants signed with the corresponding private key(s).

4. The client in trust domain A presents the authorization grant to the authorization server of trust domain B. See Access Token Request (Section 2.4.1).

5. Authorization server of trust domain B validates the JWT authorization grant and returns an access token. Validating the JWT authorization grant requires trusting the public key(s) of domain A and its authority to issue authorization grants. This might take the form of configuration and policy in domain B that associates a set of public keys with domain A. Or might rely on the keys published at domain A’s jwks_uri as listed in its Authorization Server Metadata [RFC8414].

6. The client in trust domain A uses the access token received from the authorization server in trust domain B to access the protected resource in trust domain B.

## 2.2. Authorization Server Discovery

This specification does not define authorization server discovery. A client may use the authorization_servers property as defined in OAuth 2.0 Protected Resource Metadata [RFC9728], maintain a static mapping or use other means to identify the authorization server.

## 2.3. Token Exchange

The client in trust domain A performs token exchange as defined in [RFC8693] with the authorization server in trust domain A in order to obtain a JWT authorization grant that can be used with the authorization server of trust domain B as specified in Section 1.3 of [RFC6749].

#### 2.3.1. Token Exchange Request

The parameters described in Section 2.1 of [RFC8693] apply here with the following restrictions:

scope
: Additional scopes to indicate scopes included in the returned JWT authorization grant, if required. See Claims transcription (Section 2.5).

resource
: URI of authorization server for trust domain B.

audience
: Well known/logical name of authorization server for trust domain B.

One of resource or audience is REQUIRED to indicate the intended authorization server in trust domain B.

#### 2.3.2. Processing rules

* If the request itself is not valid or if the given resource or audience are unknown, or are unacceptable based on policy, the authorization server in trust domain A MUST deny the request as defined in Section 2.2.2 of [RFC8693].

* The authorization server in trust domain A can add, remove or change claims. See Claims transcription (Section 2.5).

#### 2.3.3. Token Exchange Response

All of Section 2.2 of [RFC8693] applies. In addition, the following applies to implementations that conform to this specification.

* The "aud" claim in the returned JWT authorization grant MUST identify the requested authorization server in trust domain B. This corresponds with RFC 7523 Section 3, Point 3 (https://datatracker.ietf.org/doc/html/rfc7523#section-3) and is there to reduce misuse and to prevent clients from, among other things, presenting access tokens as an authorization grant to an authorization server in trust domain B.

* The "aud" claim included in the returned JWT authorization grant can identify multiple authorization servers, provided that trust relationships exist with them. However, it is RECOMMENDED that the "aud" claim is restricted to a single authorization server in trust domain B to prevent an authorization server from presenting

the client’s authorization grant to an authorization server in a different trust domain.  For example, this will prevent the authorization server in trust domain B from presenting the authorization grant it received from the client in trust domain A to the authorization server for trust domain C.

#### 2.3.4.  Example

The example below shows the message invoked by the client in trust domain A to perform token exchange with the authorization server in trust domain A (https://as.a.example/auth) to receive a JWT authorization grant for the authorization server in trust domain B (https://as.b.example/auth).

```
POST /auth/token HTTP/1.1
Host: as.a.example
Content-Type: application/x-www-form-urlencoded

grant_type=urn%3Aietf%3Aparams%3Aoauth%3Agrant-type%3Atoken-exchange
&resource=https%3A%2F%2Fas.b.example%2Fauth
&subject_token=ey...
&subject_token_type=
 urn%3Aietf%3Aparams%3Aoauth%3Atoken-type%3Aaccess_token
```

