oauth
Internet-Draft
Intended status: Informational
Expires: 23 April 2026

R. Li
H. Wang
Huawei Int. Pte Ltd
C. Liu
Huawei Technologies
T. Li
Huawei Int. Pte Ltd

20 October 2025

OAuth 2.0 Delegated Authorization
draft-li-oauth-delegated-authorization-00

Abstract

Delegated authorization enables a client to delegate a subset of its granted privileges to a subordinate access token (also known as a delegated access token). This mechanism allows the client to securely delegate authorization to a delegated party while maintaining fine-grained control over delegated permissions.

About This Document

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

Status information for this document may be found at https://datatracker.ietf.org/doc/draft-li-oauth-delegated-authorization/.

Discussion of this document takes place on the WG Working Group mailing list (mailto:oauth@ietf.org), which is archived at https://datatracker.ietf.org/wg/oauth/about/. Subscribe at https://www.ietf.org/mailman/listinfo/oauth/.

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 23 April 2026.

Table of Contents

1. Introduction 3
2. Requirements Language 3
3. Terminology 3
4. Overview 4
5. Delegated Party Metadata 6
5.1. Delegated Party Metadata Attributes 6
5.2. Delegated Party Metadata Example 7
5.3. WWW-Authenticate 8
6. Acquiring Delegation Tokens 9
6.1. Authorization Code Grant 9
6.2. Other Grant Types 10
7. Creating Delegated Access Tokens 10
8. Using Delegated Access Tokens 10
9. Verification of Delegated Access Tokens 11
9.1. Local Verification 11
9.2. Token Introspection 12
10. Security Considerations 12
11. Operational Considerations 12
12. IANA Considerations 13
12.1. Well-Known URIs Registry 13
12.2. OAuth Parameters Registry 13
12.3. OAuth Access Token Types Registry 14
12.4. HTTP Field Name Registry 14
12.5. OAuth Delegated Party Metadata Registry 14
12.5.1. Registration Template 14
12.5.2. Initial Registry Contents 15
13. References 16
13.1. Normative References 16
13.2. Informative References 18
Appendix A. Token Format 18

A.1. Example 1 18
A.2. Example 2 20
Appendix B. Use Cases 21
B.1. Delegating Subset of Access Rights to Specialized AI Agents 21
B.2. Third-Party Analytics Platform Integrated in an Enterprise SaaS 22
Authors' Addresses 23

1. Introduction

OAuth 2.0 [RFC6749] provides a framework for authorizing third-party applications to access protected resources on behalf of a resource owner. However, in existing implementations, access tokens issued to clients often contain excessive permissions that exceed actual requirements, creating security vulnerabilities and potential data exposure risks.

This specification extends OAuth 2.0 with a delegated authorization framework that enables clients to create subordinate access tokens with restricted permissions. This approach addresses the problem of over-privileged access tokens by implementing a two-token architecture that decouples initial authorization from final resource access.

2. 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.

3. Terminology

This specification uses the following terms defined in OAuth 2.0 [RFC6749]: authorization server, client, resource server, and resource owner.

The following additional terms are used throughout this document:

Delegated Party (DP): An entity (e.g., a service, component, or application) authorized by the client to access protected resources on behalf of the resource owner.

Delegated Resource: A resource or API endpoint hosted by the delegated party that requires access to the resource owner's protected data at a target resource server.

Delegation Token: A token issued by the authorization server for the client that enables the client to create delegated access tokens.

Delegated Access Token: A token created by the client using the delegation token, with permissions being a subset of the delegation token’s privileges and a more limited lifespan.

Delegation Key: A cryptographic key bound to the delegation token, used by the client to sign or encrypt delegated access tokens. The delegation key is presented in the token request as the delegation_key parameter.

4. Overview

The delegated authorization framework introduces a hierarchical token structure where a client can obtain a delegation token from an authorization server and use it to issue subordinate access tokens with reduced permissions. This enables fine-grained access control while maintaining the security properties of the original authorization grant.

