

OAuth Working Group
Internet-Draft
Intended status: Standards Track
Expires: October 1, 2017

B. Campbell
J. Bradley
Ping Identity
N. Sakimura
Nomura Research Institute
T. Lodderstedt
YES Europe AG
March 30, 2017

Mutual TLS Profiles for OAuth Clients
draft-campbell-oauth-mtls-00

Abstract

This document describes Transport Layer Security (TLS) mutual authentication using X.509 certificates as a mechanism for both OAuth client authentication to the token endpoint as well as for sender constrained access to OAuth protected resources.

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 <http://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 October 1, 2017.

Copyright Notice

Copyright (c) 2017 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<http://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must

include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1. Introduction	2
1.1. Requirements Notation and Conventions	3
2. Mutual TLS for Client Authentication	3
2.1. Mutual TLS Client Authentication to the Token Endpoint	3
2.2. Authorization Server Metadata	4
2.3. Dynamic Client Registration	4
3. Mutual TLS Sender Constrained Resources Access	4
3.1. X.509 Certificate SHA-256 Thumbprint Confirmation Method for JWT	5
4. IANA Considerations	6
4.1. JWT Confirmation Methods Registration	6
4.1.1. Registry Contents	6
4.2. Token Endpoint Authentication Method Registration	6
4.2.1. Registry Contents	6
4.3. OAuth Dynamic Client Registration Metadata Registration	6
4.3.1. Registry Contents	6
5. Security Considerations	7
5.1. TLS Versions and Best Practices	7
5.2. Client Identity Binding	7
6. References	7
6.1. Normative References	7
6.2. Informative References	8
Appendix A. Acknowledgements	9
Appendix B. Document(s) History	9
Authors' Addresses	10

1. Introduction

This document describes Transport Layer Security (TLS) mutual authentication using X.509 certificates as a mechanism for both OAuth client authentication to the token endpoint as well as for sender constrained access to OAuth protected resources.

The OAuth 2.0 Authorization Framework [RFC6749] defines a shared secret method of client authentication but also allows for the definition and use of additional client authentication mechanisms when interacting with the authorization server's token endpoint. This document describes an additional mechanism of client authentication utilizing mutual TLS [RFC5246] certificate-based authentication, which provides better security characteristics than shared secrets.

Mutual TLS sender constrained access to protected resources ensures that only the party in possession of the private key corresponding to the certificate can utilize the access token to get access to the associated resources. Such a constraint is unlike the case of the basic bearer token described in [RFC6750], where any party in possession of the access token can use it to access the associated resources. Mutual TLS sender constrained access prevents the use of stolen access tokens by binding the access token to the client's certificate.

Mutual TLS sender constrained access tokens and mutual TLS client authentication are distinct mechanisms that can don't necessarily need to be deployed together.

1.1. Requirements Notation and Conventions

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 RFC 2119 [RFC2119].

2. Mutual TLS for Client Authentication

2.1. Mutual TLS Client Authentication to the Token Endpoint

The following section defines, as an extension of OAuth 2.0, Section 2.3 [RFC6749], the use of mutual TLS as client credentials. The requirement of mutual TLS for client authentications is determined by the authorization server based on policy or configuration for the given client (regardless of whether the client was dynamically registered or statically configured or otherwise established). OAuth 2.0 requires that access token requests by the client to the token endpoint use TLS. In order to utilize TLS for client authentication, the TLS connection MUST have been established or reestablished with mutual X.509 certificate authentication (i.e. the Client Certificate and Certificate Verify messages are sent during the TLS Handshake [RFC5246]).

For all access token requests to the token endpoint, regardless of the grant type used, the client MUST include the "client_id" parameter, described in OAuth 2.0, Section 2.2 [RFC6749]. The presence of the "client_id" parameter enables the authorization server to easily identify the client independently from the content of the certificate and allows for trust models to vary as appropriate for a given deployment. The authorization server can locate the client configuration by the client identifier and check the certificate presented in the TLS Handshake against the expected credentials for that client. As described in Section 5.2, the

authorization server MUST enforce some method of binding a certificate to a client.

2.2. Authorization Server Metadata

"tls_client_auth" is used as a new value of the "token_endpoint_auth_methods_supported" metadata parameter to indicate server support for mutual TLS as a client authentication method in authorization server metadata such as [OpenID.Discovery] and [I-D.ietf-oauth-discovery].

2.3. Dynamic Client Registration

This draft adds the following values and metadata parameters to the OAuth 2.0 Dynamic Client Registration [RFC7591].

The value "tls_client_auth" is used to indicate the client's intention to use mutual TLS as an authentication method to the token endpoint for the "token_endpoint_auth_method" client metadata field.

For authorization servers that associate certificates with clients using subject information in the certificate, the following two new string metadata parameters can be used:

tls_client_auth_subject_dn The expected subject distinguished name of the client certificate can be represented using "tls_client_auth_subject_dn".

tls_client_auth_issuer_dn The metadata parameter "tls_client_auth_issuer_dn" can optionally be used to constrain the expected distinguished name of the root issuer of the client certificate.

For authorization servers that use the key or full certificate to associate clients with certificate, the existing "jwks_uri" or "jwks" metadata parameters from [RFC7591] shall be used.

3. Mutual TLS Sender Constrained Resources Access

When mutual TLS X.509 client certificate authentication is used at the token endpoint, the authorization server is able to bind the issued access token to the client certificate. Such a binding is accomplished by associating a hash of the certificate with the token in a way that can be accessed by the protected resource, such as embedding the certificate hash in the issued access token directly, using the syntax described in Section 3.1, or through token introspection [RFC7662]. The specific method for associating the certificate with the access token is determined by the authorization

server and the protected resource, and is beyond the scope for this specification.

The client makes protected resource requests as described in [RFC6750], however, those requests MUST be made over a mutually authenticated TLS connection using the same certificate that was used to authenticate to the token endpoint.

The protected resource MUST obtain the client certificate used for TLS authentication and MUST verify that the hash of that certificate exactly matches the hash of the certificate associated with the access token. If the hash values do not match, the resource access attempt MUST be rejected with an error.

3.1. X.509 Certificate SHA-256 Thumbprint Confirmation Method for JWT

When access tokens are represented as a JSON Web Tokens (JWT)[RFC7519], the certificate hash information SHOULD be represented using the "x5t#S256" confirmation method member defined herein.

To represent the hash of a certificate in a JWT, this specification defines the new JWT Confirmation Method RFC 7800 [RFC7800] member "x5t#S256" for the X.509 Certificate SHA-256 Thumbprint. The value of the "x5t#S256" member is a base64url-encoded SHA-256[SHS] hash (a.k.a. thumbprint or digest) of the DER encoding of the X.509 certificate[RFC5280] (note that certificate thumbprints are also sometimes also known as certificate fingerprints).

The following is an example of a JWT payload containing an "x5t#S256" certificate thumbprint confirmation method.

```
{
  "iss": "https://server.example.com",
  "aud": "https://resource.example.org",
  "sub": "ty.webb@example.com",
  "exp": "1493726400",
  "nbf": "1493722800",
  "cnf": {
    "x5t#s256": "bwck0esc3ACC3DB2Y5_lESsXE8o9ltc05O89jdN-dg2"
  }
}
```

Figure 1: Example claims of a Certificate Thumbprint Constrained JWT.

4. IANA Considerations

4.1. JWT Confirmation Methods Registration

This specification requests registration of the following value in the IANA "JWT Confirmation Methods" registry [IANA.JWT.Claims] for JWT "cnf" member values established by [RFC7800].

4.1.1. Registry Contents

- o Confirmation Method Value: "x5t#S256"
- o Confirmation Method Description: X.509 Certificate SHA-256 Thumbprint
- o Change Controller: IESG
- o Specification Document(s): Section 3.1 of [[this specification]]

4.2. Token Endpoint Authentication Method Registration

This specification requests registration of the following value in the IANA "OAuth Token Endpoint Authentication Methods" registry [IANA.OAuth.Parameters] established by [RFC7591].

4.2.1. Registry Contents

- o Token Endpoint Authentication Method Name: "tls_client_auth"
- o Change Controller: IESG
- o Specification Document(s): Section 2.2 of [[this specification]]

4.3. OAuth Dynamic Client Registration Metadata Registration

This specification requests registration of the following client metadata definitions in the IANA "OAuth Dynamic Client Registration Metadata" registry [IANA.OAuth.Parameters] established by [RFC7591]:

4.3.1. Registry Contents

- o Client Metadata Name: "tls_client_auth_subject_dn"
- o Client Metadata Description: String value specifying the expected subject distinguished name of the client certificate.
- o Change Controller: IESG
- o Specification Document(s): Section 2.3 of [[this specification]]
- o Client Metadata Name: "tls_client_auth_issuer_dn"
- o Client Metadata Description: String value specifying the expected distinguished name of the root issuer of the client certificate
- o Change Controller: IESG
- o Specification Document(s): Section 2.3 of [[this specification]]

5. Security Considerations

5.1. TLS Versions and Best Practices

TLS 1.2 [RFC5246] is cited in this document because, at the time of writing, it is latest version that is widely deployed. However, this document is applicable with other TLS versions supporting certificate-based client authentication. Implementation security considerations for TLS, including version recommendations, can be found in Recommendations for Secure Use of Transport Layer Security (TLS) and Datagram Transport Layer Security (DTLS) [BCP195].

5.2. Client Identity Binding

No specific method of binding a certificate to a client identifier at the token endpoint is prescribed by this document. However, some method **MUST** be employed so that, in addition to proving possession of the private key corresponding to the certificate, the client identity is also bound to the certificate. One such binding would be to configure for the client a value that the certificate must contain in the subject field or the subjectAltName extension and possibly a restricted set of trust anchors. An alternative method would be to configure a public key for the client directly that would have to match the subject public key info of the certificate.

6. References

6.1. Normative References

- [BCP195] Sheffer, Y., Holz, R., and P. Saint-Andre, "Recommendations for Secure Use of Transport Layer Security (TLS) and Datagram Transport Layer Security (DTLS)", BCP 195, RFC 7525, DOI 10.17487/RFC7525, May 2015, <<http://www.rfc-editor.org/info/bcp195>>.
- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <<http://www.rfc-editor.org/info/rfc2119>>.
- [RFC5246] Dierks, T. and E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.2", RFC 5246, DOI 10.17487/RFC5246, August 2008, <<http://www.rfc-editor.org/info/rfc5246>>.

- [RFC5280] Cooper, D., Santesson, S., Farrell, S., Boeyen, S., Housley, R., and W. Polk, "Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile", RFC 5280, DOI 10.17487/RFC5280, May 2008, <<http://www.rfc-editor.org/info/rfc5280>>.
- [RFC6749] Hardt, D., Ed., "The OAuth 2.0 Authorization Framework", RFC 6749, DOI 10.17487/RFC6749, October 2012, <<http://www.rfc-editor.org/info/rfc6749>>.
- [RFC6750] Jones, M. and D. Hardt, "The OAuth 2.0 Authorization Framework: Bearer Token Usage", RFC 6750, DOI 10.17487/RFC6750, October 2012, <<http://www.rfc-editor.org/info/rfc6750>>.
- [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, <<http://www.rfc-editor.org/info/rfc7800>>.
- [SHS] National Institute of Standards and Technology, "Secure Hash Standard (SHS)", FIPS PUB 180-4, March 2012, <<http://csrc.nist.gov/publications/fips/fips180-4/fips-180-4.pdf>>.

6.2. Informative References

- [I-D.ietf-oauth-discovery]
Jones, M., Sakimura, N., and J. Bradley, "OAuth 2.0 Authorization Server Metadata", draft-ietf-oauth-discovery-04 (work in progress), August 2016.
- [IANA.JWT.Claims]
IANA, "JSON Web Token Claims", <<http://www.iana.org/assignments/jwt>>.
- [IANA.OAuth.Parameters]
IANA, "OAuth Parameters", <<http://www.iana.org/assignments/oauth-parameters>>.
- [OpenID.Discovery]
Sakimura, N., Bradley, J., Jones, M., and E. Jay, "OpenID Connect Discovery 1.0", February 2014.
- [RFC7519] Jones, M., Bradley, J., and N. Sakimura, "JSON Web Token (JWT)", RFC 7519, DOI 10.17487/RFC7519, May 2015, <<http://www.rfc-editor.org/info/rfc7519>>.

- [RFC7591] Richer, J., Ed., Jones, M., Bradley, J., Machulak, M., and P. Hunt, "OAuth 2.0 Dynamic Client Registration Protocol", RFC 7591, DOI 10.17487/RFC7591, July 2015, <<http://www.rfc-editor.org/info/rfc7591>>.
- [RFC7662] Richer, J., Ed., "OAuth 2.0 Token Introspection", RFC 7662, DOI 10.17487/RFC7662, October 2015, <<http://www.rfc-editor.org/info/rfc7662>>.

Appendix A. Acknowledgements

Scott "not Tomlinson" Tomilson and Matt Peterson were involved in the original design and development work on a mutual TLS client authentication implementation that informed some of the content of this document.

Additionally, the authors would like to thank the following people for their input and contributions to the specification: Sergey Beryozkin, Vladimir Dzhuvinov, Samuel Erdtman, Phil Hunt, Sean Leonard, Jim Manico, Sascha Preibisch, Justin Richer, and Hannes Tschofenig.

Appendix B. Document(s) History

[[to be removed by the RFC Editor before publication as an RFC]]

draft-campbell-oauth-mtls-00

- o Add a Mutual TLS sender constrained protected resource access method and a x5t#s256 cnf method for JWT access tokens (concepts taken in part from draft-sakimura-oauth-jpop-04).
- o Fixed "token_endpoint_auth_methods_supported" to "token_endpoint_auth_method" for client metadata.
- o Add "tls_client_auth_subject_dn" and "tls_client_auth_issuer_dn" client metadata parameters and mention using "jwks_uri" or "jwks".
- o Say that the authentication method is determined by client policy regardless of whether the client was dynamically registered or statically configured.
- o Expand acknowledgements to those that participated in discussions around draft-campbell-oauth-tls-client-auth-00
- o Add Nat Sakimura and Torsten Lodderstedt to the author list.

draft-campbell-oauth-tls-client-auth-00

- o Initial draft.

Authors' Addresses

Brian Campbell
Ping Identity

Email: brian.d.campbell@gmail.com

John Bradley
Ping Identity

Email: ve7jtb@ve7jtb.com
URI: <http://www.thread-safe.com/>

Nat Sakimura
Nomura Research Institute

Email: n-sakimura@nri.co.jp
URI: <https://nat.sakimura.org/>

Torsten Lodderstedt
YES Europe AG

Email: torsten@lodderstedt.net

OAuth
Internet-Draft
Intended status: Standards Track
Expires: September 12, 2019

W. Denniss
Google
J. Bradley
Ping Identity
M. Jones
Microsoft
H. Tschofenig
ARM Limited
March 11, 2019

OAuth 2.0 Device Authorization Grant
draft-ietf-oauth-device-flow-15

Abstract

The OAuth 2.0 Device Authorization Grant is designed for internet-connected devices that either lack a browser to perform a user-agent based authorization, or are input-constrained to the extent that requiring the user to input text in order to authenticate during the authorization flow is impractical. It enables OAuth clients on such devices (like smart TVs, media consoles, digital picture frames, and printers) to obtain user authorization to access protected resources without using an on-device user-agent.

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 September 12, 2019.

Copyright Notice

Copyright (c) 2019 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<https://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1. Introduction	3
2. Terminology	5
3. Protocol	5
3.1. Device Authorization Request	5
3.2. Device Authorization Response	7
3.3. User Interaction	8
3.3.1. Non-textual Verification URI Optimization	9
3.4. Device Access Token Request	10
3.5. Device Access Token Response	11
4. Discovery Metadata	12
5. Security Considerations	12
5.1. User Code Brute Forcing	13
5.2. Device Code Brute Forcing	13
5.3. Device Trustworthiness	14
5.4. Remote Phishing	14
5.5. Session Spying	15
5.6. Non-confidential Clients	15
5.7. Non-Visual Code Transmission	15
6. Usability Considerations	15
6.1. User Code Recommendations	16
6.2. Non-Browser User Interaction	17
7. IANA Considerations	17
7.1. OAuth Parameters Registration	17
7.1.1. Registry Contents	17
7.2. OAuth URI Registration	17
7.2.1. Registry Contents	17
7.3. OAuth Extensions Error Registration	17
7.3.1. Registry Contents	18
7.4. OAuth 2.0 Authorization Server Metadata	18
7.4.1. Registry Contents	18
8. Normative References	18
Appendix A. Acknowledgements	19
Appendix B. Document History	20
Authors' Addresses	22

1. Introduction

This OAuth 2.0 [RFC6749] protocol extension, sometimes referred to as "device flow", enables OAuth clients to request user authorization from applications on devices that have limited input capabilities or lack a suitable browser. Such devices include those smart TVs, media console, picture frames and printers which lack an easy input method or suitable browser required for traditional OAuth interactions. The authorization flow defined by this specification instructs the user to review the authorization request on a secondary device, such as a smartphone which does have the requisite input and browser capabilities to complete the user interaction.

The Device Authorization Grant is not intended to replace browser-based OAuth in native apps on capable devices like smartphones. Those apps should follow the practices specified in OAuth 2.0 for Native Apps [RFC8252].

The operating requirements to be able to use this authorization grant type are:

- (1) The device is already connected to the Internet.
- (2) The device is able to make outbound HTTPS requests.
- (3) The device is able to display or otherwise communicate a URI and code sequence to the user.
- (4) The user has a secondary device (e.g., personal computer or smartphone) from which they can process the request.

As the device authorization grant does not require two-way communication between the OAuth client and the user-agent (unlike other OAuth 2 grant types such as the Authorization Code and Implicit grant types), it supports several use cases that cannot be served by those other approaches.

Instead of interacting with the end user's user agent, the client instructs the end user to use another computer or device and connect to the authorization server to approve the access request. Since the protocol supports clients that can't receive incoming requests, clients poll the authorization server repeatedly until the end user completes the approval process.

The device typically chooses the set of authorization servers to support (i.e., its own authorization server, or those by providers it has relationships with). It is not uncommon for the device application to support only a single authorization server, such as

with a TV application for a specific media provider that supports only that media provider's authorization server. The user may not have an established relationship yet with that authorization provider, though one can potentially be set up during the authorization flow.

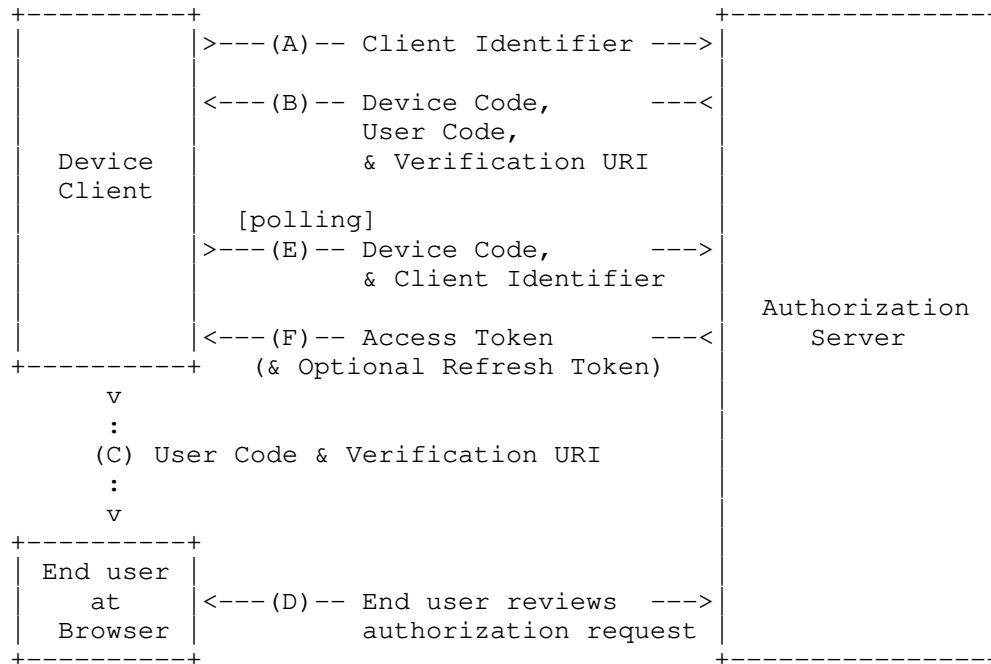


Figure 1: Device Authorization Flow

The device authorization flow illustrated in Figure 1 includes the following steps:

- (A) The client requests access from the authorization server and includes its client identifier in the request.
- (B) The authorization server issues a device code, an end-user code, and provides the end-user verification URI.
- (C) The client instructs the end user to use its user agent (on another device) and visit the provided end-user verification URI. The client provides the user with the end-user code to enter in order to review the authorization request.
- (D) The authorization server authenticates the end user (via the user agent) and prompts the user to grant the client's access

request. If the user agrees to the client's access request, the user enters the user code provided by the client. The authorization server validates the user code provided by the user.

(E) While the end user reviews the client's request (step D), the client repeatedly polls the authorization server to find out if the user completed the user authorization step. The client includes the verification code and its client identifier.

(F) The authorization server validates the verification code provided by the client and responds back with the access token if the user granted access, an error if they denied access, or indicates that the client should continue to poll.

2. Terminology

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

Device Authorization Endpoint:

The authorization server's endpoint capable of issuing device verification codes, user codes, and verification URLs.

Device Verification Code:

A short-lived token representing an authorization session.

End-User Verification Code:

A short-lived token which the device displays to the end user, is entered by the user on the authorization server, and is thus used to bind the device to the user.

3. Protocol

3.1. Device Authorization Request

This specification defines a new OAuth endpoint, the device authorization endpoint. This is separate from the OAuth authorization endpoint defined in [RFC6749] with which the user interacts with via a user-agent (i.e., a browser). By comparison, when using the device authorization endpoint, the OAuth client on the device interacts with the authorization server directly without presenting the request in a user-agent, and the end user authorizes the request on a separate device. This interaction is defined as follows.

The client initiates the authorization flow by requesting a set of verification codes from the authorization server by making an HTTP "POST" request to the device authorization endpoint.

The client constructs the request with the following parameters, sent as the body of the request, encoded with the "application/x-www-form-urlencoded" encoding algorithm defined by Section 4.10.22.6 of [HTML5]:

`client_id`

REQUIRED, if the client is not authenticating with the authorization server as described in Section 3.2.1. of [RFC6749]. The client identifier as described in Section 2.2 of [RFC6749].

`scope`

OPTIONAL. The scope of the access request as described by Section 3.3 of [RFC6749].

For example, the client makes the following HTTPS request:

```
POST /device_authorization HTTP/1.1
Host: server.example.com
Content-Type: application/x-www-form-urlencoded

client_id=459691054427
```

All requests from the device MUST use the Transport Layer Security (TLS) [RFC8446] protocol and implement the best practices of BCP 195 [RFC7525].

Parameters sent without a value MUST be treated as if they were omitted from the request. The authorization server MUST ignore unrecognized request parameters. Request and response parameters MUST NOT be included more than once.