Figure 2: Token exchange request

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

{
  "access_token":"eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpc3MiOiJo
  dHRwczovL2FzLmEub3JnL2F1dGgiLCJleHAiOjE2OTUyODQwOTIsImlhdCI6MTY5N
  TI4NzY5Miwic3ViIjoiam9obl9kb2VAYS5vcmciLCJhdWQiOiJodHRwczovL2FzLm
  Iub3JnL2F1dGgifQ.304Pv9e6PnzcQPzz14z-k2ZyZvDtP5WIRkYPScwdHW4",
  "token_type":"N_A",
  "issued_token_type":"urn:ietf:params:oauth:token-type:jwt",
  "expires_in":60
}
```

Figure 3: Token exchange response

### 2.4.  JWT Authorization Grant

The client in trust domain A uses the JWT authorization grant obtained from the authorization server in trust domain A as an assertion to request an access token from the authorization server in trust domain B, as described in [RFC7523].

### 2.4.1. Access Token Request

The JWT authorization grant is used to request an access token as defined in Section 2.1 of [RFC7523]. The following required parameters are described additionally here:

grant_type
: As defined in Section 2.1 of [RFC7523] the value urn:ietf:params:oauth:grant-type:jwt-bearer indicates the request is a JWT bearer assertion authorization grant.

assertion
: The JWT authorization grant returned by the authorization server for domain A (see Token Exchange (Section 2.3) response).

The client in trust domain A can indicate the protected resource it is trying to access through the scope parameter or the resource parameter defined in [RFC8707].

### 2.4.2. Processing rules

The authorization server in trust domain B MUST validate the JWT authorization grant as specified in Sections 3 and 3.1 of [RFC7523]. The following processing rules also apply:

* The "aud" claim MUST identify the authorization server in trust domain B as a valid intended audience of the assertion using either the token endpoint as described Section 3 of [RFC7523] or the issuer identifier as defined in Section 2 of [RFC8414].

* The authorization server in trust domain B MUST deny the request if it is not able to identify the subject.

* Due to policy the request MAY be denied (for instance if a trust relationship with trust domain A is not established).

Section 3.1 of [RFC7523] describes the error response used in request denial cases.

### 2.4.3. Access Token Response

When the authorization grant has been validated, the authorization server in trust domain B responds with an access token as described in Section 5.1 of [RFC6749].

### 2.4.4. Example

The examples below show how the client in trust domain A presents an authorization grant to the authorization server in trust domain B (https://as.b.example/auth) to receive an access token for a protected resource in trust domain B.

```
POST /auth/token HTTP/1.1
Host: as.b.example
Content-Type: application/x-www-form-urlencoded

grant_type=urn%3Aietf%3Aparams%3Aoauth%3Agrant-type%3Ajwt-bearer
&assertion=ey...
```

Figure 4: Assertion request

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

{
  "access_token":"eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpc3MiOiJo
  dHRwczovL2FzLmIub3JnL2F1dGgiLCJleHAiOjE2OTUyODQwOTIsImlhdCI6MTY5N
  TI4NzY5Miwic3ViIjoiam9obi5kb2UuMTIzIiwiYXVkIjoiaHR0cHM6Ly9iLm9yZy
  9hcGkifQ.CJBuv6sr6Snj9in5T8f7g1uB61Ql8btJiR0IXv5oeJg",
  "token_type":"Bearer",
  "expires_in":60
}
```

Figure 5: Assertion response

## 2.5. Claims transcription

Claims transcription is motivated by the need to propagate user and client identifiers, authorization context, and other relevant information across trust boundaries. This enables the various entities involved to determine on whose behalf the request is being made, what authorization has been granted, and, potentially, which other resource servers were previously involved.

Authorization servers may transcribe claims when either producing JWT authorization grants in the token exchange flow or access tokens in the assertion flow. Transcription of claims may be required for the following reasons:

* *Transcribing the subject identifier*: The subject identifier can differ between the parties involved. For example, a user is identified in trust domain A as "johndoe@a.example" but in trust

domain B they are identified as "doe.john@b.example". The mapping from one identifier to the other can either happen in the token exchange step and the updated identifier is reflected in the returned JWT authorization grant or in the assertion step where the updated identifier would be reflected in the access token. To support this, both authorization servers can add, change or remove claims as described above.