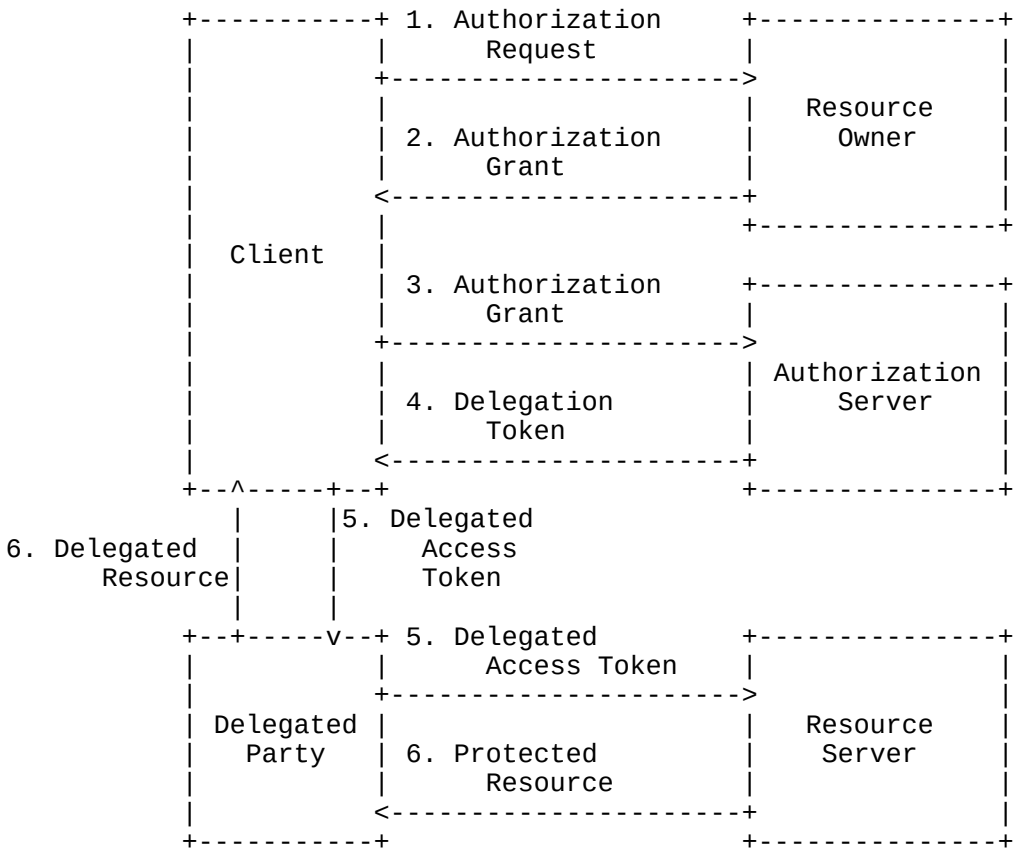

Figure 1: Delegated Authorization Framework Architecture

1. The client requests authorization from the resource owner. The client indicates in the authorization request that the requested authorization grant is for delegated authorization.

2. The client receives an authorization grant.

3. The client requests a delegation token by authenticating with the authorization server and presenting the authorization grant and its delegation key as defined in Section 3.

4. The authorization server authenticates the client and validates the authorization grant, and if valid, issues a delegation token.

5. The client calls the delegated party’s API, presenting the delegated access token generated from the delegation token. The delegated access token is issued by the client using the

delegation key. The delegated party requests the target protected resource from the resource server and presents the delegated access token.

6. The resource server validates the delegated access token, and if valid, serves the resource. The delegated party receives the resource, optionally transforms it into a service-specific response (also known as delegated resource), and returns it to the client.

Both delegation token and delegated access token can be JSON Web Tokens (JWTs) [RFC7519] or CBOR Web Tokens (CWTs) [RFC8392].

5. Delegated Party Metadata

Before the OAuth 2.0 client retrieves a delegation token and generates a delegated access token for the delegated party, the client needs to obtain the authorization server endpoint and the permissions needed by the delegated party. Such information can be manually configured into the client, or it can be dynamically discovered through delegated party metadata.

Delegated party metadata enables OAuth 2.0 clients to obtain information needed to interact with a delegated party. The structure of the metadata format is similar to "OAuth 2.0 Authorization Server Metadata" [RFC8414] and "OAuth 2.0 Protected Resource Metadata" [RFC9728].

The delegated party metadata is retrieved from a well-known [RFC8615] location as a JSON [RFC8259] document. By default, the well-known URI string used is /.well-known/oauth-delegated-party.