The client authentication requirements of Section 3.2.1 of [RFC6749] apply to requests on this endpoint, which means that confidential clients (those that have established client credentials) authenticate in the same manner as when making requests to the token endpoint, and public clients provide the "client_id" parameter to identify themselves.

Due to the polling nature of this protocol (as specified in Section 3.4), care is needed to avoid overloading the capacity of the token endpoint. To avoid unneeded requests on the token endpoint, the client SHOULD only commence a device authorization request when prompted by the user, and not automatically, such as when the app starts or when the previous authorization session expires or fails.

3.2. Device Authorization Response

In response, the authorization server generates a unique device verification code and an end-user code that are valid for a limited time and includes them in the HTTP response body using the "application/json" format [RFC8259] with a 200 (OK) status code. The response contains the following parameters:

device_code
REQUIRED. The device verification code.

user_code
REQUIRED. The end-user verification code.

verification_uri
REQUIRED. The end-user verification URI on the authorization server. The URI should be short and easy to remember as end users will be asked to manually type it into their user-agent.

verification_uri_complete
OPTIONAL. A verification URI that includes the "user_code" (or other information with the same function as the "user_code"), designed for non-textual transmission.

expires_in
REQUIRED. The lifetime in seconds of the "device_code" and "user_code".

interval
OPTIONAL. The minimum amount of time in seconds that the client SHOULD wait between polling requests to the token endpoint. If no value is provided, clients MUST use 5 as the default.

For example:

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

{
  "device_code": "GmRhmhcxhwAzkoEqiMEg_DnyEysNkuNhszIySk9eS",
  "user_code": "WDJB-MJHT",
  "verification_uri": "https://example.com/device",
  "verification_uri_complete":
    "https://example.com/device?user_code=WDJB-MJHT",
  "expires_in": 1800,
  "interval": 5
}
```

In the event of an error (such as an invalidly configured client), the authorization server responds in the same way as the token endpoint specified in Section 5.2 of [RFC6749].

3.3. User Interaction

After receiving a successful Authorization Response, the client displays or otherwise communicates the "user_code" and the "verification_uri" to the end user and instructs them to visit the URI in a user agent on a secondary device (for example, in a browser on their mobile phone), and enter the user code.

```
Using a browser on another device, visit:
https://example.com/device

And enter the code:
WDJB-MJHT
```

Figure 2: Example User Instruction

The authorizing user navigates to the "verification_uri" and authenticates with the authorization server in a secure TLS-protected ([RFC8446]) session. The authorization server prompts the end user to identify the device authorization session by entering the "user_code" provided by the client. The authorization server should then inform the user about the action they are undertaking and ask them to approve or deny the request. Once the user interaction is complete, the server MAY inform the user to return to their device.

During the user interaction, the device continuously polls the token endpoint with the "device_code", as detailed in Section 3.4, until the user completes the interaction, the code expires, or another error occurs. The "device_code" is not intended for the end user directly, and thus should not be displayed during the interaction to avoid confusing the end user.

Authorization servers supporting this specification MUST implement a user interaction sequence that starts with the user navigating to "verification_uri" and continues with them supplying the "user_code" at some stage during the interaction. Other than that, the exact sequence and implementation of the user interaction is up to the authorization server, for example, the authorization server may enable new users to sign up for an account during the authorization flow, or add additional security verification steps.

It is NOT RECOMMENDED for authorization servers to include the user code in the verification URI ("verification_uri"), as this increases the length and complexity of the URI that the user must type. While the user must still type the same number of characters with the "user_code" separated, once they successfully navigate to the "verification_uri", any errors in entering the code can be highlighted by the authorization server to improve the user experience. The next section documents user interaction with "verification_uri_complete", which is designed to carry both pieces of information.

3.3.1. Non-textual Verification URI Optimization

When "verification_uri_complete" is included in the Authorization Response (Section 3.2), clients MAY present this URI in a non-textual manner using any method that results in the browser being opened with the URI, such as with QR (Quick Response) codes or NFC (Near Field Communication), to save the user typing the URI.

For usability reasons, it is RECOMMENDED for clients to still display the textual verification URI ("verification_uri") for users not able to use such a shortcut. Clients MUST still display the "user_code", as the authorization server will require the user to confirm it to disambiguate devices, or as a remote phishing mitigation (See Section 5.4).

If the user starts the user interaction by browsing to "verification_uri_complete", then the user interaction described in Section 3.3 is still followed, but with the optimization that the user does not need to type the "user_code". The server SHOULD display the "user_code" to the user and ask them to verify that it matches the "user_code" being displayed on the device, to confirm they are authorizing the correct device. As before, in addition to taking steps to confirm the identity of the device, the user should also be afforded the choice to approve or deny the authorization request.

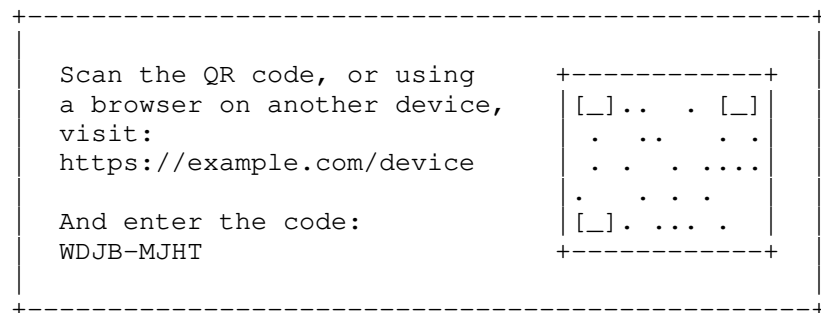


Figure 3: Example User Instruction with QR Code Representation of the Complete Verification URI

3.4. Device Access Token Request

After displaying instructions to the user, the client makes an Access Token Request to the token endpoint (as defined by Section 3.2 of [RFC6749]) with a "grant_type" of "urn:ietf:params:oauth:grant-type:device_code". This is an extension grant type (as defined by Section 4.5 of [RFC6749]) created by this specification, with the following parameters:

grant_type

REQUIRED. Value MUST be set to "urn:ietf:params:oauth:grant-type:device_code".

device_code

REQUIRED. The device verification code, "device_code" from the Device Authorization Response, defined in Section 3.2.

client_id

REQUIRED, if the client is not authenticating with the authorization server as described in Section 3.2.1. of [RFC6749]. The client identifier as described in Section 2.2 of [RFC6749].

For example, the client makes the following HTTPS request (line breaks are for display purposes only):

```
POST /token HTTP/1.1
Host: server.example.com
Content-Type: application/x-www-form-urlencoded

grant_type=urn%3Aietf%3Aparams%3Aoauth%3Agrant-type%3Adevice_code
&device_code=GmRhmcxhwAzkoEqiMEg_DnyEysNkuNhszIySk9eS
&client_id=459691054427
```

If the client was issued client credentials (or assigned other authentication requirements), the client MUST authenticate with the authorization server as described in Section 3.2.1 of [RFC6749]. Note that there are security implications of statically distributed client credentials, see Section 5.6.

The response to this request is defined in Section 3.5. Unlike other OAuth grant types, it is expected for the client to try the Access Token Request repeatedly in a polling fashion, based on the error code in the response.

3.5. Device Access Token Response

If the user has approved the grant, the token endpoint responds with a success response defined in Section 5.1 of [RFC6749]; otherwise it responds with an error, as defined in Section 5.2 of [RFC6749].

In addition to the error codes defined in Section 5.2 of [RFC6749], the following error codes are specified for use with the device authorization grant in token endpoint responses:

`authorization_pending`

The authorization request is still pending as the end user hasn't yet completed the user interaction steps (Section 3.3). The client SHOULD repeat the Access Token Request to the token endpoint (a process known as polling). Before each new request the client MUST wait at least the number of seconds specified by the "interval" parameter of the Device Authorization Response (see Section 3.2), or 5 seconds if none was provided, and respect any increase in the polling interval required by the "slow_down" error.

`slow_down`

A variant of "authorization_pending", the authorization request is still pending and polling should continue, but the interval MUST be increased by 5 seconds for this and all subsequent requests.

`access_denied`

The end user denied the authorization request.

expired_token

The "device_code" has expired and the device authorization session has concluded. The client MAY commence a new Device Authorization Request but SHOULD wait for user interaction before restarting to avoid unnecessary polling.

The "authorization_pending" and "slow_down" error codes define particularly unique behavior, as they indicate that the OAuth client should continue to poll the token endpoint by repeating the token request (implementing the precise behavior defined above). If the client receives an error response with any other error code, it MUST stop polling and SHOULD react accordingly, for example, by displaying an error to the user.

On encountering a connection timeout, clients MUST unilaterally reduce their polling frequency before retrying. The use of an exponential backoff algorithm to achieve this, such as by doubling the polling interval on each such connection timeout, is RECOMMENDED.

The assumption of this specification is that the separate device the user is authorizing the request on does not have a way to communicate back to device with the OAuth client. This protocol only requires a one-way channel in order to maximise the viability of the protocol in restricted environments, like an application running on a TV that is only capable of outbound requests. If a return channel were to exist for the chosen user interaction interface, then the device MAY wait until notified on that channel that the user has completed the action before initiating the token request (as an alternative to polling). Such behavior is, however, outside the scope of this specification.

4. Discovery Metadata

Support for this specification MAY be declared in the OAuth 2.0 Authorization Server Metadata [RFC8414] by including the value "urn:ietf:params:oauth:grant-type:device_code" in the "grant_types_supported" parameter, and by adding the following new parameter:

device_authorization_endpoint

OPTIONAL. URL of the authorization server's device authorization endpoint defined in Section 3.1.

5. Security Considerations

5.1. User Code Brute Forcing

Since the user code is typed by the user, shorter codes are more desirable for usability reasons. This means the entropy is typically less than would be used for the device code or other OAuth bearer token types where the code length does not impact usability. It is therefore recommended that the server rate-limit user code attempts.

The user code SHOULD have enough entropy that when combined with rate limiting and other mitigations makes a brute-force attack infeasible. For example, it's generally held that 128-bit symmetric keys for encryption are seen as good enough today because an attacker has to put in 2^{96} work to have a 2^{-32} chance of guessing correctly via brute force. The rate limiting and finite lifetime on the user code places an artificial limit on the amount of work an attacker can "do", so if, for instance, one uses a 8-character base-20 user code (with roughly 34.5 bits of entropy), the rate-limiting interval and validity period would need to only allow 5 attempts in order to get the same 2^{-32} probability of success by random guessing.

A successful brute forcing of the user code would enable the attacker to authenticate with their own credentials and make an authorization grant to the device. This is the opposite scenario to an OAuth bearer token being brute forced, whereby the attacker gains control of the victim's authorization grant. Such attacks may not always make economic sense, for example for a video app the device owner may then be able to purchase movies using the attacker's account, though a privacy risk would still remain and thus is important to protect against. Furthermore, some uses of the device flow give the granting account the ability to perform actions such as controlling the device, which needs to be protected.

The precise length of the user code and the entropy contained within is at the discretion of the authorization server, which needs to consider the sensitivity of their specific protected resources, the practicality of the code length from a usability standpoint, and any mitigations that are in place such as rate-limiting, when determining the user code format.

5.2. Device Code Brute Forcing

An attacker who guesses the device code would be able to potentially obtain the authorization code once the user completes the flow. As the device code is not displayed to the user and thus there are no usability considerations on the length, a very high entropy code SHOULD be used.

5.3. Device Trustworthiness

Unlike other native application OAuth 2.0 flows, the device requesting the authorization is not the same as the device that the user grants access from. Thus, signals from the approving user's session and device are not relevant to the trustworthiness of the client device.

Note that if an authorization server used with this flow is malicious, then it could man-in-the-middle the backchannel flow to another authorization server. In this scenario, the man-in-the-middle is not completely hidden from sight, as the end user would end up on the authorization page of the wrong service, giving them an opportunity to notice that the URL in the browser's address bar is wrong. For this to be possible, the device manufacturer must either directly be the attacker, shipping a device intended to perform the man-in-the-middle attack, or be using an authorization server that is controlled by an attacker, possibly because the attacker compromised the authorization server used by the device. In part, the person purchasing the device is counting on it and its business partners to be trustworthy.

5.4. Remote Phishing

It is possible for the device flow to be initiated on a device in an attacker's possession. For example, an attacker might send an email instructing the target user to visit the verification URL and enter the user code. To mitigate such an attack, it is RECOMMENDED to inform the user that they are authorizing a device during the user interaction step (see Section 3.3), and to confirm that the device is in their possession. The authorization server SHOULD display information about the device so that the person can notice if a software client was attempting to impersonating a hardware device.

For authorization servers that support the option specified in Section 3.3.1 for the client to append the user code to the authorization URI, it is particularly important to confirm that the device is in the user's possession, as the user no longer has to type the code manually. One possibility is to display the code during the authorization flow and asking the user to verify that the same code is being displayed on the device they are setting up.

The user code needs to have a long enough lifetime to be useable (allowing the user to retrieve their secondary device, navigate to the verification URI, login, etc.), but should be sufficiently short to limit the usability of a code obtained for phishing. This doesn't prevent a phisher presenting a fresh token, particularly in the case

they are interacting with the user in real time, but it does limit the viability of codes sent over email or SMS.

5.5. Session Spying

While the device is pending authorization, it may be possible for a malicious user to physically spy on the device user interface (by viewing the screen on which it's displayed, for example) and hijack the session by completing the authorization faster than the user that initiated it. Devices SHOULD take into account the operating environment when considering how to communicate the code to the user to reduce the chances it will be observed by a malicious user.

5.6. Non-confidential Clients

Device clients are generally incapable of maintaining the confidentiality of their credentials, as users in possession of the device can reverse engineer it and extract the credentials. Therefore, unless additional measures are taken, they should be treated as public clients (as defined by Section 2.1 of OAuth 2.0) susceptible to impersonation. The security considerations of Section 5.3.1 of [RFC6819] and Sections 8.5 and 8.6 of [RFC8252] apply to such clients.

The user may also be able to obtain the device_code and/or other OAuth bearer tokens issued to their client, which would allow them to use their own authorization grant directly by impersonating the client. Given that the user in possession of the client credentials can already impersonate the client and create a new authorization grant (with a new device_code), this doesn't represent a separate impersonation vector.

5.7. Non-Visual Code Transmission

There is no requirement that the user code be displayed by the device visually. Other methods of one-way communication can potentially be used, such as text-to-speech audio, or Bluetooth Low Energy. To mitigate an attack in which a malicious user can bootstrap their credentials on a device not in their control, it is RECOMMENDED that any chosen communication channel only be accessible by people in close proximity. E.g., users who can see, or hear the device.

6. Usability Considerations

This section is a non-normative discussion of usability considerations.

6.1. User Code Recommendations

For many users, their nearest Internet-connected device will be their mobile phone, and typically these devices offer input methods that are more time consuming than a computer keyboard to change the case or input numbers. To improve usability (improving entry speed, and reducing retries), these limitations should be taken into account when selecting the user-code character set.

One way to improve input speed is to restrict the character set to case-insensitive A-Z characters, with no digits. These characters can typically be entered on a mobile keyboard without using modifier keys. Further removing vowels to avoid randomly creating words results in the base-20 character set: "BCDFGHJKLMNPQRSTVWXZ". Dashes or other punctuation may be included for readability.

An example user code following this guideline containing 8 significant characters and dashes added for end-user readability, with a resulting entropy of 20^8 : "WDJB-MJHT".

Pure numeric codes are also a good choice for usability, especially for clients targeting locales where A-Z character keyboards are not used, though their length needs to be longer to maintain a high entropy.

An example numeric user code containing 9 significant digits and dashes added for end-user readability, with an entropy of 10^9 : "019-450-730".

When processing the inputted user code, the server should strip dashes and other punctuation it added for readability (making the inclusion of that punctuation by the user optional). For codes using only characters in the A-Z range as with the base-20 charset defined above, the user's input should be upper-cased before comparison to account for the fact that the user may input the equivalent lower-case characters. Further stripping of all characters outside the `user_code` charset is recommended to reduce instances where an errantly typed character (like a space character) invalidates otherwise valid input.

It is RECOMMENDED to avoid character sets that contain two or more characters that can easily be confused with each other like "0" and "O", or "1", "l" and "I". Furthermore, the extent practical, where a character set contains one character that may be confused with characters outside the character set the character outside the set MAY be substituted with the one in the character set that it is commonly confused with (for example, "O" for "0" when using a numerical 0-9 character set).

6.2. Non-Browser User Interaction

Devices and authorization servers MAY negotiate an alternative code transmission and user interaction method in addition to the one described in Section 3.3. Such an alternative user interaction flow could obviate the need for a browser and manual input of the code, for example, by using Bluetooth to transmit the code to the authorization server's companion app. Such interaction methods can utilize this protocol, as ultimately, the user just needs to identify the authorization session to the authorization server; however, user interaction other than via the verification URI is outside the scope of this specification.

7. IANA Considerations

7.1. OAuth Parameters Registration

This specification registers the following values in the IANA "OAuth Parameters" registry [IANA.OAuth.Parameters] established by [RFC6749].

7.1.1. Registry Contents

- o Parameter name: device_code
- o Parameter usage location: token request
- o Change controller: IESG
- o Specification Document: Section 3.1 of [[this specification]]

7.2. OAuth URI Registration

This specification registers the following values in the IANA "OAuth URI" registry [IANA.OAuth.Parameters] established by [RFC6755].

7.2.1. Registry Contents

- o URN: urn:ietf:params:oauth:grant-type:device_code
- o Common Name: Device flow grant type for OAuth 2.0
- o Change controller: IESG
- o Specification Document: Section 3.1 of [[this specification]]

7.3. OAuth Extensions Error Registration

This specification registers the following values in the IANA "OAuth Extensions Error Registry" registry [IANA.OAuth.Parameters] established by [RFC6749].

7.3.1. Registry Contents

- o Error name: `authorization_pending`
- o Error usage location: Token endpoint response
- o Related protocol extension: [[this specification]]
- o Change controller: IETF
- o Specification Document: Section 3.5 of [[this specification]]

- o Error name: `access_denied`
- o Error usage location: Token endpoint response
- o Related protocol extension: [[this specification]]
- o Change controller: IETF
- o Specification Document: Section 3.5 of [[this specification]]

- o Error name: `slow_down`
- o Error usage location: Token endpoint response
- o Related protocol extension: [[this specification]]
- o Change controller: IETF
- o Specification Document: Section 3.5 of [[this specification]]

- o Error name: `expired_token`
- o Error usage location: Token endpoint response
- o Related protocol extension: [[this specification]]
- o Change controller: IETF
- o Specification Document: Section 3.5 of [[this specification]]

7.4. OAuth 2.0 Authorization Server Metadata

This specification registers the following values in the IANA "OAuth 2.0 Authorization Server Metadata" registry [IANA.OAuth.Parameters] established by [RFC8414].

7.4.1. Registry Contents

- o Metadata name: `device_authorization_endpoint`
- o Metadata Description: The Device Authorization Endpoint.
- o Change controller: IESG
- o Specification Document: Section 4 of [[this specification]]

8. Normative References

[HTML5] IANA, "HTML5",
<<https://www.w3.org/TR/2014/REC-html5-20141028/>>.

[IANA.OAuth.Parameters]
IANA, "OAuth Parameters",
<<http://www.iana.org/assignments/oauth-parameters>>.

- [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>>.
- [RFC6819] Lodderstedt, T., Ed., McGloin, M., and P. Hunt, "OAuth 2.0 Threat Model and Security Considerations", RFC 6819, DOI 10.17487/RFC6819, January 2013, <<https://www.rfc-editor.org/info/rfc6819>>.
- [RFC7525] Sheffer, Y., Holz, R., and P. Saint-Andre, "Recommendations for Secure Use of Transport Layer Security (TLS) and Datagram Transport Layer Security (DTLS)", BCP 195, RFC 7525, DOI 10.17487/RFC7525, May 2015, <<https://www.rfc-editor.org/info/rfc7525>>.
- [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>>.
- [RFC8252] Denniss, W. and J. Bradley, "OAuth 2.0 for Native Apps", BCP 212, RFC 8252, DOI 10.17487/RFC8252, October 2017, <<https://www.rfc-editor.org/info/rfc8252>>.
- [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/info/rfc8259>>.
- [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/info/rfc8414>>.
- [RFC8446] Rescorla, E., "The Transport Layer Security (TLS) Protocol Version 1.3", RFC 8446, DOI 10.17487/RFC8446, August 2018, <<https://www.rfc-editor.org/info/rfc8446>>.

Appendix A. Acknowledgements

The starting point for this document was the Internet-Draft draft-recordon-oauth-v2-device, authored by David Recordon and Brent Goldman, which itself was based on content in draft versions of the OAuth 2.0 protocol specification removed prior to publication due to

a then lack of sufficient deployment expertise. Thank you to the OAuth working group members who contributed to those earlier drafts.

This document was produced in the OAuth working group under the chairpersonship of Rifaat Shekh-Yusef and Hannes Tschofenig with Benjamin Kaduk, Kathleen Moriarty, and Eric Rescorla serving as Security Area Directors.

The following individuals contributed ideas, feedback, and wording that shaped and formed the final specification:

Alissa Cooper, Ben Campbell, Brian Campbell, Roshni Chandrashekhar, Eric Fazendin, Benjamin Kaduk, Jamshid Khosravian, Torsten Lodderstedt, James Manger, Dan McNulty, Breno de Medeiros, Simon Moffatt, Stein Myrseth, Emond Papegaaïj, Justin Richer, Adam Roach, Nat Sakimura, Andrew Sciberras, Marius Scurtescu, Filip Skokan, Ken Wang, and Steven E. Wright.

Appendix B. Document History

[[to be removed by the RFC Editor before publication as an RFC]]

-15

- o Renamed and dropped most usage of the term "flow"
- o Documented error responses on the authorization endpoint
- o Documented client authentication for the authorization endpoint

-14

- o Added more normative text on polling behavior.
- o Added discussion on risk of user retrieving their own device_code.
- o Editorial improvements.

-13

- o Added a longer discussion about entropy, proposed by Benjamin Kaduk.
- o Added device_code to OAuth IANA registry.
- o Expanded explanation of "case insensitive".
- o Added security section on Device Code Brute Forcing.
- o application/x-www-form-urlencoded normativly referenced.
- o Editorial improvements.

-12

- o Set a default polling interval to 5s explicitly.

- o Defined the `slow_down` behavior that it should increase the current interval by 5s.
- o `expires_in` now REQUIRED
- o Other changes in response to review feedback.

-11

- o Updated reference to OAuth 2.0 Authorization Server Metadata.

-10

- o Added a missing definition of `access_denied` for use on the token endpoint.
- o Corrected text documenting which error code should be returned for expired tokens (it's `"expired_token"`, not `"invalid_grant"`).
- o Corrected section reference to RFC 8252 (the section numbers had changed after the initial reference was made).
- o Fixed line length of one diagram (was causing `xml2rfc` warnings).
- o Added line breaks so the URN `grant_type` is presented on an unbroken line.
- o Typos fixed and other stylistic improvements.

-09

- o Addressed review comments by Security Area Director Eric Rescorla about the potential of a confused deputy attack.

-08

- o Expanded the User Code Brute Forcing section to include more detail on this attack.

-07

- o Replaced the `"user_code"` URI parameter optimization with `verification_uri_complete` following the IETF99 working group discussion.
- o Added security consideration about spying.
- o Required that `device_code` not be shown.
- o Added text regarding a minimum polling interval.

-06

- o Clarified usage of the `"user_code"` URI parameter optimization following the IETF98 working group discussion.

-05

- o response_type parameter removed from authorization request.
- o Added option for clients to include the user_code on the verification URI.
- o Clarified token expiry, and other nits.

-04

- o Security & Usability sections. OAuth Discovery Metadata.

-03

- o device_code is now a URN. Added IANA Considerations

-02

- o Added token request & response specification.

-01

- o Applied spelling and grammar corrections and added the Document History appendix.

-00

- o Initial working group draft based on draft-recordon-oauth-v2-device.

Authors' Addresses

William Denniss
Google
1600 Amphitheatre Pkwy
Mountain View, CA 94043
USA

Email: wdenniss@google.com
URI: <http://wdenniss.com/device-flow>

John Bradley
Ping Identity

Email: ve7jtb@ve7jtb.com
URI: <http://www.thread-safe.com/>

Michael B. Jones
Microsoft

Email: mbj@microsoft.com
URI: <http://self-issued.info/>

Hannes Tschofenig
ARM Limited
Austria

Email: Hannes.Tschofenig@gmx.net
URI: <http://www.tschofenig.priv.at>

OAuth Working Group
Internet-Draft
Intended status: Standards Track
Expires: September 5, 2018

M. Jones
Microsoft
N. Sakimura
NRI
J. Bradley
Ping Identity
March 4, 2018

OAuth 2.0 Authorization Server Metadata
draft-ietf-oauth-discovery-10

Abstract

This specification defines a metadata format that an OAuth 2.0 client can use to obtain the information needed to interact with an OAuth 2.0 authorization server, including its endpoint locations and authorization server capabilities.

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 September 5, 2018.

Copyright Notice

Copyright (c) 2018 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<https://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of

the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1. Introduction	2
1.1. Requirements Notation and Conventions	3
1.2. Terminology	3
2. Authorization Server Metadata	4
2.1. Signed Authorization Server Metadata	7
3. Obtaining Authorization Server Metadata	8
3.1. Authorization Server Metadata Request	9
3.2. Authorization Server Metadata Response	9
3.3. Authorization Server Metadata Validation	10
4. String Operations	11
5. Compatibility Notes	11
6. Security Considerations	12
6.1. TLS Requirements	12
6.2. Impersonation Attacks	12
6.3. Publishing Metadata in a Standard Format	13
6.4. Protected Resources	13
7. IANA Considerations	13
7.1. OAuth Authorization Server Metadata Registry	14
7.1.1. Registration Template	15
7.1.2. Initial Registry Contents	15
7.2. Updated Registration Instructions	18
7.3. Well-Known URI Registry	19
7.3.1. Registry Contents	19
8. References	19
8.1. Normative References	19
8.2. Informative References	21
Appendix A. Acknowledgements	22
Appendix B. Document History	22
Authors' Addresses	25

1. Introduction

This specification generalizes the metadata format defined by "OpenID Connect Discovery 1.0" [OpenID.Discovery] in a way that is compatible with OpenID Connect Discovery, while being applicable to a wider set of OAuth 2.0 use cases. This is intentionally parallel to the way that the "OAuth 2.0 Dynamic Client Registration Protocol" [RFC7591] specification generalized the dynamic client registration mechanisms defined by "OpenID Connect Dynamic Client Registration 1.0" [OpenID.Registration] in a way that was compatible with it.

The metadata for an authorization server is retrieved from a well-known location as a JSON [RFC7159] document, which declares its

endpoint locations and authorization server capabilities. This process is described in Section 3.

This metadata can either be communicated in a self-asserted fashion by the server origin via HTTPS or as a set of signed metadata values represented as claims in a JSON Web Token (JWT) [JWT]. In the JWT case, the issuer is vouching for the validity of the data about the authorization server. This is analogous to the role that the Software Statement plays in OAuth Dynamic Client Registration [RFC7591].

The means by which the client chooses an authorization server is out of scope. In some cases, its issuer identifier may be manually configured into the client. In other cases, it may be dynamically discovered, for instance, through the use of WebFinger [RFC7033], as described in Section 2 of "OpenID Connect Discovery 1.0" [OpenID.Discovery].

1.1. Requirements Notation and Conventions

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.

All uses of JSON Web Signature (JWS) [JWS] and JSON Web Encryption (JWE) [JWE] data structures in this specification utilize the JWS Compact Serialization or the JWE Compact Serialization; the JWS JSON Serialization and the JWE JSON Serialization are not used.

1.2. Terminology

This specification uses the terms "Access Token", "Authorization Code", "Authorization Endpoint", "Authorization Grant", "Authorization Server", "Client", "Client Authentication", "Client Identifier", "Client Secret", "Grant Type", "Protected Resource", "Redirection URI", "Refresh Token", "Resource Owner", "Resource Server", "Response Type", and "Token Endpoint" defined by OAuth 2.0 [RFC6749], the terms "Claim Name", "Claim Value", and "JSON Web Token (JWT)" defined by JSON Web Token (JWT) [JWT], and the term "Response Mode" defined by OAuth 2.0 Multiple Response Type Encoding Practices [OAuth.Responses].

2. Authorization Server Metadata

Authorization servers can have metadata describing their configuration. The following authorization server metadata values are used by this specification and are registered in the IANA "OAuth Authorization Server Metadata" registry established in Section 7.1:

issuer

REQUIRED. The authorization server's issuer identifier, which is a URL that uses the "https" scheme and has no query or fragment components. Authorization server metadata is published at a ".well-known" RFC 5785 [RFC5785] location derived from this issuer identifier, as described in Section 3. The issuer identifier is used to prevent authorization server mix-up attacks, as described in "OAuth 2.0 Mix-Up Mitigation" [I-D.ietf-oauth-mix-up-mitigation].

authorization_endpoint

URL of the authorization server's authorization endpoint [RFC6749]. This is REQUIRED unless no grant types are supported that use the authorization endpoint.

token_endpoint

URL of the authorization server's token endpoint [RFC6749]. This is REQUIRED unless only the implicit grant type is supported.

jwks_uri

OPTIONAL. URL of the authorization server's JWK Set [JWK] document. The referenced document contains the signing key(s) the client uses to validate signatures from the authorization server. This URL MUST use the "https" scheme. The JWK Set MAY also contain the server's encryption key(s), which are used by clients to encrypt requests to the server. When both signing and encryption keys are made available, a "use" (public key use) parameter value is REQUIRED for all keys in the referenced JWK Set to indicate each key's intended usage.

registration_endpoint

OPTIONAL. URL of the authorization server's OAuth 2.0 Dynamic Client Registration endpoint [RFC7591].

scopes_supported

RECOMMENDED. JSON array containing a list of the OAuth 2.0 [RFC6749] "scope" values that this authorization server supports. Servers MAY choose not to advertise some supported scope values even when this parameter is used.

response_types_supported

REQUIRED. JSON array containing a list of the OAuth 2.0 "response_type" values that this authorization server supports. The array values used are the same as those used with the "response_types" parameter defined by "OAuth 2.0 Dynamic Client Registration Protocol" [RFC7591].

`response_modes_supported`

OPTIONAL. JSON array containing a list of the OAuth 2.0 "response_mode" values that this authorization server supports, as specified in OAuth 2.0 Multiple Response Type Encoding Practices [OAuth.Responses]. If omitted, the default is "["query", "fragment"]". The response mode value "form_post" is also defined in OAuth 2.0 Form Post Response Mode [OAuth.Post].

`grant_types_supported`

OPTIONAL. JSON array containing a list of the OAuth 2.0 grant type values that this authorization server supports. The array values used are the same as those used with the "grant_types" parameter defined by "OAuth 2.0 Dynamic Client Registration Protocol" [RFC7591]. If omitted, the default value is "["authorization_code", "implicit"]".

`token_endpoint_auth_methods_supported`

OPTIONAL. JSON array containing a list of client authentication methods supported by this token endpoint. Client authentication method values are used in the "token_endpoint_auth_method" parameter defined in Section 2 of [RFC7591]. If omitted, the default is "client_secret_basic" -- the HTTP Basic Authentication Scheme specified in Section 2.3.1 of OAuth 2.0 [RFC6749].

`token_endpoint_auth_signing_alg_values_supported`

OPTIONAL. JSON array containing a list of the JWS signing algorithms ("alg" values) supported by the token endpoint for the signature on the JWT [JWT] used to authenticate the client at the token endpoint for the "private_key_jwt" and "client_secret_jwt" authentication methods. This metadata entry MUST be present if either of these authentication methods are specified in the "token_endpoint_auth_methods_supported" entry. No default algorithms are implied if this entry is omitted. Servers SHOULD support "RS256". The value "none" MUST NOT be used.

`service_documentation`

OPTIONAL. URL of a page containing human-readable information that developers might want or need to know when using the authorization server. In particular, if the authorization server does not support Dynamic Client Registration, then information on how to register clients needs to be provided in this documentation.

ui_locales_supported

OPTIONAL. Languages and scripts supported for the user interface, represented as a JSON array of BCP47 [RFC5646] language tag values. If omitted, the set of supported languages and scripts is unspecified.

op_policy_uri

OPTIONAL. URL that the authorization server provides to the person registering the client to read about the authorization server's requirements on how the client can use the data provided by the authorization server. The registration process SHOULD display this URL to the person registering the client if it is given. As described in Section 5, despite the identifier "op_policy_uri", appearing to be OpenID-specific, its usage in this specification is actually referring to a general OAuth 2.0 feature that is not specific to OpenID Connect.

op_tos_uri

OPTIONAL. URL that the authorization server provides to the person registering the client to read about the authorization server's terms of service. The registration process SHOULD display this URL to the person registering the client if it is given. As described in Section 5, despite the identifier "op_tos_uri", appearing to be OpenID-specific, its usage in this specification is actually referring to a general OAuth 2.0 feature that is not specific to OpenID Connect.

revocation_endpoint

OPTIONAL. URL of the authorization server's OAuth 2.0 revocation endpoint [RFC7009].

revocation_endpoint_auth_methods_supported

OPTIONAL. JSON array containing a list of client authentication methods supported by this revocation endpoint. The valid client authentication method values are those registered in the IANA "OAuth Token Endpoint Authentication Methods" registry [IANA.OAuth.Parameters]. If omitted, the default is "client_secret_basic" -- the HTTP Basic Authentication Scheme specified in Section 2.3.1 of OAuth 2.0 [RFC6749].

revocation_endpoint_auth_signing_alg_values_supported

OPTIONAL. JSON array containing a list of the JWS signing algorithms ("alg" values) supported by the revocation endpoint for the signature on the JWT [JWT] used to authenticate the client at the revocation endpoint for the "private_key_jwt" and "client_secret_jwt" authentication methods. This metadata entry MUST be present if either of these authentication methods are specified in the "revocation_endpoint_auth_methods_supported"

entry. No default algorithms are implied if this entry is omitted. The value "none" MUST NOT be used.

`introspection_endpoint`

OPTIONAL. URL of the authorization server's OAuth 2.0 introspection endpoint [RFC7662].

`introspection_endpoint_auth_methods_supported`

OPTIONAL. JSON array containing a list of client authentication methods supported by this introspection endpoint. The valid client authentication method values are those registered in the IANA "OAuth Token Endpoint Authentication Methods" registry [IANA.OAuth.Parameters] or those registered in the IANA "OAuth Access Token Types" registry [IANA.OAuth.Parameters]. (These values are and will remain distinct, due to Section 7.2.) If omitted, the set of supported authentication methods MUST be determined by other means.

`introspection_endpoint_auth_signing_alg_values_supported`

OPTIONAL. JSON array containing a list of the JWS signing algorithms ("alg" values) supported by the introspection endpoint for the signature on the JWT [JWT] used to authenticate the client at the introspection endpoint for the "private_key_jwt" and "client_secret_jwt" authentication methods. This metadata entry MUST be present if either of these authentication methods are specified in the "introspection_endpoint_auth_methods_supported" entry. No default algorithms are implied if this entry is omitted. The value "none" MUST NOT be used.

`code_challenge_methods_supported`

OPTIONAL. JSON array containing a list of PKCE [RFC7636] code challenge methods supported by this authorization server. Code challenge method values are used in the "code_challenge_method" parameter defined in Section 4.3 of [RFC7636]. The valid code challenge method values are those registered in the IANA "PKCE Code Challenge Methods" registry [IANA.OAuth.Parameters]. If omitted, the authorization server does not support PKCE.

Additional authorization server metadata parameters MAY also be used. Some are defined by other specifications, such as OpenID Connect Discovery 1.0 [OpenID.Discovery].

2.1. Signed Authorization Server Metadata

In addition to JSON elements, metadata values MAY also be provided as a "signed_metadata" value, which is a JSON Web Token (JWT) [JWT] that asserts metadata values about the authorization server as a bundle. A set of claims that can be used in signed metadata are defined in

Section 2. The signed metadata MUST be digitally signed or MACed using JSON Web Signature (JWS) [JWS] and MUST contain an "iss" (issuer) claim denoting the party attesting to the claims in the signed metadata. Consumers of the metadata MAY ignore the signed metadata if they do not support this feature. If the consumer of the metadata supports signed metadata, metadata values conveyed in the signed metadata MUST take precedence over the corresponding values conveyed using plain JSON elements.

Signed metadata is included in the authorization server metadata JSON object using this OPTIONAL member:

`signed_metadata`

A JWT containing metadata values about the authorization server as claims. This is a string value consisting of the entire signed JWT. A "signed_metadata" metadata value SHOULD NOT appear as a claim in the JWT.

3. Obtaining Authorization Server Metadata

Authorization servers supporting metadata MUST make a JSON document containing metadata as specified in Section 2 available at a path formed by inserting a well-known URI string into the authorization server's issuer identifier between the host component and the path component, if any. By default, the well-known URI string used is `"/.well-known/oauth-authorization-server"`. This path MUST use the "https" scheme. The syntax and semantics of ".well-known" are defined in RFC 5785 [RFC5785]. The well-known URI suffix used MUST be registered in the IANA "Well-Known URIs" registry [IANA.well-known].

Different applications utilizing OAuth authorization servers in application-specific ways may define and register different well-known URI suffixes used to publish authorization server metadata as used by those applications. For instance, if the Example application uses an OAuth authorization server in an Example-specific way, and there are Example-specific metadata values that it needs to publish, then it might register and use the "example-configuration" URI suffix and publish the metadata document at the path formed by inserting `"/.well-known/example-configuration"` between the host and path components of the authorization server's issuer identifier. Alternatively, many such applications will use the default well-known URI string `"/.well-known/oauth-authorization-server"`, which is the right choice for general-purpose OAuth authorization servers, and not register an application-specific one.

An OAuth 2.0 application using this specification MUST specify what well-known URI suffix it will use for this purpose. The same

authorization server MAY choose to publish its metadata at multiple well-known locations derived from its issuer identifier, for example, publishing metadata at both `"/.well-known/example-configuration"` and `"/.well-known/oauth-authorization-server"`.

Some OAuth applications will choose to use the well-known URI suffix `"openid-configuration"`. As described in Section 5, despite the identifier `"/.well-known/openid-configuration"`, appearing to be OpenID-specific, its usage in this specification is actually referring to a general OAuth 2.0 feature that is not specific to OpenID Connect.

3.1. Authorization Server Metadata Request

An authorization server metadata document MUST be queried using an HTTP `"GET"` request at the previously specified path.

The client would make the following request when the issuer identifier is `"https://example.com"` and the well-known URI suffix is `"oauth-authorization-server"` to obtain the metadata, since the issuer identifier contains no path component:

```
GET /.well-known/oauth-authorization-server HTTP/1.1
Host: example.com
```

If the issuer identifier value contains a path component, any terminating `"/"` MUST be removed before inserting `"/.well-known/"` and the well-known URI suffix between the host component and the path component. The client would make the following request when the issuer identifier is `"https://example.com/issuer1"` and the well-known URI suffix is `"oauth-authorization-server"` to obtain the metadata, since the issuer identifier contains a path component:

```
GET /.well-known/oauth-authorization-server/issuer1 HTTP/1.1
Host: example.com
```

Using path components enables supporting multiple issuers per host. This is required in some multi-tenant hosting configurations. This use of `".well-known"` is for supporting multiple issuers per host; unlike its use in RFC 5785 [RFC5785], it does not provide general information about the host.

3.2. Authorization Server Metadata Response

The response is a set of claims about the authorization server's configuration, including all necessary endpoints and public key location information. A successful response MUST use the 200 OK HTTP status code and return a JSON object using the `"application/json"`

content type that contains a set of claims as its members that are a subset of the metadata values defined in Section 2. Other claims MAY also be returned.

Claims that return multiple values are represented as JSON arrays. Claims with zero elements MUST be omitted from the response.

An error response uses the applicable HTTP status code value.

The following is a non-normative example response:

```
HTTP/1.1 200 OK
Content-Type: application/json

{
  "issuer":
    "https://server.example.com",
  "authorization_endpoint":
    "https://server.example.com/authorize",
  "token_endpoint":
    "https://server.example.com/token",
  "token_endpoint_auth_methods_supported":
    ["client_secret_basic", "private_key_jwt"],
  "token_endpoint_auth_signing_alg_values_supported":
    ["RS256", "ES256"],
  "userinfo_endpoint":
    "https://server.example.com/userinfo",
  "jwks_uri":
    "https://server.example.com/jwks.json",
  "registration_endpoint":
    "https://server.example.com/register",
  "scopes_supported":
    ["openid", "profile", "email", "address",
     "phone", "offline_access"],
  "response_types_supported":
    ["code", "code token"],
  "service_documentation":
    "http://server.example.com/service_documentation.html",
  "ui_locales_supported":
    ["en-US", "en-GB", "en-CA", "fr-FR", "fr-CA"]
}
```

3.3. Authorization Server Metadata Validation

The "issuer" value returned MUST be identical to the authorization server's issuer identifier value into which the well-known URI string was inserted to create the URL used to retrieve the metadata. If

these values are not identical, the data contained in the response MUST NOT be used.

4. String Operations

Processing some OAuth 2.0 messages requires comparing values in the messages to known values. For example, the member names in the metadata response might be compared to specific member names such as "issuer". Comparing Unicode [UNICODE] strings, however, has significant security implications.

Therefore, comparisons between JSON strings and other Unicode strings MUST be performed as specified below:

1. Remove any JSON applied escaping to produce an array of Unicode code points.
2. Unicode Normalization [USA15] MUST NOT be applied at any point to either the JSON string or to the string it is to be compared against.
3. Comparisons between the two strings MUST be performed as a Unicode code point to code point equality comparison.

Note that this is the same equality comparison procedure described in Section 8.3 of [RFC7159].

5. Compatibility Notes

The identifiers `"/.well-known/openid-configuration"`, `"op_policy_uri"`, and `"op_tos_uri"` contain strings referring to the OpenID Connect [OpenID.Core] family of specifications that were originally defined by "OpenID Connect Discovery 1.0" [OpenID.Discovery]. Despite the reuse of these identifiers that appear to be OpenID-specific, their usage in this specification is actually referring to general OAuth 2.0 features that are not specific to OpenID Connect.

The algorithm for transforming the issuer identifier to an authorization server metadata location defined in Section 3 is equivalent to the corresponding transformation defined in Section 4 of "OpenID Connect Discovery 1.0" [OpenID.Discovery], provided that the issuer identifier contains no path component. However, they are different when there is a path component, because OpenID Connect Discovery 1.0 specifies that the well-known URI string is appended to the issuer identifier (e.g., `"https://example.com/issuer1/.well-known/openid-configuration"`), whereas this specification specifies that the well-known URI string is inserted before the path component

of the issuer identifier (e.g., "https://example.com/.well-known/openid-configuration/issuer1").

Going forward, OAuth authorization server metadata locations should use the transformation defined in this specification. However, when deployed in legacy environments in which the OpenID Connect Discovery 1.0 transformation is already used, it may be necessary during a transition period to publish metadata for issuer identifiers containing a path component at both locations. During this transition period, applications should first apply the transformation defined in this specification and attempt to retrieve the authorization server metadata from the resulting location; only if the retrieval from that location fails should they fall back to attempting to retrieve it from the alternate location obtained using the transformation defined by OpenID Connect Discovery 1.0. This backwards-compatibility behavior should only be necessary when the well-known URI suffix employed by the application is "openid-configuration".

6. Security Considerations

6.1. TLS Requirements

Implementations **MUST** support TLS. Which version(s) ought to be implemented will vary over time and depend on the widespread deployment and known security vulnerabilities at the time of implementation. The authorization server **MUST** support TLS version 1.2 [RFC5246] and **MAY** support additional transport-layer security mechanisms meeting its security requirements. When using TLS, the client **MUST** perform a TLS/SSL server certificate check, per RFC 6125 [RFC6125]. Implementation security considerations can be found in Recommendations for Secure Use of TLS and DTLS [BCP195].

To protect against information disclosure and tampering, confidentiality protection **MUST** be applied using TLS with a ciphersuite that provides confidentiality and integrity protection.

6.2. Impersonation Attacks

TLS certificate checking **MUST** be performed by the client, as described in Section 6.1, when making an authorization server metadata request. Checking that the server certificate is valid for the issuer identifier URL prevents man-in-middle and DNS-based attacks. These attacks could cause a client to be tricked into using an attacker's keys and endpoints, which would enable impersonation of the legitimate authorization server. If an attacker can accomplish this, they can access the resources that the affected client has access to using the authorization server that they are impersonating.

An attacker may also attempt to impersonate an authorization server by publishing a metadata document that contains an "issuer" claim using the issuer identifier URL of the authorization server being impersonated, but with its own endpoints and signing keys. This would enable it to impersonate that authorization server, if accepted by the client. To prevent this, the client MUST ensure that the issuer identifier URL it is using as the prefix for the metadata request exactly matches the value of the "issuer" metadata value in the authorization server metadata document received by the client.

6.3. Publishing Metadata in a Standard Format

Publishing information about the authorization server in a standard format makes it easier for both legitimate clients and attackers to use the authorization server. Whether an authorization server publishes its metadata in an ad-hoc manner or in the standard format defined by this specification, the same defenses against attacks that might be mounted that use this information should be applied.

6.4. Protected Resources

Secure determination of appropriate protected resources to use with an authorization server for all use cases is out of scope of this specification. This specification assumes that the client has a means of determining appropriate protected resources to use with an authorization server and that the client is using the correct metadata for each authorization server. Implementers need to be aware that if an inappropriate protected resource is used by the client, that an attacker may be able to act as a man-in-the-middle proxy to a valid protected resource without it being detected by the authorization server or the client.

The ways to determine the appropriate protected resources to use with an authorization server are in general, application-dependent. For instance, some authorization servers are used with a fixed protected resource or set of protected resources, the locations of which may be well known, or which could be published as metadata values by the authorization server. In other cases, the set of resources that can be used with an authorization server can be dynamically changed by administrative actions. Many other means of determining appropriate associations between authorization servers and protected resources are also possible.

7. IANA Considerations

The following registration procedure is used for the registry established by this specification.

Values are registered on a Specification Required [RFC8126] basis after a two-week review period on the `oauth-ext-review@ietf.org` mailing list, on the advice of one or more Designated Experts. However, to allow for the allocation of values prior to publication, the Designated Experts may approve registration once they are satisfied that such a specification will be published.

Registration requests sent to the mailing list for review should use an appropriate subject (e.g., "Request to register OAuth Authorization Server Metadata: example").

Within the review period, the Designated Experts will either approve or deny the registration request, communicating this decision to the review list and IANA. Denials should include an explanation and, if applicable, suggestions as to how to make the request successful. Registration requests that are undetermined for a period longer than 21 days can be brought to the IESG's attention (using the `iesg@ietf.org` mailing list) for resolution.

Criteria that should be applied by the Designated Experts includes determining whether the proposed registration duplicates existing functionality, determining whether it is likely to be of general applicability or whether it is useful only for a single application, and whether the registration makes sense.

IANA must only accept registry updates from the Designated Experts and should direct all requests for registration to the review mailing list.

It is suggested that multiple Designated Experts be appointed who are able to represent the perspectives of different applications using this specification, in order to enable broadly-informed review of registration decisions. In cases where a registration decision could be perceived as creating a conflict of interest for a particular Expert, that Expert should defer to the judgment of the other Experts.

7.1. OAuth Authorization Server Metadata Registry

This specification establishes the IANA "OAuth Authorization Server Metadata" registry for OAuth 2.0 authorization server metadata names. The registry records the authorization server metadata member and a reference to the specification that defines it.

The Designated Experts must either:

- (a) require that metadata names and values being registered use only printable ASCII characters excluding double quote (`'"`) and backslash

('\'') (the Unicode characters with code points U+0021, U+0023 through U+005B, and U+005D through U+007E), or

(b) if new metadata members or values are defined that use other code points, require that their definitions specify the exact Unicode code point sequences used to represent them. Furthermore, proposed registrations that use Unicode code points that can only be represented in JSON strings as escaped characters must not be accepted.

7.1.1. Registration Template

Metadata Name:

The name requested (e.g., "issuer"). This name is case-sensitive. Names may not match other registered names in a case-insensitive manner (one that would cause a match if the Unicode toLowerCase() operation were applied to both strings) unless the Designated Experts state that there is a compelling reason to allow an exception.

Metadata Description:

Brief description of the metadata (e.g., "Issuer identifier URL").

Change Controller:

For Standards Track RFCs, list the "IESG". 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.

7.1.2. Initial Registry Contents

- o Metadata Name: "issuer"
- o Metadata Description: Authorization server's issuer identifier URL
- o Change Controller: IESG
- o Specification Document(s): Section 2 of [[this specification]]

- o Metadata Name: "authorization_endpoint"
- o Metadata Description: URL of the authorization server's authorization endpoint
- o Change Controller: IESG
- o Specification Document(s): Section 2 of [[this specification]]

- o Metadata Name: "token_endpoint"

- o Metadata Description: URL of the authorization server's token endpoint
- o Change Controller: IESG
- o Specification Document(s): Section 2 of [[this specification]]

- o Metadata Name: "jwks_uri"
- o Metadata Description: URL of the authorization server's JWK Set document
- o Change Controller: IESG
- o Specification Document(s): Section 2 of [[this specification]]

- o Metadata Name: "registration_endpoint"
- o Metadata Description: URL of the authorization server's OAuth 2.0 Dynamic Client Registration Endpoint
- o Change Controller: IESG
- o Specification Document(s): Section 2 of [[this specification]]

- o Metadata Name: "scopes_supported"
- o Metadata Description: JSON array containing a list of the OAuth 2.0 "scope" values that this authorization server supports
- o Change Controller: IESG
- o Specification Document(s): Section 2 of [[this specification]]

- o Metadata Name: "response_types_supported"
- o Metadata Description: JSON array containing a list of the OAuth 2.0 "response_type" values that this authorization server supports
- o Change Controller: IESG
- o Specification Document(s): Section 2 of [[this specification]]

- o Metadata Name: "response_modes_supported"
- o Metadata Description: JSON array containing a list of the OAuth 2.0 "response_mode" values that this authorization server supports
- o Change Controller: IESG
- o Specification Document(s): Section 2 of [[this specification]]

- o Metadata Name: "grant_types_supported"
- o Metadata Description: JSON array containing a list of the OAuth 2.0 grant type values that this authorization server supports
- o Change Controller: IESG
- o Specification Document(s): Section 2 of [[this specification]]

- o Metadata Name: "token_endpoint_auth_methods_supported"
- o Metadata Description: JSON array containing a list of client authentication methods supported by this token endpoint
- o Change Controller: IESG
- o Specification Document(s): Section 2 of [[this specification]]

- o Metadata Name: "token_endpoint_auth_signing_alg_values_supported"

- o Metadata Description: JSON array containing a list of the JWS signing algorithms supported by the token endpoint for the signature on the JWT used to authenticate the client at the token endpoint
- o Change Controller: IESG
- o Specification Document(s): Section 2 of [[this specification]]

- o Metadata Name: "service_documentation"
- o Metadata Description: URL of a page containing human-readable information that developers might want or need to know when using the authorization server
- o Change Controller: IESG
- o Specification Document(s): Section 2 of [[this specification]]

- o Metadata Name: "ui_locales_supported"
- o Metadata Description: Languages and scripts supported for the user interface, represented as a JSON array of BCP47 language tag values
- o Change Controller: IESG
- o Specification Document(s): Section 2 of [[this specification]]

- o Metadata Name: "op_policy_uri"
- o Metadata Description: URL that the authorization server provides to the person registering the client to read about the authorization server's requirements on how the client can use the data provided by the authorization server
- o Change Controller: IESG
- o Specification Document(s): Section 2 of [[this specification]]

- o Metadata Name: "op_tos_uri"
- o Metadata Description: URL that the authorization server provides to the person registering the client to read about the authorization server's terms of service
- o Change Controller: IESG
- o Specification Document(s): Section 2 of [[this specification]]

- o Metadata Name: "revocation_endpoint"
- o Metadata Description: URL of the authorization server's OAuth 2.0 revocation endpoint
- o Change Controller: IESG
- o Specification Document(s): Section 2 of [[this specification]]

- o Metadata Name: "revocation_endpoint_auth_methods_supported"
- o Metadata Description: JSON array containing a list of client authentication methods supported by this revocation endpoint
- o Change Controller: IESG
- o Specification Document(s): Section 2 of [[this specification]]

- o Metadata Name:
"revocation_endpoint_auth_signing_alg_values_supported"
- o Metadata Description: JSON array containing a list of the JWS signing algorithms supported by the revocation endpoint for the signature on the JWT used to authenticate the client at the revocation endpoint
- o Change Controller: IESG
- o Specification Document(s): Section 2 of [[this specification]]

- o Metadata Name: "introspection_endpoint"
- o Metadata Description: URL of the authorization server's OAuth 2.0 introspection endpoint
- o Change Controller: IESG
- o Specification Document(s): Section 2 of [[this specification]]

- o Metadata Name: "introspection_endpoint_auth_methods_supported"
- o Metadata Description: JSON array containing a list of client authentication methods supported by this introspection endpoint
- o Change Controller: IESG
- o Specification Document(s): Section 2 of [[this specification]]

- o Metadata Name:
"introspection_endpoint_auth_signing_alg_values_supported"
- o Metadata Description: JSON array containing a list of the JWS signing algorithms supported by the introspection endpoint for the signature on the JWT used to authenticate the client at the introspection endpoint
- o Change Controller: IESG
- o Specification Document(s): Section 2 of [[this specification]]

- o Metadata Name: "code_challenge_methods_supported"
- o Metadata Description: PKCE code challenge methods supported by this authorization server
- o Change Controller: IESG
- o Specification Document(s): Section 2 of [[this specification]]

7.2. Updated Registration Instructions

This specification adds to the instructions for the Designated Experts of the following IANA registries, both of which are in the "OAuth Parameters" registry [IANA.OAuth.Parameters]:

- o OAuth Access Token Types
- o OAuth Token Endpoint Authentication Methods

IANA has added a link to this specification in the Reference sections of these registries. [[RFC Editor: The above sentence is written in the past tense as it would appear in the final specification, even

though these links won't actually be created until after the IESG has requested publication of the specification. Please delete this note after the links are in place.]]

For these registries, the designated experts must reject registration requests in one registry for values already occurring in the other registry. This is necessary because the "introspection_endpoint_auth_methods_supported" parameter allows for the use of values from either registry. That way, because the values in the two registries will continue to be mutually exclusive, no ambiguities will arise.

7.3. Well-Known URI Registry

This specification registers the well-known URI defined in Section 3 in the IANA "Well-Known URIs" registry [IANA.well-known] established by RFC 5785 [RFC5785].

7.3.1. Registry Contents

- o URI suffix: "oauth-authorization-server"
- o Change controller: IESG
- o Specification document: Section 3 of [[this specification]]
- o Related information: (none)

8. References

8.1. Normative References

- [BCP195] Sheffer, Y., Holz, R., and P. Saint-Andre, "Recommendations for Secure Use of Transport Layer Security (TLS) and Datagram Transport Layer Security (DTLS)", BCP 195, RFC 7525, DOI 10.17487/RFC7525, May 2015, <<http://www.rfc-editor.org/info/bcp195>>.
- [IANA.OAuth.Parameters] IANA, "OAuth Parameters", <<http://www.iana.org/assignments/oauth-parameters>>.
- [JWE] Jones, M. and J. Hildebrand, "JSON Web Encryption (JWE)", RFC 7516, DOI 10.17487/RFC7516, May 2015, <<http://tools.ietf.org/html/rfc7516>>.
- [JWK] Jones, M., "JSON Web Key (JWK)", RFC 7517, DOI 10.17487/RFC7517, May 2015, <<http://tools.ietf.org/html/rfc7517>>.

- [JWS] Jones, M., Bradley, J., and N. Sakimura, "JSON Web Signature (JWS)", RFC 7515, DOI 10.17487/RFC7515, May 2015, <<http://tools.ietf.org/html/rfc7515>>.
- [JWT] Jones, M., Bradley, J., and N. Sakimura, "JSON Web Token (JWT)", RFC 7519, DOI 10.17487/RFC7519, May 2015, <<http://tools.ietf.org/html/rfc7519>>.
- [OAuth.Post] Jones, M. and B. Campbell, "OAuth 2.0 Form Post Response Mode", April 2015, <http://openid.net/specs/oauth-v2-form-post-response-mode-1_0.html>.
- [OAuth.Responses] de Medeiros, B., Ed., Scurtescu, M., Tarjan, P., and M. Jones, "OAuth 2.0 Multiple Response Type Encoding Practices", February 2014, <http://openid.net/specs/oauth-v2-multiple-response-types-1_0.html>.
- [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>>.
- [RFC5246] Dierks, T. and E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.2", RFC 5246, DOI 10.17487/RFC5246, August 2008, <<https://www.rfc-editor.org/info/rfc5246>>.
- [RFC5646] Phillips, A., Ed. and M. Davis, Ed., "Tags for Identifying Languages", BCP 47, RFC 5646, DOI 10.17487/RFC5646, September 2009, <<https://www.rfc-editor.org/info/rfc5646>>.
- [RFC5785] Nottingham, M. and E. Hammer-Lahav, "Defining Well-Known Uniform Resource Identifiers (URIs)", RFC 5785, DOI 10.17487/RFC5785, April 2010, <<https://www.rfc-editor.org/info/rfc5785>>.
- [RFC6125] Saint-Andre, P. and J. Hodges, "Representation and Verification of Domain-Based Application Service Identity within Internet Public Key Infrastructure Using X.509 (PKIX) Certificates in the Context of Transport Layer Security (TLS)", RFC 6125, DOI 10.17487/RFC6125, March 2011, <<https://www.rfc-editor.org/info/rfc6125>>.
- [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>>.

- [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>>.
- [RFC7033] Jones, P., Salgueiro, G., Jones, M., and J. Smarr, "WebFinger", RFC 7033, DOI 10.17487/RFC7033, September 2013, <<https://www.rfc-editor.org/info/rfc7033>>.
- [RFC7159] Bray, T., Ed., "The JavaScript Object Notation (JSON) Data Interchange Format", RFC 7159, DOI 10.17487/RFC7159, March 2014, <<https://www.rfc-editor.org/info/rfc7159>>.
- [RFC7591] Richer, J., Ed., Jones, M., Bradley, J., Machulak, M., and P. Hunt, "OAuth 2.0 Dynamic Client Registration Protocol", RFC 7591, DOI 10.17487/RFC7591, July 2015, <<https://www.rfc-editor.org/info/rfc7591>>.
- [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/info/rfc7636>>.
- [RFC7662] Richer, J., Ed., "OAuth 2.0 Token Introspection", RFC 7662, DOI 10.17487/RFC7662, October 2015, <<https://www.rfc-editor.org/info/rfc7662>>.
- [RFC8126] Cotton, M., Leiba, B., and T. Narten, "Guidelines for Writing an IANA Considerations Section in RFCs", BCP 26, RFC 8126, DOI 10.17487/RFC8126, June 2017, <<https://www.rfc-editor.org/info/rfc8126>>.
- [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>>.
- [UNICODE] The Unicode Consortium, "The Unicode Standard", <<http://www.unicode.org/versions/latest/>>.
- [USA15] Davis, M. and K. Whistler, "Unicode Normalization Forms", Unicode Standard Annex 15, June 2015, <<http://www.unicode.org/reports/tr15/>>.

8.2. Informative References

- [I-D.ietf-oauth-mix-up-mitigation]
Jones, M., Bradley, J., and N. Sakimura, "OAuth 2.0 Mix-Up Mitigation", draft-ietf-oauth-mix-up-mitigation-01 (work in progress), July 2016.

[IANA.well-known]

IANA, "Well-Known URIs",
<<http://www.iana.org/assignments/well-known-uris>>.

[OpenID.Core]

Sakimura, N., Bradley, J., Jones, M., de Medeiros, B., and
C. Mortimore, "OpenID Connect Core 1.0", November 2014,
<http://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", November 2014,
<[http://openid.net/specs/
openid-connect-discovery-1_0.html](http://openid.net/specs/openid-connect-discovery-1_0.html)>.

[OpenID.Registration]

Sakimura, N., Bradley, J., and M. Jones, "OpenID Connect
Dynamic Client Registration 1.0", November 2014,
<[http://openid.net/specs/
openid-connect-registration-1_0.html](http://openid.net/specs/openid-connect-registration-1_0.html)>.

Appendix A. Acknowledgements

This specification is based on the OpenID Connect Discovery 1.0 specification, which was produced by the OpenID Connect working group of the OpenID Foundation. This specification standardizes the de facto usage of the metadata format defined by OpenID Connect Discovery to publish OAuth authorization server metadata.

The authors would like to thank the following people for their reviews of this specification: Shwetha Bhandari, Ben Campbell, Brian Campbell, Brian Carpenter, William Denniss, Vladimir Dzhuvinov, Donald Eastlake, Samuel Erdtman, George Fletcher, Dick Hardt, Phil Hunt, Alexey Melnikov, Tony Nadalin, Mark Nottingham, Eric Rescorla, Justin Richer, Adam Roach, Hannes Tschofenig, and Hans Zandbelt.

Appendix B. Document History

[[to be removed by the RFC Editor before publication as an RFC]]

-10

- o Clarified the meaning of "case-insensitive", as suggested by Alexey Melnikov.

-09

- o Revised the transformation between the issuer identifier and the authorization server metadata location to conform to BCP 190, as suggested by Adam Roach.
- o Defined the characters allowed in registered metadata names and values, as suggested by Alexey Melnikov.
- o Changed to using the RFC 8174 boilerplate instead of the RFC 2119 boilerplate, as suggested by Ben Campbell.
- o Acknowledged additional reviewers.

-08

- o Changed the "authorization_endpoint" to be REQUIRED only when grant types are supported that use the authorization endpoint.
- o Added the statement, to provide historical context, that this specification standardizes the de facto usage of the metadata format defined by OpenID Connect Discovery to publish OAuth authorization server metadata.
- o Applied clarifications suggested by Mark Nottingham about when application-specific well-known suffixes are and are not appropriate.
- o Acknowledged additional reviewers.

-07

- o Applied clarifications suggested by EKR.

-06

- o Incorporated resolutions to working group last call comments.

-05

- o Removed the "protected_resources" element and the reference to draft-jones-oauth-resource-metadata.

-04

- o Added the ability to list protected resources with the "protected_resources" element.
- o Added ability to provide signed metadata with the "signed_metadata" element.

- o Removed "Discovery" from the name, since this is now just about authorization server metadata.

-03

- o Changed term "issuer URL" to "issuer identifier" for terminology consistency, paralleling the same terminology consistency change in the mix-up mitigation spec.

-02

- o Changed the title to OAuth 2.0 Authorization Server Discovery Metadata.
- o Made "jwks_uri" and "registration_endpoint" OPTIONAL.
- o Defined the well-known URI string "/.well-known/oauth-authorization-server".
- o Added security considerations about publishing authorization server discovery metadata in a standard format.
- o Added security considerations about protected resources.
- o Added more information to the "grant_types_supported" and "response_types_supported" definitions.
- o Referenced the working group Mix-Up Mitigation draft.
- o Changed some example metadata values.
- o Acknowledged individuals for their contributions to the specification.

-01

- o Removed WebFinger discovery.
- o Clarified the relationship between the issuer identifier URL and the well-known URI path relative to it at which the discovery metadata document is located.

-00

- o Created the initial working group version based on draft-jones-oauth-discovery-01, with no normative changes.

Authors' Addresses

Michael B. Jones
Microsoft

Email: mbj@microsoft.com
URI: <http://self-issued.info/>

Nat Sakimura
Nomura Research Institute, Ltd.

Email: n-sakimura@nri.co.jp
URI: <http://nat.sakimura.org/>

John Bradley
Ping Identity

Email: ve7jtb@ve7jtb.com
URI: <http://www.thread-safe.com/>

Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: August 28, 2017

J. Bradley
Ping Identity
P. Hunt
Oracle Corporation
M. Jones
Microsoft
H. Tschofenig
ARM Limited
February 24, 2017

OAuth 2.0 Proof-of-Possession: Authorization Server to Client Key
Distribution
draft-ietf-oauth-pop-key-distribution-03

Abstract

RFC 6750 specified the bearer token concept for securing access to protected resources. Bearer tokens need to be protected in transit as well as at rest. When a client requests access to a protected resource it hands-over the bearer token to the resource server.

The OAuth 2.0 Proof-of-Possession security concept extends bearer token security and requires the client to demonstrate possession of a key when accessing a protected resource.

This document describes how the client obtains this keying material from the authorization server.

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 <http://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 August 28, 2017.

Copyright Notice

Copyright (c) 2017 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<http://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1. Introduction	2
2. Terminology	4
3. Audience	4
3.1. Audience Parameter	5
3.2. Processing Instructions	5
4. Symmetric Key Transport	6
4.1. Client-to-AS Request	6
4.2. Client-to-AS Response	7
5. Asymmetric Key Transport	9
5.1. Client-to-AS Request	9
5.2. Client-to-AS Response	11
6. Token Types and Algorithms	12
7. Security Considerations	13
8. IANA Considerations	14
9. Acknowledgements	15
10. References	15
10.1. Normative References	15
10.2. Informative References	16
Appendix A. Augmented Backus-Naur Form (ABNF) Syntax	17
A.1. 'aud' Syntax	17
A.2. 'key' Syntax	18
A.3. 'alg' Syntax	18
Authors' Addresses	18

1. Introduction

The work on additional security mechanisms beyond OAuth 2.0 bearer tokens [12] is motivated in [17], which also outlines use cases, requirements and an architecture. This document defines the ability for the client indicate support for this functionality and to obtain keying material from the authorization server. As an outcome of the

exchange between the client and the authorization server is an access token that is bound to keying material. Clients that access protected resources then need to demonstrate knowledge of the secret key that is bound to the access token.

To best describe the scope of this specification, the OAuth 2.0 protocol exchange sequence is shown in Figure 1. The extension defined in this document piggybacks on the message exchange marked with (C) and (D).

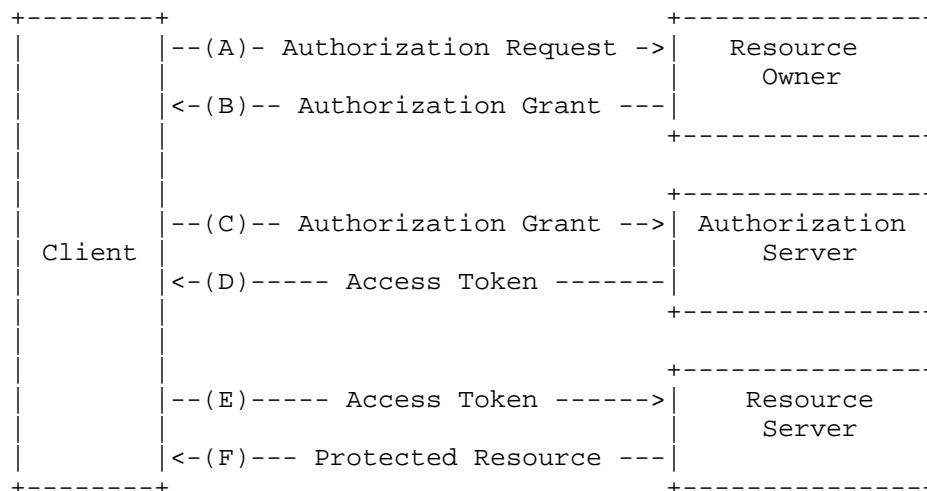


Figure 1: Abstract OAuth 2.0 Protocol Flow

In OAuth 2.0 [2] access tokens can be obtained via authorization grants and using refresh tokens. The core OAuth specification defines four authorization grants, see Section 1.3 of [2], and [14] adds an assertion-based authorization grant to that list. The token endpoint, which is described in Section 3.2 of [2], is used with every authorization grant except for the implicit grant type. In the implicit grant type the access token is issued directly.

This document extends the functionality of the token endpoint, i.e., the protocol exchange between the client and the authorization server, to allow keying material to be bound to an access token. Two types of keying material can be bound to an access token, namely symmetric keys and asymmetric keys. Conveying symmetric keys from the authorization server to the client is described in Section 4 and the procedure for dealing with asymmetric keys is described in Section 5.

2. Terminology

The key words 'MUST', 'MUST NOT', 'REQUIRED', 'SHALL', 'SHALL NOT', 'SHOULD', 'SHOULD NOT', 'RECOMMENDED', 'MAY', and 'OPTIONAL' in this specification are to be interpreted as described in [1].

Session Key:

The term session key refers to fresh and unique keying material established between the client and the resource server. This session key has a lifetime that corresponds to the lifetime of the access token, is generated by the authorization server and bound to the access token.

This document uses the following abbreviations:

JWA: JSON Web Algorithms (JWA) [7]

JWT: JSON Web Token (JWT) [9]

JWS: JSON Web Signature (JWS) [6]

JWK: JSON Web Key (JWK) [5]

JWE: JSON Web Encryption (JWE) [8]

3. Audience

When an authorization server creates an access token, according to the PoP security architecture [17], it may need to know which resource server will process it. This information is necessary when the authorization server applies integrity protection to the JWT using a symmetric key and has to select the key of the resource server that has to verify it. The authorization server also requires this audience information if it has to encrypt a symmetric session key inside the access token using a long-term symmetric key.

This section defines a new header that is used by the client to indicate what protected resource at which resource server it wants to access. This information may subsequently also be communicated by the authorization server securely to the resource server, for example within the audience field of the access token.

QUESTION: A benefit of asymmetric cryptography is to allow clients to request a PoP token for use with multiple resource servers. The downside of that approach is linkability since different resource servers will be able to link individual requests to the same client.

(The same is true if the a single public key is linked with PoP tokens used with different resource servers.) Nevertheless, to support the functionality the audience parameter could carry an array of values. Is this desirable?

3.1. Audience Parameter

The client constructs the access token request to the token endpoint by adding the 'aud' parameter using the "application/x-www-form-urlencoded" format with a character encoding of UTF-8 in the HTTP request entity-body.

The URI included in the aud parameter MUST be an absolute URI as defined by Section 4.3 of [3]. It MAY include an "application/x-www-form-urlencoded" formatted query component (Section 3.4 of [3]). The URI MUST NOT include a fragment component.

The ABNF syntax for the 'aud' element is defined in Appendix A.

3.2. Processing Instructions

Step (0): As an initial step the client typically determines the resource server it wants to interact with. This may, for example, happen as part of a discovery procedure or via manual configuration.

Step (1): The client starts the OAuth 2.0 protocol interaction based on the selected grant type.

Step (2): When the client interacts with the token endpoint to obtain an access token it MUST populate the newly defined 'audience' parameter with the information obtained in step (0).

Step (2): The authorization server who obtains the request from the client needs to parse it to determine whether the provided audience value matches any of the resource servers it has a relationship with. If the authorization server fails to parse the provided value it MUST reject the request using an error response with the error code "invalid_request". If the authorization server does not consider the resource server acceptable it MUST return an error response with the error code "access_denied". In both cases additional error information may be provided via the error_description, and the error_uri parameters. If the request has, however, been verified successfully then the authorization server MUST include the audience claim into the access token with the value copied from the audience field provided by the client. In case the access token is encoded using the JSON Web Token format [9] the "aud" claim MUST be used. The access token, if

passed per value, MUST be protected against modification by either using a digital signature or a keyed message digest. Access tokens can also be passed by reference, which then requires the token introspection endpoint (or a similiar, proprietary protocol mechanism) to be used. The authorization server returns the access token to the client, as specified in [2].

Subsequent steps for the interaction between the client and the resource server are beyond the scope of this document.

4. Symmetric Key Transport

4.1. Client-to-AS Request

In case a symmetric key shall be bound to an PoP token the following procedure is applicable. In the request message from the OAuth client to the OAuth authorization server the following parameters MAY be included:

token_type: OPTIONAL. See Section 6 for more details.

alg: OPTIONAL. See Section 6 for more details.

These two new parameters are optional in the case where the authorization server has prior knowledge of the capabilities of the client otherwise these two parameters are required. This prior knowledge may, for example, be set by the use of a dynamic client registration protocol exchange.

QUESTION: Should we register these two parameters for use with the dynamic client registration protocol?

For example, the client makes the following HTTP request using TLS (extra line breaks are for display purposes only).