* *Data Minimization*: Authorization servers can remove or hide certain claims due to privacy requirements or reduced trust towards the targeting trust domain. One example is a financial institution that integrates with a third-party payment gateway. Domain A (the financial institution) includes detailed claims such as "account type: premium" and "transaction limit: $10,000" in the JWT authorization grant. However, domain B (the payment gateway) only needs claims like "transaction limit" for its access control policies. Domain A transcribes the claims to exclude unnecessary information, ensuring that domain B receives only the claims relevant to its operations.

* *Controlling scope*: Clients can use the scope parameter to control transcribed claims (e.g. downscoping). Authorization Servers SHOULD verify that the requested scopes are not higher privileged than the scopes of the presented subject_token. For example, a cloud-based development platform that allows developers to access APIs across multiple trust domains where a developer in domain A requests access to an API in domain B but only needs limited permissions, such as "read-only" access. The authorization server in domain A transcribes the claims into the JWT authorization grant to reflect the downscoped permissions, removing higher-privileged claims like "write" or "admin." This ensures that the access token issued by domain B aligns with the developer’s intended scope of access.

* *Including JWT authorization grant claims*: The authorization server in trust domain B which is performing the assertion flow can leverage claims from the JWT authorization grant presented by the client in trust domain A and include them in the returned access token.

The representation of transcribed claims and their format is not defined in this specification.

When transcribing claims, it’s important that both the place where the claims are given and where they are interpreted agree on the semantics and that the access controls are consistent.

## 3. Authorization Server Metadata

The following authorization server metadata parameter is defined by this specification and is registered in the "OAuth Authorization Server Metadata" registry established in "OAuth 2.0 Authorization Server Metadata" [RFC8414].

identity_chaining_requested_token_types_supported
: OPTIONAL. JSON array containing a list of Token Types that can be requested as a requested_token_type in the Token Exchange request when performing Identity and Authorization Chaining Across Domains. In situations where it might be an information disclosure concern, authorization servers MAY choose not to advertise some supported requested token types even when this parameter is used, and lack of a value does not necessarily mean that the token type is unsupported.

## 4. IANA Considerations

### 4.1. OAuth Authorization Server Metadata Registry

This specification requests registration of the following parameter in the "OAuth Authorization Server Metadata" registry established in [RFC8414].

#### 4.1.1. Registry Contents

* Metadata Name: identity_chaining_requested_token_types_supported

* Metadata Description: JSON array containing a list of Token Type Identifiers supported as a requested_token_type in an Identity and Authorization Chaining Token Exchange ([RFC8693]) request.

* Change Controller: IETF

* Specification Document(s): Section 3 of [[this document]]

The parameter indicates the supported token types that can be requested in an [RFC8693] Token Exchange.

### 4.2. Media Types

This specification does not define any new media types.

Profiles or deployment-specific implementations can adopt explicit typing as defined in JSON Web Token Best Current Practices [RFC8725] and define a new media type [RFC2046] in the "Media Types" registry [IANA.media-types] in the manner described in [RFC6838].

# 5. Security Considerations

## 5.1. Client Authentication

Authorization Servers should follow Section 2.5 of the Best Current Practice for OAuth 2.0 Security [RFC9700] for client authentication.

## 5.2. Sender Constraining Tokens

Authorization Servers should follow the Best Current Practice for OAuth 2.0 Security [RFC9700] for sender constraining tokens, acknowledging, however, that bearer tokens remain the predominantly deployed access token type.

## 5.3. Authorized use of Subject Token

The authorization server in trust domain A can perform client authentication and verify that the client in trust domain A is authorized to present the token used as a subject_token in the token exchange flow before issuing an authorization grant. By doing so, it minimizes the risk of an attacker making a lateral move by using a stolen token from trust domain A to obtain an authorization grant with which to authenticate to an authorization server in trust domain B and request an access token for a resource server in trust domain B. Such authorization policy might not be present in all deployments, and is of reduced utility for public clients, but it is a recommended security measure for deployments that can support it.