5.1. Delegated Party Metadata Attributes

resources: *RECOMMENDED*. JSON array containing a list of target protected resources' resource identifiers, as defined in [RFC9728]. Either this attribute or *authorization_servers* defined below MUST be present.

authorization_servers: *OPTIONAL*. JSON array containing a list of OAuth authorization server issuer identifiers, as defined in [RFC8414]. Either this attribute or *resources* defined above MUST be present.

permissions_supported: *RECOMMENDED*. JSON object indicating the permissions the delegated party may request. The scopes attribute lists supported scope values [RFC6749]; the authorization_details attribute lists supported rich authorization request objects as

defined in [RFC9396]. These guide the client in constructing valid authorization and token requests. Either the scopes attribute or the authorization_details attribute must be present.

api_permissions: *RECOMMENDED*. JSON object mapping API endpoints (resource identifiers) to the permissions required to access them. Each value can include scopes and/or authorization_details to specify the permissions needed for that endpoint/resource.

delegated_party_documentation: *OPTIONAL*: URL of a page containing human-readable information that developers might want or need to know when using the delegated party. The value of this field MAY be internationalized.

5.2. Delegated Party Metadata Example

The following is a non-normative example delegated party metadata:

```
{
  "resources": [
    "https://res1.example.com",
    "https://res2.example.net"
  ],
  "authorization_servers": [
    "https://as1.example.com",
    "https://as2.example.net"
  ],
  "permissions_supported": {
    "scopes": ["profile:read", "profile:write", "email:read", "email:write"],
    "authorization_details": [
      {
        "type": "payment",
        "actions": ["initiate", "status", "cancel"],
        "instructedAmount": {
          "notMoreThan": {
            "currency": "USD",
            "amount": "500.00"
          }
        }
      }
    ]
  },
  "api_permissions": {
    "/emails/list": {
      "scopes": ["email:read"]
    },
    "/balance/transfer": {
      "authorization_details": [
```

```
            {
              "type": "payment",
              "actions": ["initiate"],
              "instructedAmount": {
                "notMoreThan": {
                  "currency": "USD",
                  "amount": "500.00"
                }
              }
            }
          ]
        },
        "/balance/transfer/status": {
          "authorization_details": [
            {
              "type": "payment",
              "actions": ["status"]
            }
          ]
        },
        "/balance/transfer/cancel": {
          "authorization_details": [
            {
              "type": "payment",
              "actions": ["cancel"]
            }
          ]
        }
      },
      "delegated_party_documentation": "https://dp.example.com/dp_documentation.html"
    }
```

5.3. WWW-Authenticate

Upon receipt of a request for a delegated resource that lacks credentials, the delegated party can reply with a challenge using the 401 (Unauthorized) status code ([RFC9110] Section 15.5.2) and the WWW-Authenticate header field ([RFC9110] Section 11.6.1).

This specification introduces a new parameter in the WWW-Authenticate HTTP response header field to indicate the delegated party metadata URL:

delegated_party_metadata: The URL of the delegated party metadata.

The response below is an example of a WWW-Authenticate header that includes the delegated party metadata URL. NOTE: '\' line wrapping per [RFC8792].

```
HTTP/1.1 401 Unauthorized
WWW-Authenticate: Bearer delegated_party_metadata=\
  "https://dp.example.com/.well-known/oauth-delegated-party"
```

6. Acquiring Delegation Tokens

The client requests a delegation token using standard OAuth 2.0 grant types with additional parameters to distinguish delegation requests from standard token requests.

6.1. Authorization Code Grant

For authorization code grant type, the client MUST include a delegation=true parameter in the authorization request to indicate that the client is requesting a delegation token instead of an OAuth 2.0 access token.

Additionally, the authorization request MUST include either a scope parameter (as defined in [RFC6749] Section 3.3), an authorization_details parameter (as defined in "Rich Authorization Requests" [RFC9396]), or both, that define the permissions granted to the requested delegation token.

In the token request, the client MUST include a delegation_key parameter with the value of the delegation key. This is normally a public key for digital signature. It can be extended to support encryption public keys, or secret keys for MAC or symmetric encryption.

Additionally, the client MAY include in the token request either a scope parameter, an authorization_details parameter, or both. The client MAY also include a delegation=true parameter in the token request.

In the token response, the authorization server MUST include an access_token attribute whose value is the delegation token, and MUST include a token_type attribute valued "Delegation", and MAY include a refresh_token attribute which is the refresh token for obtaining a new delegation token via the refresh token grant.

Other procedures of the authorization code grant are as described in [RFC6749]. Use of Proof Key for Code Exchange (PKCE) [RFC7636] is RECOMMENDED.