```
POST /token HTTP/1.1
Host: server.example.com
Authorization: Basic czZCaGRSa3F0MzpnWDFmQmF0M2JW
Content-Type: application/x-www-form-urlencoded;charset=UTF-8

grant_type=authorization_code
&code=SpIxl0BeZQQYbYS6WxSbIA
&redirect_uri=https%3A%2F%2Fclient%2Eexample%2Ecom%2Fcb
&token_type=pop
&alg=HS256
```

Example Request to the Authorization Server

4.2. Client-to-AS Response

If the access token request has been successfully verified by the authorization server and the client is authorized to obtain a PoP token for the indicated resource server, the authorization server issues an access token and optionally a refresh token. If client authentication failed or is invalid, the authorization server returns an error response as described in Section 5.2 of [2].

The authorization server **MUST** include an access token and a 'key' element in a successful response. The 'key' parameter either contains a plain JWK structure or a JWK encrypted with a JWE. The difference between the two approaches is the following:

Plain JWK: If the JWK container is placed in the 'key' element then the security of the overall PoP architecture relies on Transport Layer Security (TLS) between the authorization server and the client. Figure 2 illustrates an example response using a plain JWK for key transport from the authorization server to the client.

JWK protected by a JWE: If the JWK container is protected by a JWE then additional security protection at the application layer is provided between the authorization server and the client beyond the use of TLS. This approach is a reasonable choice, for example, when a hardware security module is available on the client device and confidentiality protection can be offered directly to this hardware security module.

Note that there are potentially two JSON-encoded structures in the response, namely the access token (with the recommended JWT encoding) and the actual key transport mechanism itself. Note, however, that the two structures serve a different purpose and are consumed by different parties. The access token is created by the authorization server and processed by the resource server (and opaque to the

client) whereas the key transport payload is created by the authorization server and processed by the client; it is never forwarded to the resource server.

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

{
  "access_token": "SlAV32hkKG ...
    (remainder of JWT omitted for brevity;
    JWT contains JWK in the cnf claim)",
  "token_type": "pop",
  "expires_in": 3600,
  "refresh_token": "8xLOxBtZp8",
  "key": "eyJhbGciOiJSU0ExXzUi ...
    (remainder of plain JWK omitted for brevity)"
}
```

Figure 2: Example: Response from the Authorization Server (Symmetric Variant)

The content of the key parameter, which is a JWK in our example, is shown in Figure 3.

```
{
  "kty": "oct",
  "kid": "id123",
  "alg": "HS256",
  "k": "ZoRSOrFzN_FzUA5XKMYoVHyzzff5oRJxl-IXRtztJ6uE"
}
```

Figure 3: Example: Key Transport to Client via a JWK

The content of the 'access_token' in JWT format contains the 'cnf' (confirmation) claim, as shown in Figure 4. The confirmation claim is defined in [10]. The digital signature or the keyed message digest offering integrity protection is not shown in this example but MUST be present in a real deployment to mitigate a number of security threats. Those security threats are described in [17].

The JWK in the key element of the response from the authorization server, as shown in Figure 2, contains the same session key as the JWK inside the access token, as shown in Figure 4. It is, in this

example, protected by TLS and transmitted from the authorization server to the client (for processing by the client).

```
{
  "iss": "https://server.example.com",
  "sub": "24400320",
  "aud": "s6BhdRkqt3",
  "nonce": "n-0S6_WzA2Mj",
  "exp": 1311281970,
  "iat": 1311280970,
  "cnf": {
    "jwk":
      "JDLUhTMjU2IiwiY3R5Ijoi ...
      (remainder of JWK protected by JWE omitted for brevity)"
  }
}
```

Figure 4: Example: Access Token in JWT Format

Note: When the JWK inside the access token contains a symmetric key it MUST be confidentiality protected using a JWE to maintain the security goals of the PoP architecture, as described in [17] since content is meant for consumption by the selected resource server only.

Note: This document does not impose requirements on the encoding of the access token. The examples used in this document make use of the JWT structure since this is the only standardized format.

If the access token is only a reference then a look-up by the resource server is needed, as described in the token introspection specification [18].

5. Asymmetric Key Transport

5.1. Client-to-AS Request

In case an asymmetric key shall be bound to an access token then the following procedure is applicable. In the request message from the OAuth client to the OAuth authorization server the request MAY include the following parameters:

token_type: OPTIONAL. See Section 6 for more details.

alg: OPTIONAL. See Section 6 for more details.

key: OPTIONAL. This field contains information about the public key the client would like to bind to the access token in the JWK format. If the client does not provide a public key then the authorization server MUST create an ephemeral key pair (considering the information provided by the client) or alternatively respond with an error message. The client may also convey the fingerprint of the public key to the authorization server instead of passing the entire public key along (to conserve bandwidth). [11] defines a way to compute a thumbprint for a JWK and to embed it within the JWK format.

The 'token_type' and the 'alg' parameters are optional in the case where the authorization server has prior knowledge of the capabilities of the client otherwise these two parameters are required.

For example, the client makes the following HTTP request using TLS (extra line breaks are for display purposes only) shown in Figure 5.

```
POST /token HTTP/1.1
Host: server.example.com
Authorization: Basic czZCaGRSa3F0MzpnWDFmQmF0M2JW
Content-Type: application/x-www-form-urlencoded;charset=UTF-8

grant_type=authorization_code
&code=SpIxl0BeZQQYbYS6WxSbIA
&redirect_uri=https%3A%2F%2Fclient%2Eexample%2Ecom%2Fcb
&token_type=pop
&alg=RS256
&key=eyJhbGciOiJSU0ExXzUi ...
(remainder of JWK omitted for brevity)
```

Figure 5: Example Request to the Authorization Server (Asymmetric Key Variant)

As shown in Figure 6 the content of the 'key' parameter contains the RSA public key the client would like to associate with the access token.

```
{
  "kty": "RSA",
  "n": "0vx7agoebGcQSuuPiLJXZptN9nndrQmbXEps2aiAFbWhM78LhWx
4cbbfAAAtVT86zwulRK7aPFFxuhDR1L6tSoc_BJECPEbWKRXjBZCiFV4n3oknjhMs
tn64tZ_2W-5JsGY4Hc5n9yBXArwl93lqt7_RN5w6Cf0h4QyQ5v-65YGjQR0_FDW2
QvzqY368QQMicAtaSqzs8KJZgnYb9c7d0zgdAZHzu6QMqvRL5hajrnln91CbOpbI
SD08qNLyrdkt-bFTWhAI4vMQFh6WeZu0fM4lFd2NcRwr3XPksINHaQ-G_xBniIqb
w0LsljF44-csFCur-kEgU8awapJzKnqDKgw",
  "e": "AQAB",
  "alg": "RS256",
  "kid": "id123"
}
```

Figure 6: Client Providing Public Key to Authorization Server

5.2. Client-to-AS Response

If the access token request is valid and authorized, the authorization server issues an access token and optionally a refresh token. If the request client authentication failed or is invalid, the authorization server returns an error response as described in Section 5.2 of [2].

The authorization server also places information about the public key used by the client into the access token to create the binding between the two. The new token type "public_key" is placed into the 'token_type' parameter.

An example of a successful response is shown in Figure 7.

```
HTTP/1.1 200 OK
Content-Type: application/json; charset=UTF-8
Cache-Control: no-store
Pragma: no-cache

{
  "access_token": "2YotnFZFE....jrlzCsicMWpAA",
  "token_type": "pop",
  "alg": "RS256",
  "expires_in": 3600,
  "refresh_token": "tGzv3JOkF0XG5Qx2TlKWIA"
}
```

Figure 7: Example: Response from the Authorization Server (Asymmetric Variant)

The content of the 'access_token' field contains an encoded JWT with the following structure, as shown in Figure 8. The digital signature

or the keyed message digest offering integrity protection is not shown (but must be present).

```
{
  "iss": "xas.example.com",
  "aud": "http://auth.example.com",
  "exp": "1361398824",
  "nbf": "1360189224",
  "cnf": {
    "jwk": { "kty": "RSA",
              "n": "0vx7agoebGcQSuuPiLJXZptN9nndrQmbXEps2aiAFbWhM78LhWx
4cbbfAAAtVT86zwulRK7aPFFxuhDR1L6tSoc_BJECPEbWKRXjBZCiFV4n3oknjhMs
tn64tZ_2W-5JsGY4Hc5n9yBXArwl93lqt7_RN5w6Cf0h4QyQ5v-65YGjQR0_FDW2
QvzqY368QQMicAtaSqzs8KJZgnYb9c7d0zgdAZHzu6QMqvRL5hajrnl9lCbOpbI
SD08qNLyrdkt-bFTWhAI4vMQFh6WeZu0fM4lFd2NcRwr3XPksINHq-G_xBniIqb
w0LsljF44-csFCur-kEgU8awapJzKnqDKgw",
              "e": "AQAB",
              "alg": "RS256",
              "kid": "id123"
            }
  }
}
```

Figure 8: Example: Access Token Structure (Asymmetric Variant)

Note: In this example there is no need for the authorization server to convey further keying material to the client since the client is already in possession of the private RSA key.

6. Token Types and Algorithms

To allow clients to indicate support for specific token types and respective algorithms they need to interact with authorization servers. They can either provide this information out-of-band, for example, via pre-configuration or up-front via the dynamic client registration protocol [16].

The value in the 'alg' parameter together with value from the 'token_type' parameter allow the client to indicate the supported algorithms for a given token type. The token type refers to the specification used by the client to interact with the resource server to demonstrate possession of the key. The 'alg' parameter provides further information about the algorithm, such as whether a symmetric or an asymmetric crypto-system is used. Hence, a client supporting a specific token type also knows how to populate the values to the 'alg' parameter.

The value for the 'token_type' MUST be taken from the 'OAuth Access Token Types' registry created by [2].

This document does not register a new value for the OAuth Access Token Types registry nor does it define values to be used for the 'alg' parameter since this is the responsibility of specifications defining the mechanism for clients interacting with resource servers. An example of such specification can be found in [19].

The values in the 'alg' parameter are case-sensitive. If the client supports more than one algorithm then each individual value MUST be separated by a space.

7. Security Considerations

[17] describes the architecture for the OAuth 2.0 proof-of-possession security architecture, including use cases, threats, and requirements. This requirements describes one solution component of that architecture, namely the mechanism for the client to interact with the authorization server to either obtain a symmetric key from the authorization server, to obtain an asymmetric key pair, or to offer a public key to the authorization. In any case, these keys are then bound to the access token by the authorization server.

To summarize the main security recommendations: A large range of threats can be mitigated by protecting the contents of the access token by using a digital signature or a keyed message digest. Consequently, the token integrity protection MUST be applied to prevent the token from being modified, particularly since it contains a reference to the symmetric key or the asymmetric key. If the access token contains the symmetric key (see Section 2.2 of [10] for a description about how symmetric keys can be securely conveyed within the access token) this symmetric key MUST be encrypted by the authorization server with a long-term key shared with the resource server.

To deal with token redirect, it is important for the authorization server to include the identity of the intended recipient (the audience), typically a single resource server (or a list of resource servers), in the token. Using a single shared secret with multiple authorization server to simplify key management is NOT RECOMMENDED since the benefit from using the proof-of-possession concept is significantly reduced.

Token replay is also not possible since an eavesdropper will also have to obtain the corresponding private key or shared secret that is bound to the access token. Nevertheless, it is good practice to

limit the lifetime of the access token and therefore the lifetime of associated key.

The authorization server MUST offer confidentiality protection for any interactions with the client. This step is extremely important since the client will obtain the session key from the authorization server for use with a specific access token. Not using confidentiality protection exposes this secret (and the access token) to an eavesdropper thereby making the OAuth 2.0 proof-of-possession security model completely insecure. OAuth 2.0 [2] relies on TLS to offer confidentiality protection and additional protection can be applied using the JWK [5] offered security mechanism, which would add an additional layer of protection on top of TLS for cases where the keying material is conveyed, for example, to a hardware security module. Which version(s) of TLS ought to be implemented will vary over time, and depend on the widespread deployment and known security vulnerabilities at the time of implementation. At the time of this writing, TLS version 1.2 [4] is the most recent version. The client MUST validate the TLS certificate chain when making requests to protected resources, including checking the validity of the certificate.

Similarly to the security recommendations for the bearer token specification [12] developers MUST ensure that the ephemeral credentials (i.e., the private key or the session key) is not leaked to third parties. An adversary in possession of the ephemeral credentials bound to the access token will be able to impersonate the client. Be aware that this is a real risk with many smart phone app and Web development environments.

Clients can at any time request a new proof-of-possession capable access token. Using a refresh token to regularly request new access tokens that are bound to fresh and unique keys is important. Keeping the lifetime of the access token short allows the authorization server to use shorter key sizes, which translate to a performance benefit for the client and for the resource server. Shorter keys also lead to shorter messages (particularly with asymmetric keying material).

When authorization servers bind symmetric keys to access tokens then they SHOULD scope these access tokens to a specific permissions.

8. IANA Considerations

This specification registers the following parameters in the OAuth Parameters Registry established by [2].

Parameter name: alg

Parameter usage location: token request, token response,
authorization response

Change controller: IETF

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

Related information: None

Parameter name: key

Parameter usage location: token request, token response,
authorization response

Change controller: IETF

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

Related information: None

Parameter name: aud

Parameter usage location: token request

Change controller: IETF

Specification document(s): [[This document.]]

Related information: None

9. Acknowledgements

We would like to thank Chuck Mortimore for his review comments.

10. References

10.1. Normative References

- [1] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <<http://www.rfc-editor.org/info/rfc2119>>.
- [2] Hardt, D., Ed., "The OAuth 2.0 Authorization Framework", RFC 6749, DOI 10.17487/RFC6749, October 2012, <<http://www.rfc-editor.org/info/rfc6749>>.

- [3] Berners-Lee, T., Fielding, R., and L. Masinter, "Uniform Resource Identifier (URI): Generic Syntax", STD 66, RFC 3986, DOI 10.17487/RFC3986, January 2005, <<http://www.rfc-editor.org/info/rfc3986>>.
- [4] Dierks, T. and E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.2", RFC 5246, DOI 10.17487/RFC5246, August 2008, <<http://www.rfc-editor.org/info/rfc5246>>.
- [5] Jones, M., "JSON Web Key (JWK)", RFC 7517, DOI 10.17487/RFC7517, May 2015, <<http://www.rfc-editor.org/info/rfc7517>>.
- [6] Jones, M., Bradley, J., and N. Sakimura, "JSON Web Signature (JWS)", RFC 7515, DOI 10.17487/RFC7515, May 2015, <<http://www.rfc-editor.org/info/rfc7515>>.
- [7] Jones, M., "JSON Web Algorithms (JWA)", RFC 7518, DOI 10.17487/RFC7518, May 2015, <<http://www.rfc-editor.org/info/rfc7518>>.
- [8] Jones, M. and J. Hildebrand, "JSON Web Encryption (JWE)", RFC 7516, DOI 10.17487/RFC7516, May 2015, <<http://www.rfc-editor.org/info/rfc7516>>.
- [9] Jones, M., Bradley, J., and N. Sakimura, "JSON Web Token (JWT)", RFC 7519, DOI 10.17487/RFC7519, May 2015, <<http://www.rfc-editor.org/info/rfc7519>>.
- [10] 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, <<http://www.rfc-editor.org/info/rfc7800>>.
- [11] Jones, M. and N. Sakimura, "JSON Web Key (JWK) Thumbprint", RFC 7638, DOI 10.17487/RFC7638, September 2015, <<http://www.rfc-editor.org/info/rfc7638>>.

10.2. Informative References

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

- [13] Crocker, D., Ed. and P. Overell, "Augmented BNF for Syntax Specifications: ABNF", STD 68, RFC 5234, DOI 10.17487/RFC5234, January 2008, <<http://www.rfc-editor.org/info/rfc5234>>.
- [14] 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, <<http://www.rfc-editor.org/info/rfc7521>>.
- [15] 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, <<http://www.rfc-editor.org/info/rfc7636>>.
- [16] Richer, J., Ed., Jones, M., Bradley, J., Machulak, M., and P. Hunt, "OAuth 2.0 Dynamic Client Registration Protocol", RFC 7591, DOI 10.17487/RFC7591, July 2015, <<http://www.rfc-editor.org/info/rfc7591>>.
- [17] Hunt, P., Richer, J., Mills, W., Mishra, P., and H. Tschofenig, "OAuth 2.0 Proof-of-Possession (PoP) Security Architecture", draft-ietf-oauth-pop-architecture-08 (work in progress), July 2016.
- [18] Richer, J., Ed., "OAuth 2.0 Token Introspection", RFC 7662, DOI 10.17487/RFC7662, October 2015, <<http://www.rfc-editor.org/info/rfc7662>>.
- [19] Richer, J., Bradley, J., and H. Tschofenig, "A Method for Signing HTTP Requests for OAuth", draft-ietf-oauth-signed-http-request-03 (work in progress), August 2016.

Appendix A. Augmented Backus-Naur Form (ABNF) Syntax

This section provides Augmented Backus-Naur Form (ABNF) syntax descriptions for the elements defined in this specification using the notation of [13].

A.1. 'aud' Syntax

The ABNF syntax is defined as follows where by the "URI-reference" definition is taken from [3]:

aud = URI-reference

A.2. 'key' Syntax

The "key" element is defined in Section 4 and Section 5:

key = 1*VSCHAR

A.3. 'alg' Syntax

The "alg" element is defined in Section 6:

alg = alg-token *(SP alg-token)

alg-token = 1*NQCHAR

Authors' Addresses

John Bradley
Ping Identity

Email: ve7jtb@ve7jtb.com
URI: <http://www.thread-safe.com/>

Phil Hunt
Oracle Corporation

Email: phil.hunt@yahoo.com
URI: <http://www.indepdentid.com>

Michael B. Jones
Microsoft

Email: mbj@microsoft.com
URI: <http://self-issued.info/>

Hannes Tschofenig
ARM Limited
Austria

Email: Hannes.Tschofenig@gmx.net
URI: <http://www.tschofenig.priv.at>

OAuth Working Group
Internet-Draft
Intended status: Standards Track
Expires: April 22, 2019

M. Jones
Microsoft
B. Campbell
Ping Identity
J. Bradley
Yubico
W. Denniss
Google
October 19, 2018

OAuth 2.0 Token Binding
draft-ietf-oauth-token-binding-08

Abstract

This specification enables OAuth 2.0 implementations to apply Token Binding to Access Tokens, Authorization Codes, Refresh Tokens, JWT Authorization Grants, and JWT Client Authentication. This cryptographically binds these tokens to a client's Token Binding key pair, possession of which is proven on the TLS connections over which the tokens are intended to be used. This use of Token Binding protects these tokens from man-in-the-middle and token export and replay attacks.

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 April 22, 2019.