## 5.4. Refresh Tokens

The authorization server in trust domain B SHOULD NOT issue refresh tokens to the client in trust domain A within the scope of this specification. When the access token has expired, clients can re-submit the original JWT Authorization Grant (if not expired and reuse is allowed) to obtain a new Access Token. If the JWT Authorization Grant is unusable, the client can request a new grant from the authorization server in trust domain A before presenting it to the authorization server in trust domain B. The issuance of Refresh Tokens by the authorization server in trust domain B introduces a redundant credential requiring additional security measures, and creating unnecessary security risks. It also allows the client to obtain access tokens within trust domain B, even if the initial session in trust domain A has finished (e.g. the user has logged out or access has been revoked). This is consistent with Section 4.1 of [RFC7521] which discourages but does not prohibit the issuance of refresh tokens in the context of assertion grants.

The advice of this section is only applicable to refresh token issuance across domains in the context of an assertion grant. It does not relate to the issuance of refresh tokens by the authorization server in trust domain A to the client in trust domain A.

## 5.5. Replay of Authorization Grant

The authorization grant obtained from the Token Exchange process is a bearer token. If an attacker obtains an authorization grant issued to a client in trust domain A, it could replay it to an authorization server in trust domain B to obtain an access token. Implementations must evaluate this risk and deploy appropriate mitigations based on their threat model and deployment environment. Mitigations include, but are not limited to:

* Issuing short-lived authorization grants to minimize the window of exposure.

* Limiting authorization grants to a single use to prevent repeated replay.

* Requiring client authentication to ensure the client presenting the grant is known to the authorization server in trust domain B.

Authorization servers in trust domain B can enforce these mitigations.

Implementations and profiles of this specification MAY define additional mitigations tailored to specific use cases and operational contexts.

# 6. Privacy Considerations

In addition to the privacy considerations outlined in [RFC8693] and [RFC7523], additional privacy considerations apply to this specification.

This specification enables the exchange of tokens and claims between disparate trust domains. If excessive or unnecessary user data is included in these tokens, it may lead to unintended privacy consequences. As noted in [RFC8693] and [RFC7523], deployments should determine the minimum amount of information necessary to complete the exchange and ensure that only that information is included in the token.

Inconsistent user privacy practices can result from varying interpretations and implementations of the protocol across authorization servers in different trust domains. This inconsistency can lead to a lack of transparency and user control over what data is shared and with whom. To mitigate this, trust relationships between domains must be carefully established and maintained with user privacy in mind. This includes verifying that privacy policies are aligned across trust domains and clearly define how user data is collected, used, and protected.

# 7. References

## 7.1. Normative References

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

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

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

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

[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/rfc/rfc8707>.

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

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

[RFC2046] Freed, N. and N. Borenstein, "Multipurpose Internet Mail Extensions (MIME) Part Two: Media Types", RFC 2046, DOI 10.17487/RFC2046, November 1996, <https://www.rfc-editor.org/rfc/rfc2046>.

[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/rfc/rfc6838>.

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

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

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

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

## 7.2. Informative References

[IANA.media-types] IANA, "Media Types", <https://www.iana.org/assignments/media-types>.

## Appendix A. Use cases

This appendix outlines some use cases where the identity and authorization chaining described in this document can be applied. The use cases described are not exhaustive, but are representative of the type of use cases enabled by this specification.  Other use cases may also be supported by this specification.

### A.1. Preserve User Context across Multi-cloud, Multi-Hybrid environments

A user attempts to access a service that is implemented as a number of on-premise and cloud-based workloads. Both the on-premise and cloud-based services are segmented by multiple trust boundaries that span one or more on-premise or cloud service environments. Each workload can apply an authorization policy that takes the context of the original user, as well as intermediary services into account, irrespective of where the workloads are running and even when a workload in one trust domain calls another service in another trust domain.