6.2. Other Grant Types

Other OAuth 2.0 grant types, such as the refresh token grant or client credentials grant, MAY support delegated authorization by including the delegation and delegation_key parameters when applicable. The authorization server MUST validate that the client is authorized to request delegation tokens using the given grant type.

7. Creating Delegated Access Tokens

The client creates delegated access tokens by:

1. Validating the delegation token’s validity and permissions.
2. Generating a subordinate access token with (optionally) reduced privileges.
3. Applying cryptographic protection using the delegation key (digital signature, encryption or MAC).

The client MUST include the delegation token in the delegation_token attribute of the delegated access token.

The client MUST ensure that the delegated access token’s scope, lifetime, audience, and other claims do not exceed those of the delegation token. The client MAY generate single-use delegated access tokens that the resource server or authorization server only consider valid when validating it for the first time.

The client is RECOMMENDED to "sender-constrain" the delegated access tokens by binding the delegated access tokens with public keys or certificates where the corresponding private keys are owned by the delegated parties, via techniques similar to OAuth 2.0 mTLS [RFC8705] or OAuth 2.0 DPoP [RFC9449].

8. Using Delegated Access Tokens

When the client accesses a delegated resource on the delegated party, the client MUST include the delegated access token as a bearer token [RFC6750] in the Delegated-Authorization header, used by the target resource server to verify requests from the delegated party. The Delegated-Authorization header MAY be used in combination with an Authorization header used by the delegated party to verify the request from the client.

For example:

```
GET /dp-resource HTTP/1.1
Host: delegated-party.example.com
Authorization: Bearer mF_9.B5g1234
Delegated-Authorization: Bearer mF_9.B5f-4.1JqM
```

Upon receiving a delegated resource request with a Delegated-Authorization header, the delegated party sends a request to the target resource server for the respective target resource. The delegated party MUST include the received delegated access token as a bearer token in the Authorization header.

For example:

```
GET /target-resource HTTP/1.1
Host: resource.example.com
Authorization: Bearer mF_9.B5f-4.1JqM
```

9. Verification of Delegated Access Tokens

Resource servers verify delegated access tokens through either local validation using pre-configured public keys or remote validation via token introspection [RFC7662] at the authorization server.

9.1. Local Verification

The resource server verifies delegated access tokens by:

1. The resource server is pre-configured with the authorization server’s public key, or it fetches the public key via the authorization server’s JWKS endpoint [RFC7517].

2. Checking the digital signature of the delegation token (part of the delegated access token) against the authorization server’s public key.

3. Checking the digital signature of the delegated access token against the delegation key bound to the delegation token.

4. Verifying the delegated access token’s permissions and validity are within the scope of the delegation token.

5. Verifying the delegated access token is within validity period, and the delegated access token’s permissions cover the resource request.

9.2. Token Introspection

The resource server sends the delegated access token to the authorization server via the token introspection endpoint. The authorization server verifies the delegated access token against its keys.

10. Security Considerations

This specification extends OAuth 2.0 to support delegated authorization through hierarchical token issuance. While this enables fine-grained privilege delegation, it also introduces new trust and security considerations.

* Delegation tokens MUST NOT be sent to resource servers without subordinate delegated access tokens. Resource servers and authorization servers MUST NOT treat delegation tokens as regular OAuth access tokens.

* Clients MUST protect the delegation key, as compromise allows an attacker to mint valid delegated access tokens within the scope of the delegation token.

* Delegated access tokens SHOULD have short lifetimes and be bound to specific audiences, methods, and sender keys (e.g., via DPoP or mTLS) to mitigate replay and token leakage risks. Resource servers MUST validate both the delegation token and the delegated access token, ensuring the latter does not exceed the former’ s permissions.

* Token introspection CAN be used in scenarios where the tokens are kept opaque from the delegated party and the resource server. If employed, token introspection responses MUST NOT reveal sensitive internal information. Authorization servers SHOULD enforce rate limiting and audit token issuance and validation activities.

11. Operational Considerations

Deployments of this specification should consider the following operational aspects:

* *Key Management*: Clients MUST securely store and rotate delegation keys. Authorization servers SHOULD support key rotation for delegation tokens and provide mechanisms to revoke compromised keys.

* *Token Lifetimes*: Delegation tokens SHOULD have longer lifetimes than delegated access tokens to reduce authorization server load, and SHOULD be refreshable using refresh tokens.