Copyright Notice

Copyright (c) 2018 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<https://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1. Introduction	3
1.1. Requirements Notation and Conventions	3
1.2. Terminology	3
2. Token Binding for Refresh Tokens	4
2.1. Example Token Binding for Refresh Tokens	4
3. Token Binding for Access Tokens	6
3.1. Access Tokens Issued from the Authorization Endpoint . .	7
3.1.1. Example Access Token Issued from the Authorization Endpoint	8
3.2. Access Tokens Issued from the Token Endpoint	9
3.2.1. Example Access Token Issued from the Token Endpoint .	9
3.3. Protected Resource Token Binding Validation	11
3.3.1. Example Protected Resource Request	11
3.4. Representing Token Binding in JWT Access Tokens	11
3.5. Representing Token Binding in Introspection Responses . .	12
4. Token Binding Metadata	13
4.1. Token Binding Client Metadata	13
4.2. Token Binding Authorization Server Metadata	13
5. Token Binding for Authorization Codes	14
5.1. Native Application Clients	14
5.1.1. Code Challenge	14
5.1.1.1. Example Code Challenge	15
5.1.2. Code Verifier	15
5.1.2.1. Example Code Verifier	16
5.2. Web Server Clients	16
5.2.1. Code Challenge	17
5.2.1.1. Example Code Challenge	17
5.2.2. Code Verifier	18
5.2.2.1. Example Code Verifier	18
6. Token Binding JWT Authorization Grants and Client Authentication	19
6.1. JWT Format and Processing Requirements	19
6.2. Token Bound JWTs for Client Authentication	20
6.3. Token Bound JWTs for as Authorization Grants	20
7. Security Considerations	21
7.1. Phasing in Token Binding	21

7.2. Binding of Refresh Tokens	21
8. IANA Considerations	22
8.1. OAuth Dynamic Client Registration Metadata Registration	22
8.1.1. Registry Contents	22
8.2. OAuth Authorization Server Metadata Registration	23
8.2.1. Registry Contents	23
8.3. PKCE Code Challenge Method Registration	23
8.3.1. Registry Contents	23
9. Token Endpoint Authentication Method Registration	23
9.1. Registry Contents	24
10. Sub-Namespace Registrations	24
10.1. Registry Contents	24
11. References	24
11.1. Normative References	24
11.2. Informative References	26
Appendix A. Acknowledgements	27
Appendix B. Document History	27
Authors' Addresses	29

1. Introduction

This specification enables OAuth 2.0 [RFC6749] implementations to apply Token Binding (TLS Extension for Token Binding Protocol Negotiation [RFC8472], The Token Binding Protocol Version 1.0 [RFC8471] and Token Binding over HTTP [RFC8473]) to Access Tokens, Authorization Codes, Refresh Tokens, JWT Authorization Grants, and JWT Client Authentication. This cryptographically binds these tokens to a client's Token Binding key pair, possession of which is proven on the TLS connections over which the tokens are intended to be used. This use of Token Binding protects these tokens from man-in-the-middle and token export and replay attacks.

1.1. Requirements Notation and Conventions

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.

1.2. Terminology

This specification uses the terms "Access Token", "Authorization Code", "Authorization Endpoint", "Authorization Server", "Client", "Protected Resource", "Refresh Token", and "Token Endpoint" defined by OAuth 2.0 [RFC6749], the terms "Claim" and "JSON Web Token (JWT)" defined by JSON Web Token (JWT) [JWT], the term "User Agent" defined by RFC 7230 [RFC7230], and the terms "Provided", "Referred", "Token

Binding" and "Token Binding ID" defined by Token Binding over HTTP [RFC8473].

2. Token Binding for Refresh Tokens

Token Binding of refresh tokens is a straightforward first-party scenario, applying term "first-party" as used in Token Binding over HTTP [RFC8473]. It cryptographically binds the refresh token to the client's Token Binding key pair, possession of which is proven on the TLS connections between the client and the token endpoint. This case is straightforward because the refresh token is both retrieved by the client from the token endpoint and sent by the client to the token endpoint. Unlike the federation use cases described in Token Binding over HTTP [RFC8473], Section 4, and the access token case described in the next section, only a single TLS connection is involved in the refresh token case.

Token Binding a refresh token requires that the authorization server do two things. First, when refresh token is sent to the client, the authorization server needs to remember the Provided Token Binding ID and remember its association with the issued refresh token. Second, when a token request containing a refresh token is received at the token endpoint, the authorization server needs to verify that the Provided Token Binding ID for the request matches the remembered Token Binding ID associated with the refresh token. If the Token Binding IDs do not match, the authorization server should return an error in response to the request.

How the authorization server remembers the association between the refresh token and the Token Binding ID is an implementation detail that beyond the scope of this specification. Some authorization servers will choose to store the Token Binding ID (or a cryptographic hash of it, such a SHA-256 hash [SHS]) in the refresh token itself, provided it is integrity-protected, thus reducing the amount of state to be kept by the server. Other authorization servers will add the Token Binding ID value (or a hash of it) to an internal data structure also containing other information about the refresh token, such as grant type information. These choices make no difference to the client, since the refresh token is opaque to it.

2.1. Example Token Binding for Refresh Tokens

This section provides an example of what the interactions around a Token Bound refresh token might look like, along with some details of the involved processing. Token Binding of refresh tokens is most useful for native application clients so the example has protocol elements typical of a native client flow. Extra line breaks in all examples are for display purposes only.

A native application client makes the following access token request with an authorization code using a TLS connection where Token Binding has been negotiated. A PKCE "code_verifier" is included because use of PKCE is considered best practice for native application clients [BCP212]. The base64url-encoded representation of the exported keying material (EKM) from that TLS connection is "p6ZuSwfl6pIe8es5KyeV76T4swZmQp0_awd27jHfrbo", which is needed to validate the Token Binding Message.

```
POST /as/token.oauth2 HTTP/1.1
Host: server.example.com
Content-Type: application/x-www-form-urlencoded
Sec-Token-Binding: AIkAAgBBQGto7hHRR0Y5nkOWqc9KNfwW95dEFmSI_tCZ_Cbl
7LWlt6Xjp3DbjiDjavGfiKP2HV_2JSE42VzmKOVVV8m7eqAAQOKiDKl0i0z6v4X5B
P7uc0pFestVZ42TTOdJmoHpji06Qq3jsCiCRSJx9ck2fWJYx8tLVXRZPATB3x6c24
ay0ZEAAA

grant_type=authorization_code&code=4bwcZesc7Xacc330ltc66Wxk8EAfp9j2
&code_verifier=2x6_ylS390-8V7jaT9wj.8qP9nKmYcf.V-rD9O4r_1
&client_id=example-native-client-id
```

Figure 1: Initial Request with Code

A refresh token is issued in response to the prior request. Although it looks like a typical response to the client, the authorization server has bound the refresh token to the Provided Token Binding ID from the encoded Token Binding message in the "Sec-Token-Binding" header of the request. In this example, that binding is done by saving the Token Binding ID alongside other information about the refresh token in some server side persistent storage. The base64url-encoded representation of that Token Binding ID is "AgBBQGto7hHRR0Y5nkOWqc9KNfwW95dEFmSI_tCZ_Cbl7LWlt6Xjp3DbjiDjavGfiKP2HV_2JSE42VzmKOVVV8m7eqA".

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

{
  "access_token": "EdRs7qMrLb167Z9fV2dcwoLTC",
  "refresh_token": "ACClZEIQTjW9arT9GOJGGd7QNwqOMmUYfsJTiv8his4",
  "token_type": "Bearer",
  "expires_in": 3600
}
```

Figure 2: Successful Response

When the access token expires, the client requests a new one with a refresh request to the token endpoint. In this example, the request is made on a new TLS connection so the EKM (base64url-encoded: "va-84Ukw4Zqfd7uWotFrAJda96WwgbdaPDX2knoOiAE") and signature in the Token Binding Message are different than in the initial request.

```
POST /as/token.oauth2 HTTP/1.1
Host: server.example.com
Content-Type: application/x-www-form-urlencoded
Sec-Token-Binding: AikAAgBBQGto7hHRR0Y5nkOWqc9KNfwW95dEFmSI_tCZ_Cbl
  7LWlt6Xjp3DbjiDJavGFikP2HV_2JSE42VzmKOVVV8m7eqAAQCpGbaG_YRf27qOra
  L0UT4fsKKjL6PukuOT00qzamoAXxOq7m_id7O3mLpnbsM7kwSxLi7iNHzzDgCAkP
  t3lHwAAA

refresh_token=ACClZEIQTjW9arT9GOJGGd7QNwqOMmUYfsJTiv8his4
&grant_type=refresh_token&client_id=example-native-client-id
```

Figure 3: Refresh Request

However, because the Token Binding ID is long-lived and may span multiple TLS sessions and connections, it is the same as in the initial request. That Token Binding ID is what the refresh token is bound to, so the authorization server is able to verify it and issue a new access token.

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

{
  "access_token": "bwcESCwC4yOCQ8iPsgcn117k7",
  "token_type": "Bearer",
  "expires_in": 3600
}
```

Figure 4: Successful Response

3. Token Binding for Access Tokens

Token Binding for access tokens cryptographically binds the access token to the client's Token Binding key pair, possession of which is proven on the TLS connections between the client and the protected resource. Token Binding is applied to access tokens in a similar manner to that described in Token Binding over HTTP [RFC8473], Section 4 (Federation Use Cases). It also builds upon the mechanisms for Token Binding of ID Tokens defined in OpenID Connect Token Bound Authentication 1.0 [OpenID.TokenBinding].

In the OpenID Connect [OpenID.Core] use case, HTTP redirects are used to pass information between the identity provider and the relying party; this HTTP redirect makes the Token Binding ID of the relying party available to the identity provider as the Referred Token Binding ID, information about which is then added to the ID Token. No such redirect occurs between the authorization server and the protected resource in the access token case; therefore, information about the Token Binding ID for the TLS connection between the client and the protected resource needs to be explicitly communicated by the client to the authorization server to achieve Token Binding of the access token.

This information is passed to the authorization server using the Referred Token Binding ID, just as in the ID Token case. The only difference is that the client needs to explicitly communicate the Token Binding ID of the TLS connection between the client and the protected resource to the Token Binding implementation so that it is sent as the Referred Token Binding ID in the request to the authorization server. This functionality provided by Token Binding implementations is described in Implementation Considerations of Token Binding over HTTP [RFC8473], Section 6.

Note that to obtain this Token Binding ID, the client may need to establish a TLS connection between itself and the protected resource prior to making the request to the authorization server so that the Provided Token Binding ID for the TLS connection to the protected resource can be obtained. How the client retrieves this Token Binding ID from the underlying Token Binding API is implementation and operating system specific. An alternative, if supported, is for the client to generate a Token Binding key to use for the protected resource, use the Token Binding ID for that key, and then later use that key when the TLS connection to the protected resource is established.

3.1. Access Tokens Issued from the Authorization Endpoint

For access tokens returned directly from the authorization endpoint, such as with the implicit grant defined in OAuth 2.0 [RFC6749], Section 4.2, the Token Binding ID of the client's TLS channel to the protected resource is sent with the authorization request as the Referred Token Binding ID in the "Sec-Token-Binding" header, and is used to Token Bind the access token.

Upon receiving the Referred Token Binding ID in an authorization request, the authorization server associates (Token Binds) the ID with the access token in a way that can be accessed by the protected resource. Such methods include embedding the Referred Token Binding ID (or a cryptographic hash of it) in the issued access token itself,

possibly using the syntax described in Section 3.4, or through token introspection as described in Section 3.5. The method for associating the referred token binding ID with the access token is determined by the authorization server and the protected resource, and is beyond the scope for this specification.

3.1.1. Example Access Token Issued from the Authorization Endpoint

This section provides an example of what the interactions around a Token Bound access token issued from the authorization endpoint might look like, along with some details of the involved processing. Extra line breaks in all examples are for display purposes only.

The client directs the user-agent to make the following HTTP request to the authorization endpoint. It is a typical authorization request that, because Token Binding was negotiated on the underlying TLS connection and the user-agent was signaled to reveal the Referred Token Binding, also includes the "Sec-Token-Binding" header with a Token Binding Message that contains both a Provided and Referred Token Binding. The base64url-encoded EKM from the TLS connection over which the request was made is "jI5UAYjs5XCPISUGQIwgcSrOiVIWq4fhLVIFTQ4nLxc".

```
GET /as/authorization.oauth2?response_type=token
  &client_id=example-client-id&state=rM8pZxG1c3gKy6rEbsD8s
  &redirect_uri=https%3A%2F%2Fclient%2Eexample%2Eorg%2Fcb HTTP/1.1
Host: server.example.com
Sec-Token-Binding: ARIAAGBBQIEE8mSMtDy2dj9EEBdXaQT9W3Rq1NS-jW8ebPoF
6FyL0jIfATVE55zlircgOTZmEglxeIrc3DsGegwjs4bhw14AQGKDLAXFFMyQkZegC
wlbTlqX3F9HTt-1JxFU_pil6ezka7qVRCpSF0BQLfSqlsxMbYfSSCJX1BDtrIL7PX
j__fUAAAECAEFAlBNUnP3te5WrwlEwiejEz0OpesmC5PElWc7kZ5nlLSqQTj1ciIp
5vQ30LLUCyM_a2BYTUPKtd5EdS-PalT4t6ABADgeizRa5NkTMuX4zOdC-R4cLNWVV
08lLu2Psko-UJLR_XAH4Q0H7-m0_nQR1zBN78nYMKPvHsz8L3zWKRvYXEgAA
```

Figure 5: Authorization Request

The authorization server issues an access token and delivers it to the client by redirecting the user-agent with the following HTTP response:

```
HTTP/1.1 302 Found
Location: https://client.example.org/cb#state=rM8pZxG1c3gKy6rEbsD8s
  &expires_in=3600&token_type=Bearer
  &access_token=eyJhbGciOiJIUzI1IiwiaXNjaWkiOiJ1IiwiaWF0IjoiMTYxMjM0MjM0In08xy5W5sQ
```

Figure 6: Authorization Response

The access token is bound to the Referred Token Binding ID from the authorization request, which when represented as a JWT, as described in Section 3.4, contains the SHA-256 hash of the Token Binding ID as the value of the "tbh" (token binding hash) member of the "cnf" (confirmation) claim. The confirmation claim portion of the JWT Claims Set is shown in the following figure.

```
{
  ...other claims omitted for brevity...
  "cnf":{
    "tbh": "vowQESa_MgbGJwIXaFm_BTN2QDPwh8PhuBm-EtUAqxc"
  }
}
```

Figure 7: Confirmation Claim

3.2. Access Tokens Issued from the Token Endpoint

For access tokens returned from the token endpoint, the Token Binding ID of the client's TLS channel to the protected resource is sent as the Referred Token Binding ID in the "Sec-Token-Binding" header, and is used to Token Bind the access token. This applies to all the grant types from OAuth 2.0 [RFC6749] using the token endpoint, including, but not limited to the refresh and authorization code token requests, as well as some extension grants, such as JWT assertion authorization grants [RFC7523].

Upon receiving the Referred Token Binding ID in a token request, the authorization server associates (Token Binds) the ID with the access token in a way that can be accessed by the protected resource. Such methods include embedding the Referred Token Binding ID (or a cryptographic hash of it) in the issued access token itself, possibly using the syntax described in Section 3.4, or through token introspection as described in Section 3.5. The method for associating the referred token binding ID with the access token is determined by the authorization server and the protected resource, and is beyond the scope for this specification.

Note that if the request results in a new refresh token being generated, it can be Token bound using the Provided Token Binding ID, per Section 2.

3.2.1. Example Access Token Issued from the Token Endpoint

This section provides an example of what the interactions around a Token Bound access token issued from the token endpoint might look like, along with some details of the involved processing. Extra line breaks in all examples are for display purposes only.

The client makes an access token request to the token endpoint and includes the "Sec-Token-Binding" header with a Token Binding Message that contains both Provided and Referred Token Binding IDs. The Provided Token Binding ID is used to validate the token binding of the refresh token in the request (and to Token Bind a new refresh token, if one is issued), and the Referred Token Binding ID is used to Token Bind the access token that is generated. The base64url-encoded EKM from the TLS connection over which the access token request was made is "4jTc5elQpocqPTZ5l6jsb6pRP18IFKdwwPvasYjnl-E".

```
POST /as/token.oauth2 HTTP/1.1
Host: server.example.com
Content-Type: application/x-www-form-urlencoded
Sec-Token-Binding: ARIAAGBBQJFXJir2w4gbJ7grBx9uTYWIrs9V50-PW4ZijegQ
  0LUM-_bGnGT6DizxUK-m5n3dQUIkeH7ybn6wb1C5dGyV_IAAQDDFTtoFrHt41Zppq7
  u_SEMF_E-KimAB-HewWl2MvZzAQ9QKoWiJCLFiCkjpgtr1RrA2-jaJvoB8o51DTGXQ
  ydWYkAAAECAEFaUc1G1YU83rqTGHEauloqvNwy0fDsdXzIyT_4x1FcldsMxjFkJac
  IBJFGuYcccvnCak_duFi3QKFENuwxql-H9ABAMcU7IjJOUA4IyE6YoEcfz9BMPQqw
  M5M6hw4RZNQd58fsTCCslQE_NmNc19JXy4NkdkeZBxqvZGPr0y8QZ_bmAwAA

refresh_token=gZR_ZI8EAhLgWR-gWxBimbgZRzi_8EAhLgWRgWxBimbf
&grant_type=refresh_token&client_id=example-client-id
```

Figure 8: Access Token Request

The authorization server issues an access token bound to the Referred Token Binding ID and delivers it in a response the client.

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

{
  "access_token": "eyJhbGciOiJIJFZlIiwiaXNpIjpb...omitted...lcs29j5c3",
  "token_type": "Bearer",
  "expires_in": 3600
}
```

Figure 9: Response

The access token is bound to the Referred Token Binding ID of the access token request, which when represented as a JWT, as described in Section 3.4, contains the SHA-256 hash of the Token Binding ID as the value of the "tbh" (token binding hash) member of the "cnf" (confirmation) claim. The confirmation claim portion of the JWT Claims Set of the access token is shown in the following figure.

```
{
  ...other claims omitted for brevity...
  "cnf":{
    "tbh": "7NRBu9iDdJlYCTOqyeYuLxXv0blEA-yTpmGirAwKAws"
  }
}
```

Figure 10: Confirmation Claim

3.3. Protected Resource Token Binding Validation

Upon receiving a token bound access token, the protected resource validates the binding by comparing the Provided Token Binding ID to the Token Binding ID for the access token. Alternatively, cryptographic hashes of these Token Binding ID values can be compared. If the values do not match, the resource access attempt MUST be rejected with an error.

3.3.1. Example Protected Resource Request

For example, a protected resource request using the access token from Section 3.2.1 would look something like the following. The base64url-encoded EKM from the TLS connection over which the request was made is "7LSNP3BT1aHHdXdk6meEWjtSkiPVLb7YS6iHp-JXmuE". The protected resource validates the binding by comparing the Provided Token Binding ID from the "Sec-Token-Binding" header to the token binding hash confirmation of the access token. Extra line breaks in the example are for display purposes only.

```
GET /api/stuff HTTP/1.1
Host: resource.example.org
Authorization: Bearer eyJhbGciOiJIUzI1NiIsI[...omitted...]cs29j5c3
Sec-Token-Binding: AIkAAgBBQLGtRpWFPN66kxhxGrtaKrzcmThw7HV8yMk_-Mdr
XJXbDMYxZCWnCASRRrmHHHL5wmpP3bhYt0ChRDbsMapfh_QAQN1He3Ftj4Wa_S_fz
ZVns4saLfj6aBoMSQW6rLs19IiVhZe7LrGjKyCfPTKXjaJebxp-TLPFZCc0JTqTY5
0MBAAAA
```

Figure 11: Protected Resource Request

3.4. Representing Token Binding in JWT Access Tokens

If the access token is represented as a JWT, the token binding information SHOULD be represented in the same way that it is in token bound OpenID Connect ID Tokens [OpenID.TokenBinding]. That specification defines the new JWT Confirmation Method RFC 7800 [RFC7800] member "tbh" (token binding hash) to represent the SHA-256 hash of a Token Binding ID in an ID Token. The value of the "tbh" member is the base64url encoding of the SHA-256 hash of the Token

Binding ID. All trailing pad '=' characters are omitted from the encoded value and no line breaks, whitespace, or other additional characters are included.

The following example demonstrates the JWT Claims Set of an access token containing the base64url encoding of the SHA-256 hash of a Token Binding ID as the value of the "tbh" (token binding hash) element in the "cnf" (confirmation) claim:

```
{
  "iss": "https://server.example.com",
  "aud": "https://resource.example.org",
  "sub": "brian@example.com",
  "iat": 1467324320,
  "exp": 1467324920,
  "cnf": {
    "tbh": "7NRBu9iDdJlYCTOgyeYuLxXv0bleA-yTpmGirAwKAws"
  }
}
```

Figure 12: JWT with Token Binding Hash Confirmation Claim

3.5. Representing Token Binding in Introspection Responses

OAuth 2.0 Token Introspection [RFC7662] defines a method for a protected resource to query an authorization server about the active state of an access token as well as to determine meta-information about the token.

For a token bound access token, the hash of the Token Binding ID to which the token is bound is conveyed to the protected resource as meta-information in a token introspection response. The hash is conveyed using same structure as the token binding hash confirmation method, described in Section 3.4, as a top-level member of the introspection response JSON. The protected resource compares that token binding hash to a hash of the provided Token Binding ID and rejects the request, if they do not match.

The following is an example of an introspection response for an active token bound access token with a "tbh" token binding hash confirmation method.


```
HTTP/1.1 200 OK
Content-Type: application/json

{
  "active": true,
  "iss": "https://server.example.com",
  "aud": "https://resource.example.org",
  "sub": "brian@example.com",
  "iat": 1467324320,
  "exp": 1467324920,
  "cnf": {
    "tbh": "7NRBu9iDdJlYCTOqyeYuLxXv0blEA-yTpmGirAwKAws"
  }
}
```

Figure 13: Example Introspection Response for a Token Bound Access Token

4. Token Binding Metadata

4.1. Token Binding Client Metadata

Clients supporting Token Binding that also support the OAuth 2.0 Dynamic Client Registration Protocol [RFC7591] use these metadata values to declare their support for Token Binding of access tokens and refresh tokens:

`client_access_token_token_binding_supported`
OPTIONAL. Boolean value specifying whether the client supports Token Binding of access tokens. If omitted, the default value is "false".

`client_refresh_token_token_binding_supported`
OPTIONAL. Boolean value specifying whether the client supports Token Binding of refresh tokens. If omitted, the default value is "false". Authorization servers MUST NOT Token Bind refresh tokens issued to a client that does not support Token Binding of refresh tokens, but MAY reject requests completely from such clients if token binding is required by authorization server policy by returning an OAuth error response.

4.2. Token Binding Authorization Server Metadata

Authorization servers supporting Token Binding that also support OAuth 2.0 Authorization Server Metadata [RFC8414] use these metadata values to declare their support for Token Binding of access tokens and refresh tokens:

`as_access_token_token_binding_supported`

OPTIONAL. Boolean value specifying whether the authorization server supports Token Binding of access tokens. If omitted, the default value is "false".

`as_refresh_token_token_binding_supported`

OPTIONAL. Boolean value specifying whether the authorization server supports Token Binding of refresh tokens. If omitted, the default value is "false".

5. Token Binding for Authorization Codes

There are two variations for Token Binding of an authorization code. One is appropriate for native application clients and the other for web server clients. The nature of where the various components reside for the different client types demands different methods of Token Binding the authorization code so that it is bound to a Token Binding key on the end user's device. This ensures that a lost or stolen authorization code cannot be successfully utilized from a different device. For native application clients, the code is bound to a Token Binding key pair that the native client itself possesses. For web server clients, the code is bound to a Token Binding key pair on the end user's browser. Both variations utilize the extensible framework of Proof Key for Code Exchange (PKCE) [RFC7636], which enables the client to show possession of a certain key when exchanging the authorization code for tokens. The following subsections individually describe each of the two PKCE methods respectively.

5.1. Native Application Clients

This section describes a PKCE method suitable for native application clients that cryptographically binds the authorization code to a Token Binding key pair on the client, which the client proves possession of on the TLS connection during the access token request containing the authorization code. The authorization code is bound to the Token Binding ID that the native application client uses to resolve the authorization code at the token endpoint. This binding ensures that the client that made the authorization request is the same client that is presenting the authorization code.

5.1.1. Code Challenge

As defined in Proof Key for Code Exchange [RFC7636], the client sends the code challenge as part of the OAuth 2.0 authorization request with the two additional parameters: "code_challenge" and "code_challenge_method".

For this Token Binding method of PKCE, "TB-S256" is used as the value of the "code_challenge_method" parameter.

The value of the "code_challenge" parameter is the base64url encoding (per Section 5 of [RFC4648] with all trailing padding ('=')) characters omitted and without the inclusion of any line breaks or whitespace) of the SHA-256 hash of the Provided Token Binding ID that the client will use when calling the authorization server's token endpoint. Note that, prior to making the authorization request, the client may need to establish a TLS connection between itself and the authorization server's token endpoint in order to establish the appropriate Token Binding ID.

When the authorization server issues the authorization code in the authorization response, it associates the code challenge and method values with the authorization code so they can be verified later when the authorization code is presented in the access token request.

5.1.1.1. Example Code Challenge

For example, a native application client sends an authorization request by sending the user's browser to the authorization endpoint. The resulting HTTP request looks something like the following (with extra line breaks for display purposes only).