### A.2. Continuous Integration Accessing External Resources

A continuous integration system needs to access external resources, for example to upload an artifact or to run tests. These resources are protected by different authorization servers. The identity information of the build, for example metadata such as commit hashes or repository, should be preserved and carried across the domain boundary. This not just prevents maintaining credentials it also allows fine-grained access control at the resource.

### A.3. API Security Use Case

A home devices company provides a "Camera API" to enable access to home cameras. Partner companies use this Camera API to integrate the camera feeds into their security dashboards. Using OAuth between the partner and the Camera API, a partner can request the feed from a home camera to be displayed in their dashboard. The user has an account with the camera provider. The user may be logged in to view the partner provided dashboard, or they may authorize emergency access to the camera. The home devices company must be able to independently verify that the request originated and was authorized by a user who is authorized to view the feed of the requested home camera.

### A.4. Extend Single Sign-On to API Access

A user that authenticated to an enterprise Identity Provider (IdP) does not have to sign in to multiple SaaS applications if the SaaS applications are configured to trust the enterprise IdP. It is possible to extend this SSO relationship to API access by allowing the Client to contact the enterprise IdP and exchange the identity assertion (ID Token or SAML Token) that it previously received from the enterprise IdP for an authorization grant. The authorization grant can be used to obtain an access token from the SaaS application’s authorization server, provided that a trust relationship has been established between the enterprise IdP which issues the authorization grant and the SaaS authorization server. As a result SaaS servers that trust the enterprise IdP do not require the user to complete an interactive delegated OAuth 2.0 flow to obtain an access token to access the SaaS provider's APIs.

### A.5. Cross-domain API authorization

An email client can be used with arbitrary email servers, without requiring pre-established relationships between each email client and each email server. An email client obtains an identity assertion (ID Token or SAML token) from an IdP. When the email client needs access to a separate API, such as a third-party calendaring application, the email client exchanges the identity assertion for an authorization grant and uses this authorization grant to obtain an access token for the third-party calendaring application from the authorization server trusted by the third-party calendaring application. If the authorization server trusts the issuer of the authorization grant, the email client obtains an access token without any additional user interaction.

## Appendix B. Examples

This appendix contains two examples, demonstrating how this specification may be used in different environments with specific requirements. The first example shows the resource server acting as the client and the second example shows the authorization server acting as the client.

### B.1. Resource server acting as client

As part of completing a request, a resource server in trust domain A may need to access a resource server in trust domain B. This requires the resource server in trust domain A to obtain an Access Token from an authorization server in trust domain B, which it may then present to the resource server in trust domain B. A resource server in trust domain A may use the flows described in this specification by assuming the role of a client when attempting to access the resource server in trust domain B. Resource servers may act as clients if the following is true:

* The resource server has the ability to determine the authorization server of the protected resource outside trust domain A.

* The authorization server in trust domain B is reachable by the resource server in trust domain A and is able to perform the appropriate client authentication (if required).

The flow would look like this:

```
 +-------------+       +---------------+     +-------------+ +---------+
 |Authorization|       |Resource Server|     |Authorization| |Protected|
 |Server       |       |Trust Domain A |     |Server       | |Resource |
 |Trust        |       |(acting as     |     |Trust        | |Trust    |
 |Domain A     |       | a client)     |     |Domain B     | |Domain B |
 +-------------+       +---------------+     +-------------+ +---------+
        |                     |                     |             |
        |                     |   (1) request protected resource  |
        |                     |      metadata                     |
        |                     | --------------------------------->|
        |                     | <- - - - - - - - - - - - - - - - -|
        |                     |                     |             |
        | (2) exchange token  |                     |             |
        |   [RFC 8693]        |                     |             |
        |<--------------------|                     |             |
        |                     |                     |             |
        | (3) <authorization  |                     |             |
        |        grant>       |                     |             |
        | - - - - - - - - -  >|                     |             |
        |                     |                     |             |
        |                     | (4) present         |             |
        |                     |  authorization      |             |
        |                     |  grant [RFC 7523]   |             |
        |                     |-------------------->|             |
        |                     |                     |             |
        |                     | (5) <access token>  |             |
        |                     |<- - - - - - - - - - |             |
        |                     |                     |             |
        |                     |              (6) access           |
        |                     | --------------------------------->|
        |                     |                     |             |
```