* *Metadata Caching*: Delegated party metadata at the well-known URI /.well-known/oauth-delegated-party can be cached by clients; a reasonable default TTL (e.g., 24 hours) is RECOMMENDED.

* *Error Handling*: Delegated parties and resource servers SHOULD provide clear error responses (e.g., invalid token, insufficient scope) without exposing implementation details.

* *Interoperability*: Implementers SHOULD ensure compatibility with existing OAuth 2.0 features such as PKCE, Rich Authorization Requests, and sender-constrained tokens.

12. IANA Considerations

12.1. Well-Known URIs Registry

This specification registers the following entry in the "Well-Known URIs" registry:

* *URI suffix*: oauth-delegated-party

* *Reference*: [this document]

* *Status*: permanent

* *Change controller*: IETF

* *Related information*: (none)

12.2. OAuth Parameters Registry

This specification registers the following parameters in the "OAuth Parameters" registry:

Delegation:

* *Name*: delegation

* *Parameter Usage Location*: authorization request, token request

* *Change Controller*: IETF

* *Reference*: [this document]

Delegation Key:

* *Name*: delegation_key

* *Parameter Usage Location*: token request

* *Change Controller*: IETF

* *Reference*: [this document]

12.3. OAuth Access Token Types Registry

This specification registers the following parameters in the "OAuth Access Token Types" registry:

* *Name*: Delegation

* *Additional Token Endpoint Response Parameters*: (none)

* *HTTP Authentication Scheme(s)*: Bearer

* *Change Controller*: IETF

* *Reference*: [this document]

12.4. HTTP Field Name Registry

This specification registers the following parameters in the "Hypertext Transfer Protocol (HTTP) Field Name" registry:

* *Field Name*: Delegated-Authorization

* *Status*: permanent

* *Structured Type*: (none)

* *Reference*: [this document]

12.5. OAuth Delegated Party Metadata Registry

This specification establishes the "OAuth Delegated Party Metadata" registry for OAuth 2.0 dalagated party metadata names. The registry records the dalagated party metadata parameter and a reference to the specification that defines it.

12.5.1. Registration Template

Metadata Name: The name requested (e.g., "resource"). This name

is case sensitive. Names may not match other registered names in a case-insensitive manner unless the designated experts state that there is a compelling reason to allow an exception.

Metadata Description: Brief description of the metadata (e.g., "Resource identifier URL").

Change Controller: For IETF Stream RFCs, list "IETF". For others, give the name of the responsible party. Other details (e.g., postal address, email address, home page URI) may also be included.

Specification Document(s): Reference to the document or documents that specify the parameter, preferably including URIs that can be used to retrieve copies of the documents. An indication of the relevant sections may also be included but is not required.

12.5.2. Initial Registry Contents

Resources:

* *Metadata Name*: resources

* *Metadata Description*: JSON array containing a list of target protected resources' resource identifier URLs

* *Change Controller*: IETF

* *Specification Document(s)*: [this document]

Authorization Servers:

* *Metadata Name*: authorization_servers

* *Metadata Description*: JSON array containing a list of authorization server issuer identifiers

* *Change Controller*: IETF

* *Specification Document(s)*: [this document]

Supported Permissions:

* *Metadata Name*: permissions_supported

* *Metadata Description*: JSON object indicating the permissions the delegated party may request

* *Change Controller*: IETF

* *Specification Document(s)*: [this document]

API Permissions:

* *Metadata Name*: api_permissions

* *Metadata Description*: JSON object mapping API endpoints (resource identifiers) to the permissions required to access them

* *Change Controller*: IETF

* *Specification Document(s)*: [this document]

Delegated Party Documentation:

* *Metadata Name*: delegated_party_documentation

* *Metadata Description*: URL of a page containing human-readable information that developers might want or need to know when using the delegated party

* *Change Controller*: IETF

* *Specification Document(s)*: [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/rfc/rfc2119>.

[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>.

[RFC6750] Jones, M. and D. Hardt, "The OAuth 2.0 Authorization Framework: Bearer Token Usage", RFC 6750, DOI 10.17487/RFC6750, October 2012, <https://www.rfc-editor.org/rfc/rfc6750>.

[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/rfc/rfc7515>.

[RFC7516] Jones, M. and J. Hildebrand, "JSON Web Encryption (JWE)", RFC 7516, DOI 10.17487/RFC7516, May 2015, <https://www.rfc-editor.org/rfc/rfc7516>.

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

[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/rfc/rfc7519>.