```
GET /as/authorization.oauth2?response_type=code
    &client_id=example-native-client-id&state=oUC2jyYtzRCrMyWrVnGj
    &code_challenge=rBlgOyMY4teiuJMDgOwkrpsAjPyI07D2WseM-dnq6eE
    &code_challenge_method=TB-S256 HTTP/1.1
Host: server.example.com
```

Figure 14: Authorization Request with PKCE Challenge

5.1.1.2. Code Verifier

Upon receipt of the authorization code, the client sends the access token request to the token endpoint. The Token Binding Protocol [RFC8471] is negotiated on the TLS connection between the client and the authorization server and the "Sec-Token-Binding" header, as defined in Token Binding over HTTP [RFC8473], is included in the access token request. The authorization server extracts the Provided Token Binding ID from the header value, hashes it with SHA-256, and compares it to the "code_challenge" value previously associated with the authorization code. If the values match, the token endpoint continues processing as normal (as defined by OAuth 2.0 [RFC6749]). If the values do not match, an error response indicating "invalid_grant" MUST be returned.

The "Sec-Token-Binding" header contains sufficient information for verification of the authorization code and its association to the original authorization request. However, PKCE [RFC7636] requires that a "code_verifier" parameter be sent with the access token request, so the static value "provided_tb" is used to meet that requirement and indicate that the Provided Token Binding ID is used for the verification.

5.1.2.1. Example Code Verifier

An example access token request, correlating to the authorization request in the previous example, to the token endpoint over a TLS connection for which Token Binding has been negotiated would look like the following (with extra line breaks for display purposes only). The base64url-encoded EKM from the TLS connection over which the request was made is

```
"pNVKtPuQFvylNYn000QowWrQKoeMkeX9H32hVuU71Bs".
```

```
POST /as/token.oauth2 HTTP/1.1
Host: server.example.com
Content-Type: application/x-www-form-urlencoded
Sec-Token-Binding: AIkAAgBBQEO09GRFP-LM0hoWw6-2i318BsuuUum5AL8bt1sz
  lr1EFfp5DMXMNW3O8WjcIXr2DKJnI4xnuGse6GywQd9RbD0AQJDb3xyo9PBxj8M6Y
  jLt-6OaxgDkyoBoTkrynNbLc8tJQ0JtXomKzBbj5qPtHDduXc6xz_lzvNpxSPxi42
  8m7wkAAA
```

```
grant_type=authorization_code&code=mJAReTWKX7zI3oHUNd4o3PeNqNqxKGp6
  &code_verifier=provided_tb&client_id=example-native-client-id
```

Figure 15: Token Request with PKCE Verifier

5.2. Web Server Clients

This section describes a PKCE method suitable for web server clients, which cryptographically binds the authorization code to a Token Binding key pair on the browser. The authorization code is bound to the Token Binding ID that the browser uses to deliver the authorization code to a web server client, which is sent to the authorization server as the Referred Token Binding ID during the authorization request. The web server client conveys the Token Binding ID to the authorization server when making the access token request containing the authorization code. This binding ensures that the authorization code cannot successfully be played or replayed to the web server client from a different browser than the one that made the authorization request.

5.2.1. Code Challenge

As defined in Proof Key for Code Exchange [RFC7636], the client sends the code challenge as part of the OAuth 2.0 Authorization Request with the two additional parameters: "code_challenge" and "code_challenge_method".

The client must send the authorization request through the browser such that the Token Binding ID established between the browser and itself is revealed to the authorization server's authorization endpoint as the Referred Token Binding ID. Typically, this is done with an HTTP redirection response and the "Include-Referred-Token-Binding-ID" header, as defined in Token Binding over HTTP [RFC8473], Section 5.3.

For this Token Binding method of PKCE, "referred_tb" is used for the value of the "code_challenge_method" parameter.

The value of the "code_challenge" parameter is "referred_tb". The static value for the required PKCE parameter indicates that the authorization code is to be bound to the Referred Token Binding ID from the Token Binding Message sent in the "Sec-Token-Binding" header of the authorization request.

When the authorization server issues the authorization code in the authorization response, it associates the Token Binding ID (or hash thereof) and code challenge method with the authorization code so they can be verified later when the authorization code is presented in the access token request.

5.2.1.1. Example Code Challenge

For example, the web server client sends the authorization request by redirecting the browser to the authorization endpoint. That HTTP redirection response looks like the following (with extra line breaks for display purposes only).

```
HTTP/1.1 302 Found
Location: https://server.example.com?response_type=code
        &client_id=example-web-client-id&state=P4FUFqYzslj3ffsYCP34d3
        &redirect_uri=https%3A%2F%2Fclient%2Eexample%2Eorg%2Fcb
        &code_challenge=referred_tb&code_challenge_method=referred_tb
Include-Referred-Token-Binding-ID: true
```

Figure 16: Redirect the Browser

The redirect includes the "Include-Referred-Token-Binding-ID" response header field that signals to the user-agent that it should

reveal, to the authorization server, the Token Binding ID used on the connection to the web server client. The resulting HTTP request to the authorization server looks something like the following (with extra line breaks for display purposes only). The base64url-encoded EKM from the TLS connection over which the request was made is "7gOdRzMhPeO-1YwZGmnVHyReN5vd2CxcSRBN69Ue4cI".

```
GET /as/authorization.oauth2?response_type=code
  &client_id=example-web-client-id&state=dryo8YFpWacBUPjhBf4Nvt51
  &redirect_uri=https%3A%2F%2Fclient%2Eexample%2Eorg%2Fcb
  &code_challenge=referred_tb
  &code_challenge_method=referred_tb HTTP/1.1
Host: server.example.com
Sec-Token-Binding: ARIAAGBBQB-XOPf5ePlf7ikATiAFEGOS503lPmRfkyymzdWw
  HCxl0njxC3D0E_OVfBNqrIQxzIfkF7tWby2ZfyaE6XpwTsAQBYqhFX78vM0gDX_F
  d_b2dlHyHlMmkIz8iMVBY_reM98OUaJFz5IB7PG9nZ11j58LoG5QhmQoI9NXYktKZ
  RXxrYAAAECAEFAdUFTnfQADknluDbQnvJEk6oQs38L92gv-KO-qlYadLoDIKe2h53
  hSiKwIP98iRj_unedkNkAMyg9e2mY4Gp7WwBAeDUOwaSXNz1e6gKohwN4SAZ5eNyx
  45Mh8VI4woLlBipLoqrJRok6dxFkWGHRMuBROcLGUj5PiOoxybQH_Tom3gAA
```

Figure 17: Authorization Request

5.2.2. Code Verifier

The web server client receives the authorization code from the browser and extracts the Provided Token Binding ID from the "Sec-Token-Binding" header of the request. The client sends the base64url-encoded (per Section 5 of [RFC4648] with all trailing padding ('=') characters omitted and without the inclusion of any line breaks or whitespace) Provided Token Binding ID as the value of the "code_verifier" parameter in the access token request to the authorization server's token endpoint. The authorization server compares the value of the "code_verifier" parameter to the Token Binding ID value previously associated with the authorization code. If the values match, the token endpoint continues processing as normal (as defined by OAuth 2.0 [RFC6749]). If the values do not match, an error response indicating "invalid_grant" MUST be returned.

5.2.2.1. Example Code Verifier

Continuing the example from the previous section, the authorization server sends the code to the web server client by redirecting the browser to the client's "redirect_uri", which results in the browser making a request like the following (with extra line breaks for display purposes only) to the web server client over a TLS channel for which Token Binding has been established. The base64url-encoded EKM from the TLS connection over which the request was made is "EzW60vyINbsb_tajt8ij3tV6cwY2KH-i8BdEMYXcNn0".

```
GET /cb?state=dryo8YFpWacbUPjhBf4Nvt5l&code=jwD3oOa5cQvvLc81bwc4CMw
Host: client.example.org
Sec-Token-Binding: AIAAgBBQHVBu530AA5J9bg20J7yRJOqELN_C_doL_ijvqpW
  GnS6AyCnloed4UoisCD_fIkY_7p3nZDZADMoPXtpmOBqelsAQEwgC9Zpg7QFCDBib
  6GlZki3MhH32KNfLefLJclvRlxE8l7OMfPLZHP2Woxh6rEtmgBcAABubEbTz7muNl
  Ln8uoAAA
```

Figure 18: Authorization Response to Web Server Client

The web server client takes the Provided Token Binding ID from the above request from the browser and sends it, base64url encoded, to the authorization server in the "code_verifier" parameter of the authorization code grant type request. Extra line breaks in the example request are for display purposes only.

```
POST /as/token.oauth2 HTTP/1.1
Host: server.example.com
Content-Type: application/x-www-form-urlencoded
Authorization: Basic b3JnLmV4YWlwbGUuY2xpZW50OmlldGY5OGNoaWNhZ28=