Figure 6: Resource server acting as client

The flow contains the following steps:

The resource server of trust domain A needs to access protected resource in trust domain B. It requires an access token to do so. In order to obtain the required access token, the resource server in trust domain A will act as a client.

1. The resource server (acting as a client) in trust domain A requests protected resource metadata from the resource server in trust domain B as described in [RFC9728]. It uses the resource metadata to discover information about the authorization server for trust domain B. This step MAY be skipped if discovery is not

needed and other means of discovery MAY be used. The protected resource in trust domain B returns its metadata along with the authorization server information in trust domain A.

2. Once the resource server (acting as a client) in trust domain A identified the authorization server for trust domain B, it requests a JWT authorization grant for the authorization server in trust domain B from the authorization server in trust domain A (its own authorization server). This happens via the token exchange protocol (See Token Exchange (Section 2.3)).

3. If successful, the authorization server in trust domain A returns a JWT authorization grant to the resource server (acting as client) in trust domain A.

4. The resource server (acting as client) in trust domain A presents the JWT authorization grant to the authorization server in trust domain B.

5. The authorization server in trust domain B uses claims from the JWT authorization grant to identify the user and establish additional authorization context. If access is granted, the authorization server in trust domain B returns an access token.

6. The resource server (acting as a client) in trust domain A uses the access token to access the protected resource in trust domain B.

### B.2. Authorization server acting as client

Authorization servers may act as clients too. This can be necessary because of following reasons:

* Clients in trust domain A may not have knowledge of authorization servers in trust domain B.

* Clients in trust domain A may not have network access to other authorization servers in trust domain B.

* Strict access control on resources in trust domain B is required. This access control is enforced by authorization servers in trust domain B.

* Authorization servers in trust domain B require client authentication, but are unable to manage clients outside of trust domain B.

Under these conditions, an authorization server in trust domain A may obtain an access token from an authorization server in trust domain B on-behalf-of any client in trust domain A.  This enables clients in trust domain A to access a protected resource server in trust domain B.  Resource servers in trust domain A may act as a client to the authorization server in trust domain A in order to obtain an access token to access a protected resource in trust domain B in order to complete a request.

The authorization server in trust domain A may use the flows described in this specification by acting first as a client to itself to obtain an assertion grant and then act as a client to the authorization server in trust domain B to request an access token for a protected resource in trust domain B.  The flow when authorization servers act as a client on-behalf of another client in its own trust domain is shown below:

```
 +--------+          +--------------+       +--------------+ +---------+
 |Client  |          |Authorization |       |Authorization | |Protected|
 |Trust   |          |Server        |       |Server        | |Resource |
 |Domain A|          |Trust Domain A|       |Trust Domain B| |Trust    |
 |        |          |(acting as    |       |              | |Domain   |
 |        |          |client)       |       |              | |         |
 +--------+          +--------------+       +--------------+ +---------+
     |                      |                       |             |
     | (1) request          |                       |             |
     | token for            |                       |             |
     | protected resource   |                       |             |
     | in trust domain B.   |                       |             |
     | -------------------->|                       |             |
     |                      |                       |             |
     |                      |----+                  |             |
     |                      |    | (2) determine    |             |
     |                      |<---+ authorization    |             |
     |                      |      server trust     |             |
     |                      |      domain B         |             |
     |                      |                       |             |
     |                      |----+                  |             |
     |                      |    | (3) generates    |             |
     |                      |<---+ authorization    |             |
     |                      |      grant            |             |
     |                      |                       |             |
     |                      | (4) present           |             |
     |                      |   authorization grant |             |
     |                      |   [RFC 7523]          |             |
     |                      | --------------------->|             |
     |                      |                       |             |
     |                      | (5) <access token>    |             |
     |                      | <- - - - - - - - - - -|             |
     |                      |                       |             |
     |  (6) <access token>  |                       |             |
     | <- - - - - - - - - - |                       |             |
     |                      |                       |             |
     |                      |           (7) access  |             |
     | ---------------------------------------------------------->|
     |                      |                       |             |
```