[RFC7636] Sakimura, N., Ed., Bradley, J., and N. Agarwal, "Proof Key for Code Exchange by OAuth Public Clients", RFC 7636, DOI 10.17487/RFC7636, September 2015, <https://www.rfc-editor.org/rfc/rfc7636>.

[RFC7662] Richer, J., Ed., "OAuth 2.0 Token Introspection", RFC 7662, DOI 10.17487/RFC7662, October 2015, <https://www.rfc-editor.org/rfc/rfc7662>.

[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>.

[RFC8259] Bray, T., Ed., "The JavaScript Object Notation (JSON) Data Interchange Format", STD 90, RFC 8259, DOI 10.17487/RFC8259, December 2017, <https://www.rfc-editor.org/rfc/rfc8259>.

[RFC8392] Jones, M., Wahlstroem, E., Erdtman, S., and H. Tschofenig, "CBOR Web Token (CWT)", RFC 8392, DOI 10.17487/RFC8392, May 2018, <https://www.rfc-editor.org/rfc/rfc8392>.

[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>.

[RFC8615] Nottingham, M., "Well-Known Uniform Resource Identifiers (URIs)", RFC 8615, DOI 10.17487/RFC8615, May 2019, <https://www.rfc-editor.org/rfc/rfc8615>.

[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/rfc/rfc8705>.

[RFC9110] Fielding, R., Ed., Nottingham, M., Ed., and J. Reschke, Ed., "HTTP Semantics", STD 97, RFC 9110, DOI 10.17487/RFC9110, June 2022, <https://www.rfc-editor.org/rfc/rfc9110>.

[RFC9396] Lodderstedt, T., Richer, J., and B. Campbell, "OAuth 2.0 Rich Authorization Requests", RFC 9396, DOI 10.17487/RFC9396, May 2023, <https://www.rfc-editor.org/rfc/rfc9396>.

[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/rfc/rfc9449>.

[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>.

13.2. Informative References

[RFC8792] Watsen, K., Auerswald, E., Farrel, A., and Q. Wu, "Handling Long Lines in Content of Internet-Drafts and RFCs", RFC 8792, DOI 10.17487/RFC8792, June 2020, <https://www.rfc-editor.org/rfc/rfc8792>.

Appendix A. Token Format

The example tokens in this section are shown in Flattened JSON Serialization [RFC7515] [RFC7516], un-base64url-encoded/unencrypted, and with comments for ease of reading. When used as JWTs [RFC7519], they should be represented in Compact Serialization [RFC7515] [RFC7516]. Similarly, they can be represented as CWTs [RFC8392].

A.1. Example 1

In this example, the delegation token is a JWS token signed with HS256, and the delegated access token is a JWS token signed with RS256.

Delegation Token:

```
   {
     "protected": {
       "_comment": "to be base64url-encoded",
       "alg": "HS256",
       "typ": "JWT",
       "kid": "as-key-1"
     },
     "payload": {
       "_comment": "to be base64url-encoded",
       "iss": "https://as1.example.com",
       "sub": "user@example.com",
       "aud": "https://res1.example.com",
       "iat": 1760946495,
       "exp": 1763538495,
       "scope": "email:read email:send",
       "delegation_key": {
         "kty": "RSA",
         "n": "xoGV-drpIhwQ9Q3M5ouoA4Y76j4r0c2YcJoPT2qUd8UxV1PZH61TGZUbdUAdQLqi7Pik3GwTk3
4b6Xxb2-UkW3zoaBx_2FXXfVWwSVbfxi4RCbFP-rWGlbyYTRILj6CJM5JXI8VQdcSF8yfPZVytw-aKU-5k4RddKxg
yMwkWNCShWPa_H2WRsDzcy88pE-8q1cg6hbaq5GTywdiSeGWrjMYebQqIN-V63bX2aiOHhFvPVpEoI7AlxlrQd7aJ
tFwfuRl-0FxJH-2ITrnHFZaFAdoJqvFSD3OKZNkECBpuDL-DHcZUZfEyr4Rvb3WB0iuHHfHXzhbzqAt3NbZalmdQN
w",
         "e": "AQAB"
       }
     },
     "signature": "1gR7TSa8ft8Wt4ZA9HuLFTYW2uAw86X2pFRrq9jDoQQ"
   }
```


Delegated Access Token:


```
   {
     "protected": {
       "_comment": "to be base64url-encoded",
       "alg": "RS256",
       "typ": "JWT",
       "kid": "delegation-key-1"
     },
     "payload": {
       "_comment": "to be base64url-encoded",
       "iss": "user@example.com",
       "sub": "https://dp1.example.com",
       "aud": "https://res1.example.com",
       "iat": 1760950095,
       "exp": 1760953695,
       "scope": "email:read",
       "delegationToken": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCIsImtpZCI6ImFzLWtleS0xIn0.ey
Jpc3MiOiJodHRwczovL2FzMS5leGFtcGxlLmNvbSIsInN1YiI6InVzZXJAZXhhbXBsZS5jb20iLCJhdWQiOiJodHR
wczovL3JlczEuZXhhbXBsZS5jb20iLCJpYXQiOjE3NjA5NDY0OTUsImV4cCI6MTc2MzUzODQ5NSwic2NvcGUiOiJl
bWFpbDpyZWFkIGVtYWlsOnNlbmQiLCJkZWxlZ2F0aW9uX2tleSI6eyJrdHkiOiJSU0EiLCJuIjoieG9HVi1kcnBJa
HdROVEzTTVvdW9BNFk3Nmo0cjBjMlljSm9QVDJxVWQ4VXhWMVBaSDYxVEdaVWJkVUFkUUxxaTdQaWszR3dUazM0Yj
ZYeGIyLVVrVzN6b2FCeF8yRlhYZlZXd1NWYmZ4aTRSQ2JGUC1yV0dsYnlZVFJJTGo2Q0pNNUpYSThWUWRjU0Y4eWZ
QWlZ5dHctYUtVLTVrNFJkZEt4Z3lNd2tXTkNTaFdQYV9IMldSc0R6Y3k4OHBFLThxMWNnNmhiYXE1R1R5d2RpU2VH
V3JqTVllYlFxSU4tVjYzYlgyYWlPSGhGdlBWcEVvSTdBbHhsclFkN2FKdEZ3ZnVSbC0wRnhKSC0ySVRybkhGWmFGQ
WRvSnF2RlNEM09LWk5rRUNCcHVETC1ESGNaVVpmRXlyNFJ2YjNXQjBpdUhIZkhYemhienFBdDNOYlphbG1kUU53Ii
wiZSI6IkFRQUIifX0.1gR7TSa8ft8Wt4ZA9HuLFTYW2uAw86X2pFRrq9jDoQQ"
     },
     "signature": "r5O4a3d3NMN7vZlOB9P4qPLbHyy12bZH5Ha3DZATa8NUdHYPJBMieiS1tqbhwfnTvwcSR_
0Ac42dRHDrk8MWPkCl_9-bfYNvf9eGrcwSRn7889-pleH-QphxZG4Tsr8m2WlGM8VFnC9sjkhqGvmM9ZdC02GEaip
tU9D859QU3-u-tvRdSgrHXeuLY2a3hgpWnj0j4gfgpug-VSvNB26vqCWwM4mwMGWYUPDabBaPp-KL38_M1T7q4wvF
E0_KLvTSdozMe1ngkLewvSTBrnW1rULhXFk54j381wQ_ovaJhM1aAbWshb1AMzu-aiv0EZJjB1XlgORVh-KbId01T
ZQLwA"
   }
```

A.2. Example 2

In this example, the delegation token is a JWE token encrypted with A128CBC-HS256, and the delegated access token is a JWS token signed with ES256.

Delegation Token:


```
{
  "protected": {
    "_comment": "to be base64url-encoded",
    "alg": "dir",
    "enc": "A128CBC-HS256",
    "typ": "JWT",
    "kid": "as-key-2"
  },
  "iv": "s99tD84KH1_kgbA2ArpUZg",
  "ciphertext": {
    "_comment": "to be encrypted",
    "iss": "https://as1.example.com",
    "sub": "user@example.com",
    "aud": "https://res1.example.com",
    "iat": 1760946495,
    "exp": 1763538495,
    "scope": "email:read email:send",
    "delegation_key": {
      "kty": "EC",
      "crv": "P-256",
      "x": "iZgQ3t5EK4rVdkex6LAGfxB0deOHtE3-vb-OBxoFv88",
      "y": "RQVYHkEWrTR6jckD7iHXnRRs60-u9ikSfVnM4epiOLY"
    }
  },
  "tag": "9Z4ONkwLwv3szT-eIKa4uQ"
}
```