Figure 7: Authorization server acting as client

The flow contains the following steps:

1. The client in trust domain A requests a token for the protected resource in trust domain B from the authorization server in trust domain A. This specification does not define this step. A profile of Token Exchange [RFC8693] may be used.

2. The authorization server for trust domain A determines the authorization server for trust domain B. This could have been passed by the client, is statically maintained or dynamically resolved.

3. Once the authorization server in trust domain B is determined, the authorization server in domain A generates a JWT authorization grant suitable for presentations to the authorization server in trust domain B.

4. The authorization server in trust domain A acts as a client and presents the JWT authorization grant to the authorization server for trust domain B. This presentation happens between the authorization servers. The authorization server in trust domain A may be required to perform client authentication while doing so. This reflects the Access Token Request (Section 2.4.1) in this specification.

5. The authorization server of trust domain B returns an access token for the protected resource in trust domain B to the authorization server (acting as a client) in trust domain A.

6. The authorization server of trust domain A returns the access token to the client in trust domain A.

7. The client in trust domain A uses the received access token to access the protected resource in trust domain B.

### B.3. Delegated Key Binding

In some environments, there is a need to bind the access token issued by the authorization server in trust domain B to a private key held by the client in trust domain A. This is so that the resource server in trust domain B can verify the proof of possession of the private key of the client in trust domain A when the client in trust domain A presents the token to the resource server in trust domain B. Any application in trust domain A may act as a client, including applications that are resource servers in trust domain A and need to access resource servers in trust domain B in order to complete a request.

In the case where the resource server in trust domain A is acting as the client, the access token may be constrained using existing techniques as described in Security Considerations (See Sender Constraining Tokens (Section 5.2)).

The case where the authorization server in trust domain A is acting as a client is more complicated since the authorization server in trust domain A (acting as client) does not have access to the key material of the client on whose behalf the access token is being requested.

However, the trust relationship between the authorization server in trust domain A and the authorization server in trust domain B can be leveraged to sender constrain the access token issued by the authorization server in trust domain B.  This can be achieved as follows.

* The authorization server in trust domain A verifies proof of possession of the key presented by the client.

* The authorization server in trust domain A then conveys the key of the client in trust domain A in the token request sent to the authorization server in trust domain B.  This can, for example, be accomplished by including a "requested_cnf" claim that contains the "cnf" claim of the client in trust domain A, in the assertion authorization grant sent to the authorization server in trust domain B.

* The authorization server in trust domain B then includes a "cnf" claim that matches the value of the "requested_cnf" claim included in the authorization grant in the returned access token.

* The client in trust domain A that presents the access token must use the key matching the "cnf" claim to generate a DPoP proof or set up a MTLS session when presenting the access token to a resource server in trust domain B.

## Appendix C.  Acknowledgements

The editors would like to thank Deb Cooley, Lars Eggert, Patrick Harding, Russ Housley, Joe Jubinski, Watson Ladd, Justin Richer, Adam Rusbridge, and Dean H.  Saxe for their valuable input and general support of this work.