Delegated Access Token:

```
   {
     "protected": {
       "_comment": "to be base64url-encoded",
       "alg": "ES256",
       "typ": "JWT",
       "kid": "delegation-key-2"
     },
     "payload": {
       "_comment": "to be base64url-encoded",
       "iss": "user@example.com",
       "sub": "https://dp1.example.com",
       "aud": "https://res1.example.com",
       "iat": 1760950095,
       "exp": 1760953695,
       "scope": "email:read",
       "delegationToken": "eyJhbGciOiJkaXIiLCJlbmMiOiJBMTI4Q0JDLUhTMjU2IiwidHlwIjoiSldUIi
wia2lkIjoiYXMta2V5LTIifQ..s99tD84KH1_kgbA2ArpUZg.DS5aM1uuKpFFacu7rIvKOYnA-6BRPy6jyZ-3uF8b
pplWJkCQmsAi8KqS-qTZuHWNfIWw7ulK-a8rWhfPPfLrgdXky6Ujc_3vm5YXRXmwxlaNJlhr2LexPXsTX2wA_3aIo
9c0b5kQB2MHqeI5ucJDSdERCV2AaQTgl7vRmJhZ_FAY5cnd2URHEsqnm5usrywzGMLw5CnXg2MBl1jWxiHp-PmzOm
RPHks4jFV2es2jjr-yB5PlX2d-OBCU2hauM_JjtnYOhByiXmAVVE6XKjJHXYM-d5q3JheTDg5gA4f1Io38_r2KA3p
W07CF94Nx3i7VRxaFRSVYuNxlEhUx0vxOIlwRBa3_ZOK4Kkg-og66ADs73RuBg91cCthbr63NfIdEXmmYKG2Nx3De
hojbVKZQrg.9Z4ONkwLwv3szT-eIKa4uQ"
     },
     "signature": "hlxIMOUb_Wdjh53VPcvuXBwTDiGzC7O8-ofV2LAvkws-LRIqKF6WRZ3KoPG1iTEDDhel3X
XAGCyfRFCMXH3KiQ"
   }
```


Appendix B. Use Cases

B.1. Delegating Subset of Access Rights to Specialized AI Agents

Enterprise Identity and Access Management systems often employ Role Based Access Control (RBAC) or Attribute Based Access Control (ABAC), assigning a set of minimal permissions to the employee based on its role, department, or other attributes. AI Agent can be an employee’s personal assistant, or a virtual employee of a certain department in general. An Agent’s delegated permissions CAN be long-termed, but MUST NOT be a direct inheritance of all its owner’s access right. Rather, they SHOULD be a subset of its owner, bound to specfic service/API/database/codebase according to its specialty and dedicated workflow.

Role	Service / Component
Resource Owner	an enterprise, individual or a department
Client	agent’s client application
Delegated Party	CI-CD agent, test agent, DEV agent, research agent
Authorization Server	enterprise IAM system
Resource Server	enterprise IT systems
Target Protected Resource	DEV/STAGE/PROD environments, internal knowledge database

Table 1: AI Agents

B.2. Third-Party Analytics Platform Integrated in an Enterprise SaaS

In this scenario, a corporate customer uses a Software-as-a-Service (SaaS) Customer Relationship Management (CRM) application. The customer wishes to gain business insights by granting a specialized third-party analytics platform limited access to its CRM data.

The CRM application obtains a delegation token from the enterprise’s identity provider. It then creates a narrowly scoped delegated access token for the analytics service. This token only permits read access to a predefined, non-sensitive subset of customer data (e.g., names and identifiers, but not personal email addresses). The analytics platform uses this token to pull data, generates an aggregated business intelligence report, and delivers it back to the CRM application for the corporate customer to view.

Role	Service / Component
Resource Owner	companyA (the tenant)
Client	SaaS CRM application
Delegated Party	analytics service
Authorization Server	enterprise IdP
Resource Server	CRM application server
Target Protected Resource	CRM application’s data retrieval API

Table 2: Enterprise-SaaS

Authors’ Addresses

Ruochen Li
Huawei Int. Pte Ltd
Email: li.ruochen@h-partners.com

Haiguang Wang
Huawei Int. Pte Ltd
Email: wang.haiguang.shieldlab@huawei.com

Chunchi Peter Liu
Huawei Technologies
Email: liuchunchi@huawei.com

Tieyan Li
Huawei Int. Pte Ltd
Email: Li.Tieyan@huawei.com