## Appendix D.  Document History

[[ To be removed from the final specification ]]

-12

* GENART review comments addressed

-11

* ARTART review comments addressed

-10

* Move Aaron Parecki from contributors to authors in acknowledgement of significant contributions

-09

* AD comments (hopefully) addressed

-08

* Change some references from informative to normative and remove the unused OAuth 2.1 one

-07

* Add a (hopefully helpful) sentence to the end of the first paragraph of the Overview

* Reword bullet (C) of the Overview (because you cannot use public keys to sign)

* Explicitly reference RFC8693 Section 2.2.2 for token exchange error

* Try and better explain that the access token request content is more desricption of Sec 2.1 RFC7523 and delete the empty scope parameter

* Explicitly reference RFC7523 Section 3.1 for authorization grant error

* Remove a seemingly nonsensical sentence about preventing injection of invalid claims

* Try and explain why ASs might not want to advertise some supported requested token types

* Endeavor to qualify the SHOULDs on client auth and sender constrained tokens

* Qualify the only SHOULD NOT on RTs from assertion grants being inline with historical decisions in RFC7521

* Quality the Authorized use of Subject Token security recommendations a bit

* Change Intended Status to Standards Track from Informational

-06

* Use IANA.media-types so the tooling can find the media types registry without an explicit target

* Mention that the RFC8693 token exchange is not strictly necessary, if trust domain A’s platform provides other means to obtain a JWT authorization grant

* Better describe the trust relationship necessary (domain B has to trusts domain A to issue JWT authz grants and trust its signing key(s)) and mention that AS Metadata’s jwks_uri can be used to obtain the verification keys for trust domain A

* add a note about agreeing on semantics etc. when transcribing claims

* Editorial fixes

-05

* Editorial pass on Appendix for consistency

* Clarified introduction

* Added security considerations for unconstrained authorization grants.

* Updated some contributors’ affiliation and contact information

* Added examples in claims transcription text

* Simplify some text in the JWT Authorization Grant section

* Fix some toolchain complaints and other nitpicks

* Added some Privacy Considerations

* Move Mr. Parecki from acknowledgements to contributors in acknowledgement of his contributions

* Added Authorization Server Metadata registry to publish supported Token Exchange requested token types

-04

* Clarified diagrams and description of authorization server acting as a client.

* Remove references to sd-jwt.

* Added text to recommend use of explicit typing.

* Added security consideration on preventing lateral moves.

* Editorial updates to be consistent about the trust domain for a client, authorization server or resource server.

* Added sender constraining of tokens to security considerations

-03

* Editorial updates

-02

* remove recommendation to not use RFC8693’s requested_token_type

* Corrected discrepancy between alphabetic numbering of the diagram and text in the resource acting as client example

-01

* limit the authorization grant format to RFC7523 JWT

* minor example fixes

* editorial fixes

* added Aaron Parecki to acknowledgements

* renamed section headers to be more explicit

* use more specific term "JWT authorization grant"

* changed name to "OAuth Identity and Authorization Chaining Across Domains"

* move use cases to appendix and add continuous integration use case

-00

* initial working group version (previously draft-schwenkschuster-oauth-identity-chaining)

## Contributors

Atul Tulshibagwale
SGNL
Email: atuls@sgnl.ai

George Fletcher
Practical Identity LLC
Email: george@practicalidentity.com

Rifaat Shekh-Yusef
Ciena
Email: rifaat.s.ietf@gmail.com

Hannes Tschofenig
Email: hannes.tschofenig@gmx.net

## Authors' Addresses

Arndt Schwenkschuster
Defakto Security
Email: arndts.ietf@gmail.com

Pieter Kasselmann
Defakto Security
Email: pieter@defakto.security

Kelley Burgin
MITRE
Email: kburgin@mitre.org

Mike Jenkins
NSA-CCSS
Email: mjjenki@cyber.nsa.gov

Brian Campbell
Ping Identity
Email: bcampbell@pingidentity.com

Aaron Parecki
Okta
Email: aaron@parecki.com