grant_type=authorization_code&code=jwD3oOa5cQvvLc81bwc4CMw
&redirect_uri=https%3A%2F%2Fclient%2Eexample%2Eorg%2Fcb
&client_id=example-web-client-id
&code_verifier=AgBBQHVBu530AA5J9bg20J7yRJOqELN_C_doL_ijv
qpWGnS6AyCnloed4UoisCD_fIkY_7p3nZDZADMoPXtpmOBqels
```

Figure 19: Exchange Authorization Code

6. Token Binding JWT Authorization Grants and Client Authentication

The JWT Profile for OAuth 2.0 Client Authentication and Authorization Grants [RFC7523] defines the use of bearer JWTs as a means for requesting an OAuth 2.0 access token as well as for client authentication. This section describes extensions to that specification enabling the application of Token Binding to JWT client authentication and JWT authorization grants.

6.1. JWT Format and Processing Requirements

In addition the requirements set forth in Section 3 of RFC 7523 [RFC7523], the following criteria must also be met for token bound JWTs used as authorization grants or for client authentication.

- o The JWT MUST contain a "cnf" (confirmation) claim with a "tbh" (token binding hash) member identifying the Token Binding ID of the Provided Token Binding used by the client on the TLS connection to the authorization server. The authorization server MUST reject any JWT that has a token binding hash confirmation

that does not match the corresponding hash of the Provided Token Binding ID from the "Sec-Token-Binding" header of the request.

6.2. Token Bound JWTs for Client Authentication

To use a token bound JWT for client authentication, the client uses the parameter values and encodings from Section 2.2 of RFC 7523 [RFC7523] with one exception: the value of the "client_assertion_type" is "urn:ietf:params:oauth:client-assertion-type:jwt-token-bound".

The "OAuth Token Endpoint Authentication Methods" registry [IANA.OAuth.Parameters] contains values, each of which specify a method of authenticating a client to the authorization server. The values are used to indicate supported and utilized client authentication methods in authorization server metadata, such as [OpenID.Discovery] and [RFC8414], and in OAuth 2.0 Dynamic Client Registration Protocol [RFC7591]. The values "private_key_jwt" and "client_secret_jwt" are designated by OpenID Connect [OpenID.Core] as authentication method values for bearer JWT client authentication using asymmetric and symmetric JWS [RFC7515] algorithms respectively. For Token Bound JWT for client authentication, this specification defines and registers the following authentication method values.

private_key_token_bound_jwt

Indicates that client authentication to the authorization server will occur with a Token Bound JWT, which is signed with a client's private key.

client_secret_token_bound_jwt

Indicates that client authentication to the authorization server will occur with a Token Bound JWT, which is integrity protected with a MAC using the octets of the UTF-8 representation of the client secret as the shared key.

Note that just as with the "private_key_jwt" and "client_secret_jwt" authentication methods, the "token_endpoint_auth_signing_alg" client registration parameter may be used to indicate the JWS algorithm used for signing the client authentication JWT for the authentication methods defined above.

6.3. Token Bound JWTs for as Authorization Grants

To use a token bound JWT for an authorization grant, the client uses the parameter values and encodings from Section 2.1 of RFC 7523 [RFC7523] with one exception: the value of the "grant_type" is "urn:ietf:params:oauth:grant-type:jwt-token-bound".

7. Security Considerations

7.1. Phasing in Token Binding

Many OAuth implementations will be deployed in situations in which not all participants support Token Binding. Any combination of the client, the authorization server, the protected resource, and the user agent may not yet support Token Binding, in which case it will not work end-to-end.

It is a context-dependent deployment choice whether to allow interactions to proceed in which Token Binding is not supported or whether to treat the omission of Token Binding at any step as a fatal error. Particularly in dynamic deployment environments in which End Users have choices of clients, authorization servers, protected resources, and/or user agents, it is recommended that, for some reasonable period of time during which Token Binding technology is being adopted, authorizations using one or more components that do not implement Token Binding be allowed to successfully proceed. This enables different components to be upgraded to supporting Token Binding at different times, providing a smooth transition path for phasing in Token Binding. However, when Token Binding has been performed, any Token Binding key mismatches **MUST** be treated as fatal errors.

In more controlled deployment environments where the participants in an authorization interaction are known or expected to support Token Binding and yet one or more of them does not use it, the authorization **SHOULD** be aborted with an error. For instance, an authorization server should reject a token request that does not include the "Sec-Token-Binding" header, if the request is from a client known to support Token Binding (via configuration or the "client_access_token_token_binding_supported" metadata parameter).

7.2. Binding of Refresh Tokens

Section 6 of RFC 6749 [RFC6749] requires that a refresh token be bound to the client to which it was issued and that, if the client type is confidential or the client was issued client credentials (or assigned other authentication requirements), the client must authenticate with the authorization server when presenting the refresh token. As a result, for non-public clients, refresh tokens are indirectly bound to the client's credentials and cannot be used without the associated client authentication. Non-public clients then are afforded protections (equivalent to the strength of their authentication credentials) against unauthorized replay of refresh tokens and it is reasonable to not Token Bind refresh tokens for such clients while still Token Binding the issued access tokens. Refresh

tokens issued to public clients, however, do not have the benefit of such protections and authorization servers MAY elect to disallow public clients from registering or establishing configuration that would allow Token Bound access tokens but unbound refresh tokens.

Some web-based confidential clients implemented as distributed nodes may be perfectly capable of implementing access token binding (if the access token remains on the node it was bound to, the token binding keys would be locally available for that node to prove possession), but may struggle with refresh token binding due to an inability to share token binding key material between nodes. As confidential clients already have credentials which are required to use the refresh token, and those credentials should only ever be sent over TLS server-to-server between the client and the Token Endpoint, there is still value in token binding access tokens without token binding refresh tokens. Authorization servers SHOULD consider supporting access token binding without refresh token binding for confidential web clients as there are still security benefits to do so.

Clients MUST declare through dynamic (Section 4.1) or static registration information what types of token bound tokens they support to enable the server to bind tokens accordingly, taking into account any phase-in policies. Authorization servers MAY reject requests from any client who does not support token binding (by returning an OAuth error response) per their own security policies.

8. IANA Considerations

8.1. OAuth Dynamic Client Registration Metadata Registration

This specification registers the following client metadata definitions in the IANA "OAuth Dynamic Client Registration Metadata" registry [IANA.OAuth.Parameters] established by [RFC7591]:

8.1.1. Registry Contents

- o Client Metadata Name:
"client_access_token_token_binding_supported"
- o Client Metadata Description: Boolean value specifying whether the client supports Token Binding of access tokens
- o Change Controller: IESG
- o Specification Document(s): Section 4.1 of [[this specification]]

- o Client Metadata Name:
"client_refresh_token_token_binding_supported"
- o Client Metadata Description: Boolean value specifying whether the client supports Token Binding of refresh tokens
- o Change Controller: IESG

- o Specification Document(s): Section 4.1 of [[this specification]]

8.2. OAuth Authorization Server Metadata Registration

This specification registers the following metadata definitions in the IANA "OAuth Authorization Server Metadata" registry [IANA.OAuth.Parameters] established by [RFC8414]:

8.2.1. Registry Contents

- o Metadata Name: "as_access_token_token_binding_supported"
- o Metadata Description: Boolean value specifying whether the authorization server supports Token Binding of access tokens
- o Change Controller: IESG
- o Specification Document(s): Section 4.2 of [[this specification]]
- o Metadata Name: "as_refresh_token_token_binding_supported"
- o Metadata Description: Boolean value specifying whether the authorization server supports Token Binding of refresh tokens
- o Change Controller: IESG
- o Specification Document(s): Section 4.2 of [[this specification]]

8.3. PKCE Code Challenge Method Registration

This specification requests registration of the following Code Challenge Method Parameter Names in the IANA "PKCE Code Challenge Methods" registry [IANA.OAuth.Parameters] established by [RFC7636].

8.3.1. Registry Contents

- o Code Challenge Method Parameter Name: TB-S256
- o Change controller: IESG
- o Specification document(s): Section 5.1.1 of [[this specification]]
- o Code Challenge Method Parameter Name: referred_tb
- o Change controller: IESG
- o Specification document(s): Section 5.2.1 of [[this specification]]

9. Token Endpoint Authentication Method Registration

This specification requests registration of the following values in the IANA "OAuth Token Endpoint Authentication Methods" registry [IANA.OAuth.Parameters] established by [RFC7591].

9.1. Registry Contents

- o Token Endpoint Authentication Method Name:
"client_secret_token_bound_jwt"
- o Change Controller: IESG
- o Specification Document(s): Section 6 of [[this specification]]

- o Token Endpoint Authentication Method Name:
"private_key_token_bound_jwt"
- o Change Controller: IESG
- o Specification Document(s): Section 6 of [[this specification]]

10. Sub-Namespace Registrations

This specification requests registration of the following values in the IANA "OAuth URI" registry [IANA.OAuth.Parameters] established in An IETF URN Sub-Namespace for OAuth [RFC6755].

10.1. Registry Contents

- o URN: urn:ietf:params:oauth:grant-type:jwt-token-bound
- o Common Name: Token Bound JWT Grant Type for OAuth 2.0
- o Change controller: IESG
- o Specification Document: Section 6 of [[this specification]]

- o URN: urn:ietf:params:oauth:client-assertion-type:jwt-token-bound
- o Common Name: Token Bound JWT for OAuth 2.0 Client Authentication
- o Change controller: IESG
- o Specification Document: Section 6 of [[this specification]]

11. References

11.1. Normative References

- [IANA.OAuth.Parameters]
IANA, "OAuth Parameters",
<<http://www.iana.org/assignments/oauth-parameters>>.
- [JWT] Jones, M., Bradley, J., and N. Sakimura, "JSON Web Token (JWT)", RFC 7519, DOI 10.17487/RFC7519, May 2015,
<<http://tools.ietf.org/html/rfc7519>>.
- [OpenID.TokenBinding]
Jones, M., Bradley, J., and B. Campbell, "OpenID Connect Token Bound Authentication 1.0", October 2017,
<http://openid.net/specs/openid-connect-token-bound-authentication-1_0-03.html>.

- [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>>.
- [RFC4648] Josefsson, S., "The Base16, Base32, and Base64 Data Encodings", RFC 4648, DOI 10.17487/RFC4648, October 2006, <<https://www.rfc-editor.org/info/rfc4648>>.
- [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>>.
- [RFC7230] Fielding, R., Ed. and J. Reschke, Ed., "Hypertext Transfer Protocol (HTTP/1.1): Message Syntax and Routing", RFC 7230, DOI 10.17487/RFC7230, June 2014, <<https://www.rfc-editor.org/info/rfc7230>>.
- [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/info/rfc7523>>.
- [RFC7591] Richer, J., Ed., Jones, M., Bradley, J., Machulak, M., and P. Hunt, "OAuth 2.0 Dynamic Client Registration Protocol", RFC 7591, DOI 10.17487/RFC7591, July 2015, <<https://www.rfc-editor.org/info/rfc7591>>.
- [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/info/rfc7636>>.
- [RFC7662] Richer, J., Ed., "OAuth 2.0 Token Introspection", RFC 7662, DOI 10.17487/RFC7662, October 2015, <<https://www.rfc-editor.org/info/rfc7662>>.
- [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>>.

- [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/info/rfc8414>>.
- [RFC8471] Popov, A., Ed., Nystroem, M., Balfanz, D., and J. Hodges, "The Token Binding Protocol Version 1.0", RFC 8471, DOI 10.17487/RFC8471, October 2018, <<https://www.rfc-editor.org/info/rfc8471>>.
- [RFC8472] Popov, A., Ed., Nystroem, M., and D. Balfanz, "Transport Layer Security (TLS) Extension for Token Binding Protocol Negotiation", RFC 8472, DOI 10.17487/RFC8472, October 2018, <<https://www.rfc-editor.org/info/rfc8472>>.
- [RFC8473] Popov, A., Nystroem, M., Balfanz, D., Ed., Harper, N., and J. Hodges, "Token Binding over HTTP", RFC 8473, DOI 10.17487/RFC8473, October 2018, <<https://www.rfc-editor.org/info/rfc8473>>.
- [SHS] National Institute of Standards and Technology, "Secure Hash Standard (SHS)", FIPS PUB 180-4, March 2012, <<http://csrc.nist.gov/publications/fips/fips180-4/fips-180-4.pdf>>.

11.2. Informative References

- [BCP212] Denniss, W. and J. Bradley, "OAuth 2.0 for Native Apps", BCP 212, RFC 8252, DOI 10.17487/RFC8252, October 2017, <<https://www.rfc-editor.org/info/rfc8252>>.
- [OpenID.Core] Sakimura, N., Bradley, J., Jones, M., de Medeiros, B., and C. Mortimore, "OpenID Connect Core 1.0", August 2015, <http://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", August 2015, <http://openid.net/specs/openid-connect-discovery-1_0.html>.
- [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>>.

[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>>.

Appendix A. Acknowledgements

This specification was developed within the OAuth Working Group under the chairmanship of Hannes Tschofenig and Rifaat Shekh-Yusef with Kathleen Moriarty, Eric Rescorla, and Benjamin Kaduk serving as Security Area Directors. Additionally, the following individuals contributed ideas, feedback, and wording that helped shape this specification: Dirk Balfanz, Andrei Popov, Justin Richer, and Nat Sakimura.

Appendix B. Document History

[[to be removed by the RFC Editor before publication as an RFC]]

-08

- o Update reference to -03 of openid-connect-token-bound-authentication.
- o Update the references to the core token binding specs, which are now RFCs 8471, 8472, and 8473.
- o Update reference to AS metadata, which is now RFC 8414.
- o Add chairs and ADs to the Acknowledgements.

-07

- o Explicitly state that the base64url encoding of the tbh value doesn't include any trailing pad characters, line breaks, whitespace, etc.
- o Update to latest references for tokbind drafts and draft-ietf-oauth-discovery.
- o Update reference to Implementation Considerations in draft-ietf-tokbind-https, which is section 6 rather than 5.
- o Try to tweak text that references specific sections in other documents so that the HTML generated by the ietf tools doesn't link to the current document (based on old suggestion from Barry <https://www.ietf.org/mail-archive/web/jose/current/msg04571.html>).

-06

- o Use the boilerplate from RFC 8174.
- o Update reference for draft-ietf-tokbind-https to -12 and draft-ietf-oauth-discovery to -09.
- o Minor editorial fixes.

-05

- o State that authorization servers should not token bind refresh tokens issued to a client that doesn't support bound refresh tokens, which can be indicated by the "client_refresh_token_token_binding_supported" client metadata parameter.
- o Add Token Binding for JWT Authorization Grants and JWT Client Authentication.
- o Adjust the language around aborting authorizations in Phasing in Token Binding to be somewhat more general and not only about downgrades.
- o Remove reference to, and usage of, 'OAuth 2.0 Protected Resource Metadata', which is no longer a going concern.
- o Moved "Token Binding Metadata" section before "Token Binding for Authorization Codes" to be closer to the "Token Binding for Access Tokens" and "Token Binding for Refresh Tokens", to which it is more closely related.
- o Update references for draft-ietf-tokbind- negotiation(-10), protocol(-16), and https(-10), as well as draft-ietf-oauth-discovery(-07), and BCP212/RFC8252 OAuth 2.0 for Native Apps.

-04

- o Define how to convey token binding information of an access token via RFC 7662 OAuth 2.0 Token Introspection (note that the Introspection Response Registration request for cnf/Confirmation is in <https://tools.ietf.org/html/draft-ietf-oauth-mtls-02#section-4.3> which will likely be published and registered prior to this document).
- o Minor editorial fixes.
- o Added an open issue about needing to allow for web server clients to opt-out of having refresh tokens bound while still allowing for binding of access tokens (following from mention of the problem on

slide 16 of the presentation from Chicago
<https://www.ietf.org/proceedings/98/slides/slides-98-oauth-sessb-token-binding-00.pdf>).

-03

- o Fix a few mistakes in and around the examples that were noticed preparing the slides for IETF 98 Chicago.

-02

- o Added a section on Token Binding for authorization codes with one variation for native clients and one for web server clients.
- o Updated language to reflect that the binding is to the token binding key pair and that proof-of-possession of that key is done on the TLS connection.
- o Added a bunch of examples.
- o Added a few Open Issues so they are tracked in the document.
- o Updated the Token Binding and OAuth Metadata references.
- o Added William Denniss as an author.

-01

- o Changed Token Binding for access tokens to use the Referred Token Binding ID, now that the Implementation Considerations in the Token Binding HTTPS specification make it clear that implementations will enable using the Referred Token Binding ID.
- o Defined Protected Resource Metadata value.
- o Changed to use the more specific term "protected resource" instead of "resource server".

-00

- o Created the initial working group version from draft-jones-oauth-token-binding-00.

Authors' Addresses

Michael B. Jones
Microsoft

Email: mbj@microsoft.com
URI: <http://self-issued.info/>

Brian Campbell
Ping Identity

Email: brian.d.campbell@gmail.com

John Bradley
Yubico

Email: ve7jtb@ve7jtb.com
URI: <http://www.thread-safe.com/>

William Denniss
Google
1600 Amphitheatre Pkwy
Mountain View, CA 94043
USA

Email: wdenniss@google.com
URI: <http://wdenniss.com/>

OAuth Working Group
Internet-Draft
Intended status: Standards Track
Expires: January 21, 2020

M. Jones
A. Nadalin
Microsoft
B. Campbell, Ed.
Ping Identity
J. Bradley
Yubico
C. Mortimore
Salesforce
July 20, 2019

OAuth 2.0 Token Exchange
draft-ietf-oauth-token-exchange-19

Abstract

This specification defines a protocol for an HTTP- and JSON- based Security Token Service (STS) by defining how to request and obtain security tokens from OAuth 2.0 authorization servers, including security tokens employing impersonation and delegation.

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 January 21, 2020.

Copyright Notice

Copyright (c) 2019 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<https://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents

carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1. Introduction	3
1.1. Delegation vs. Impersonation Semantics	4
1.2. Requirements Notation and Conventions	6
1.3. Terminology	6
2. Token Exchange Request and Response	6
2.1. Request	6
2.1.1. Relationship Between Resource, Audience and Scope . .	9
2.2. Response	9
2.2.1. Successful Response	9
2.2.2. Error Response	11
2.3. Example Token Exchange	11
3. Token Type Identifiers	13
4. JSON Web Token Claims and Introspection Response Parameters .	15
4.1. "act" (Actor) Claim	15
4.2. "scope" (Scopes) Claim	17
4.3. "client_id" (Client Identifier) Claim	18
4.4. "may_act" (Authorized Actor) Claim	18
5. Security Considerations	19
6. Privacy Considerations	20
7. IANA Considerations	20
7.1. OAuth URI Registration	20
7.1.1. Registry Contents	20
7.2. OAuth Parameters Registration	21
7.2.1. Registry Contents	21
7.3. OAuth Access Token Type Registration	22
7.3.1. Registry Contents	22
7.4. JSON Web Token Claims Registration	22
7.4.1. Registry Contents	22
7.5. OAuth Token Introspection Response Registration	23
7.5.1. Registry Contents	23
7.6. OAuth Extensions Error Registration	23
7.6.1. Registry Contents	23
8. References	24
8.1. Normative References	24
8.2. Informative References	24
Appendix A. Additional Token Exchange Examples	26
A.1. Impersonation Token Exchange Example	26
A.1.1. Token Exchange Request	26
A.1.2. Subject Token Claims	26
A.1.3. Token Exchange Response	27

A.1.4. Issued Token Claims	27
A.2. Delegation Token Exchange Example	28
A.2.1. Token Exchange Request	28
A.2.2. Subject Token Claims	29
A.2.3. Actor Token Claims	29
A.2.4. Token Exchange Response	29
A.2.5. Issued Token Claims	30
Appendix B. Acknowledgements	31
Appendix C. Document History	31
Authors' Addresses	35

1. Introduction

A security token is a set of information that facilitates the sharing of identity and security information in heterogeneous environments or across security domains. Examples of security tokens include JSON Web Tokens (JWTs) [JWT] and SAML 2.0 Assertions [OASIS.saml-core-2.0-os]. Security tokens are typically signed to achieve integrity and sometimes also encrypted to achieve confidentiality. Security tokens are also sometimes described as Assertions, such as in [RFC7521].

A Security Token Service (STS) is a service capable of validating security tokens provided to it and issuing new security tokens in response, which enables clients to obtain appropriate access credentials for resources in heterogeneous environments or across security domains. Web Service clients have used WS-Trust [WS-Trust] as the protocol to interact with an STS for token exchange. While WS-Trust uses XML and SOAP, the trend in modern Web development has been towards RESTful patterns and JSON. The OAuth 2.0 Authorization Framework [RFC6749] and OAuth 2.0 Bearer Tokens [RFC6750] have emerged as popular standards for authorizing third-party applications' access to HTTP and RESTful resources. The conventional OAuth 2.0 interaction involves the exchange of some representation of resource owner authorization for an access token, which has proven to be an extremely useful pattern in practice. However, its input and output are somewhat too constrained as is to fully accommodate a security token exchange framework.

This specification defines a protocol extending OAuth 2.0 that enables clients to request and obtain security tokens from authorization servers acting in the role of an STS. Similar to OAuth 2.0, this specification focuses on client developer simplicity and requires only an HTTP client and JSON parser, which are nearly universally available in modern development environments. The STS protocol defined in this specification is not itself RESTful (an STS doesn't lend itself particularly well to a REST approach) but does

utilize communication patterns and data formats that should be familiar to developers accustomed to working with RESTful systems.

A new grant type for a token exchange request and the associated specific parameters for such a request to the token endpoint are defined by this specification. A token exchange response is a normal OAuth 2.0 response from the token endpoint with a few additional parameters defined herein to provide information to the client.

The entity that makes the request to exchange tokens is considered the client in the context of the token exchange interaction. However, that does not restrict usage of this profile to traditional OAuth clients. An OAuth resource server, for example, might assume the role of the client during token exchange in order to trade an access token that it received in a protected resource request for a new token that is appropriate to include in a call to a backend service. The new token might be an access token that is more narrowly scoped for the downstream service or it could be an entirely different kind of token.

The scope of this specification is limited to the definition of a basic request-and-response protocol for an STS-style token exchange utilizing OAuth 2.0. Although a few new JWT claims are defined that enable delegation semantics to be expressed, the specific syntax, semantics and security characteristics of the tokens themselves (both those presented to the authorization server and those obtained by the client) are explicitly out of scope and no requirements are placed on the trust model in which an implementation might be deployed. Additional profiles may provide more detailed requirements around the specific nature of the parties and trust involved, such as whether signing and/or encryption of tokens is needed or if proof-of-possession style tokens will be required or issued; however, such details will often be policy decisions made with respect to the specific needs of individual deployments and will be configured or implemented accordingly.

The security tokens obtained may be used in a number of contexts, the specifics of which are also beyond the scope of this specification.

1.1. Delegation vs. Impersonation Semantics

One common use case for an STS (as alluded to in the previous section) is to allow a resource server A to make calls to a backend service C on behalf of the requesting user B. Depending on the local site policy and authorization infrastructure, it may be desirable for A to use its own credentials to access C along with an annotation of some form that A is acting on behalf of B ("delegation"), or for A to be granted a limited access credential to C but that continues to

identify B as the authorized entity ("impersonation"). Delegation and impersonation can be useful concepts in other scenarios involving multiple participants as well.

When principal A impersonates principal B, A is given all the rights that B has within some defined rights context and is indistinguishable from B in that context. Thus, when principal A impersonates principal B, then insofar as any entity receiving such a token is concerned, they are actually dealing with B. It is true that some members of the identity system might have awareness that impersonation is going on, but it is not a requirement. For all intents and purposes, when A is impersonating B, A is B within the context of the rights authorized by the token. A's ability to impersonate B could be limited in scope or time, or even with a one-time-use restriction, whether via the contents of the token or an out-of-band mechanism.

Delegation semantics are different than impersonation semantics, though the two are closely related. With delegation semantics, principal A still has its own identity separate from B and it is explicitly understood that while B may have delegated some of its rights to A, any actions taken are being taken by A representing B. In a sense, A is an agent for B.

Delegation and impersonation are not inclusive of all situations. When a principal is acting directly on its own behalf, for example, neither delegation nor impersonation are in play. They are, however, the more common semantics operating for token exchange and, as such, are given more direct treatment in this specification.

Delegation semantics are typically expressed in a token by including information about both the primary subject of the token as well as the actor to whom that subject has delegated some of its rights. Such a token is sometimes referred to as a composite token because it is composed of information about multiple subjects. Typically, in the request, the "subject_token" represents the identity of the party on behalf of whom the token is being requested while the "actor_token" represents the identity of the party to whom the access rights of the issued token are being delegated. A composite token issued by the authorization server will contain information about both parties. When and if a composite token is issued is at the discretion of the authorization server and applicable policy and configuration.

The specifics of representing a composite token and even whether or not such a token will be issued depend on the details of the implementation and the kind of token. The representations of composite tokens that are not JWTs are beyond the scope of this

specification. The "actor_token" request parameter, however, does provide a means for providing information about the desired actor and the JWT "act" claim can provide a representation of a chain of delegation.

1.2. Requirements Notation and Conventions

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.

1.3. Terminology

This specification uses the terms "access token type", "authorization server", "client", "client identifier", "resource server", "token endpoint", "token request", and "token response" defined by OAuth 2.0 [RFC6749], and the terms "Base64url Encoding", "Claim", and "JWT Claims Set" defined by JSON Web Token (JWT) [JWT].

2. Token Exchange Request and Response

2.1. Request

A client requests a security token by making a token request to the authorization server's token endpoint using the extension grant type mechanism defined in Section 4.5 of [RFC6749].

Client authentication to the authorization server is done using the normal mechanisms provided by OAuth 2.0. Section 2.3.1 of [RFC6749] defines password-based authentication of the client, however, client authentication is extensible and other mechanisms are possible. For example, [RFC7523] defines client authentication using bearer JSON Web Tokens (JWTs) [JWT]. The supported methods of client authentication and whether or not to allow unauthenticated or unidentified clients are deployment decisions that are at the discretion of the authorization server. Note that omitting client authentication allows for a compromised token to be leveraged via an STS into other tokens by anyone possessing the compromised token. Thus client authentication allows for additional authorization checks by the STS as to which entities are permitted to impersonate or receive delegations from other entities.

The client makes a token exchange request to the token endpoint with an extension grant type using the HTTP "POST" method. The following parameters are included in the HTTP request entity-body using the

"application/x-www-form-urlencoded" format with a character encoding of UTF-8 as described in Appendix B of RFC6749 [RFC6749].

grant_type

REQUIRED. The value "urn:ietf:params:oauth:grant-type:token-exchange" indicates that a token exchange is being performed.

resource

OPTIONAL. A URI that indicates the target service or resource where the client intends to use the requested security token. This enables the authorization server to apply policy as appropriate for the target, such as determining the type and content of the token to be issued or if and how the token is to be encrypted. In many cases, a client will not have knowledge of the logical organization of the systems with which it interacts and will only know a URI of the service where it intends to use the token. The "resource" parameter allows the client to indicate to the authorization server where it intends to use the issued token by providing the location, typically as an https URL, in the token exchange request in the same form that will be used to access that resource. The authorization server will typically have the capability to map from a resource URI value to an appropriate policy. The value of the "resource" parameter MUST be an absolute URI, as specified by Section 4.3 of [RFC3986], which MAY include a query component and MUST NOT include a fragment component. Multiple "resource" parameters may be used to indicate that the issued token is intended to be used at the multiple resources listed. See [I-D.ietf-oauth-resource-indicators] for additional background and uses of the "resource" parameter.

audience

OPTIONAL. The logical name of the target service where the client intends to use the requested security token. This serves a purpose similar to the "resource" parameter, but with the client providing a logical name for the target service. Interpretation of the name requires that the value be something that both the client and the authorization server understand. An OAuth client identifier, a SAML entity identifier [OASIS.saml-core-2.0-os], an OpenID Connect Issuer Identifier [OpenID.Core], are examples of things that might be used as "audience" parameter values. However, "audience" values used with a given authorization server must be unique within that server, to ensure that they are properly interpreted as the intended type of value. Multiple "audience" parameters may be used to indicate that the issued token is intended to be used at the multiple audiences listed. The "audience" and "resource" parameters may be used together to indicate multiple target services with a mix of logical names and resource URIs.

scope

OPTIONAL. A list of space-delimited, case-sensitive strings, as defined in Section 3.3 of [RFC6749], that allow the client to specify the desired scope of the requested security token in the context of the service or resource where the token will be used. The values and associated semantics of scope are service specific and expected to be described in the relevant service documentation.

requested_token_type

OPTIONAL. An identifier, as described in Section 3, for the type of the requested security token. If the requested type is unspecified, the issued token type is at the discretion of the authorization server and may be dictated by knowledge of the requirements of the service or resource indicated by the "resource" or "audience" parameter.

subject_token

REQUIRED. A security token that represents the identity of the party on behalf of whom the request is being made. Typically, the subject of this token will be the subject of the security token issued in response to the request.

subject_token_type

REQUIRED. An identifier, as described in Section 3, that indicates the type of the security token in the "subject_token" parameter.

actor_token

OPTIONAL. A security token that represents the identity of the acting party. Typically, this will be the party that is authorized to use the requested security token and act on behalf of the subject.

actor_token_type

An identifier, as described in Section 3, that indicates the type of the security token in the "actor_token" parameter. This is REQUIRED when the "actor_token" parameter is present in the request but MUST NOT be included otherwise.

In processing the request, the authorization server MUST perform the appropriate validation procedures for the indicated token type and, if the actor token is present, also perform the appropriate validation procedures for its indicated token type. The validity criteria and details of any particular token are beyond the scope of this document and are specific to the respective type of token and its content.

In the absence of one-time-use or other semantics specific to the token type, the act of performing a token exchange has no impact on the validity of the subject token or actor token. Furthermore, the exchange is a one-time event and does not create a tight linkage between the input and output tokens, so that (for example) while the expiration time of the output token may be influenced by that of the input token, renewal or extension of the input token is not expected to be reflected in the output token's properties. It may still be appropriate or desirable to propagate token revocation events. However, doing so is not a general property of the STS protocol and would be specific to a particular implementation, token type or deployment.

2.1.1. Relationship Between Resource, Audience and Scope

When requesting a token, the client can indicate the desired target service(s) where it intends to use that token by way of the "audience" and "resource" parameters, as well as indicating the desired scope of the requested token using the "scope" parameter. The semantics of such a request are that the client is asking for a token with the requested scope that is usable at all the requested target services. Effectively, the requested access rights of the token are the cartesian product of all the scopes at all the target services.

An authorization server may be unwilling or unable to fulfill any token request but the likelihood of an unfulfillable request is significantly higher when very broad access rights are being solicited. As such, in the absence of specific knowledge about the relationship of systems in a deployment, clients should exercise discretion in the breadth of the access requested, particularly the number of target services. An authorization server can use the "invalid_target" error code, defined in Section 2.2.2, to inform a client that it requested access to too many target services simultaneously.

2.2. Response

The authorization server responds to a token exchange request with a normal OAuth 2.0 response from the token endpoint, as specified in Section 5 of [RFC6749]. Additional details and explanation are provided in the following subsections.

2.2.1. Successful Response

If the request is valid and meets all policy and other criteria of the authorization server, a successful token response is constructed by adding the following parameters to the entity-body of the HTTP

response using the "application/json" media type, as specified by [RFC8259], and an HTTP 200 status code. The parameters are serialized into a JavaScript Object Notation (JSON) structure by adding each parameter at the top level. Parameter names and string values are included as JSON strings. Numerical values are included as JSON numbers. The order of parameters does not matter and can vary.

access_token

REQUIRED. The security token issued by the authorization server in response to the token exchange request. The "access_token" parameter from Section 5.1 of [RFC6749] is used here to carry the requested token, which allows this token exchange protocol to use the existing OAuth 2.0 request and response constructs defined for the token endpoint. The identifier "access_token" is used for historical reasons and the issued token need not be an OAuth access token.

issued_token_type

REQUIRED. An identifier, as described in Section 3, for the representation of the issued security token.

token_type

REQUIRED. A case-insensitive value specifying the method of using the access token issued, as specified in Section 7.1 of [RFC6749]. It provides the client with information about how to utilize the access token to access protected resources. For example, a value of "Bearer", as specified in [RFC6750], indicates that the issued security token is a bearer token and the client can simply present it as is without any additional proof of eligibility beyond the contents of the token itself. Note that the meaning of this parameter is different from the meaning of the "issued_token_type" parameter, which declares the representation of the issued security token; the term "token type" is more typically used with the aforementioned meaning as the structural or syntactical representation of the security token, as it is in all "*_token_type" parameters in this specification. If the issued token is not an access token or usable as an access token, then the "token_type" value "N_A" is used to indicate that an OAuth 2.0 "token_type" identifier is not applicable in that context.

expires_in

RECOMMENDED. The validity lifetime, in seconds, of the token issued by the authorization server. Oftentimes the client will not have the inclination or capability to inspect the content of the token and this parameter provides a consistent and token-type-agnostic indication of how long the token can be expected to be

valid. For example, the value 1800 denotes that the token will expire in thirty minutes from the time the response was generated.

scope

OPTIONAL, if the scope of the issued security token is identical to the scope requested by the client; otherwise, REQUIRED.

refresh_token

OPTIONAL. A refresh token will typically not be issued when the exchange is of one temporary credential (the subject_token) for a different temporary credential (the issued token) for use in some other context. A refresh token can be issued in cases where the client of the token exchange needs the ability to access a resource even when the original credential is no longer valid (e.g., user-not-present or offline scenarios where there is no longer any user entertaining an active session with the client). Profiles or deployments of this specification should clearly document the conditions under which a client should expect a refresh token in response to "urn:ietf:params:oauth:grant-type:token-exchange" grant type requests.

2.2.2. Error Response

If the request itself is not valid or if either the "subject_token" or "actor_token" are invalid for any reason, or are unacceptable based on policy, the authorization server MUST construct an error response, as specified in Section 5.2 of [RFC6749]. The value of the "error" parameter MUST be the "invalid_request" error code.

If the authorization server is unwilling or unable to issue a token for any target service indicated by the "resource" or "audience" parameters, the "invalid_target" error code SHOULD be used in the error response.

The authorization server MAY include additional information regarding the reasons for the error using the "error_description" as discussed in Section 5.2 of [RFC6749].

Other error codes may also be used, as appropriate.

2.3. Example Token Exchange

The following example demonstrates a hypothetical token exchange in which an OAuth resource server assumes the role of the client during the exchange. It trades an access token, which it received in a protected resource request, for a new token that it will use to call to a backend service (extra line breaks and indentation in the examples are for display purposes only).

Figure 1 shows the resource server receiving a protected resource request containing an OAuth access token in the Authorization header, as specified in Section 2.1 of [RFC6750].

```
GET /resource HTTP/1.1
Host: frontend.example.com
Authorization: Bearer accVkjcJyb4BWCxGsndESCJQbdfMogUC5PbRDqceLTC
```

Figure 1: Protected Resource Request

In Figure 2, the resource server assumes the role of client for the token exchange and the access token from the request in Figure 1 is sent to the authorization server using a request as specified in Section 2.1. The value of the "subject_token" parameter carries the access token and the value of the "subject_token_type" parameter indicates that it is an OAuth 2.0 access token. The resource server, acting in the role of the client, uses its identifier and secret to authenticate to the authorization server using the HTTP Basic authentication scheme. The "resource" parameter indicates the location of the backend service, <https://backend.example.com/api>, where the issued token will be used.

```
POST /as/token.oauth2 HTTP/1.1
Host: as.example.com
Authorization: Basic cnMwODpsb25nLXNlY3VyZS1yYW5kb20tc2VjcmV0
Content-Type: application/x-www-form-urlencoded

grant_type=urn%3Aietf%3Aparams%3Aoauth%3Agrant-type%3Atoken-exchange
&resource=https%3A%2F%2Fbackend.example.com%2Fapi
&subject_token=accVkjcJyb4BWCxGsndESCJQbdfMogUC5PbRDqceLTC
&subject_token_type=
urn%3Aietf%3Aparams%3Aoauth%3Atoken-type%3Aaccess_token
```

Figure 2: Token Exchange Request

The authorization server validates the client credentials and the "subject_token" presented in the token exchange request. From the "resource" parameter, the authorization server is able to determine the appropriate policy to apply to the request and issues a token suitable for use at <https://backend.example.com>. The "access_token" parameter of the response shown in Figure 3 contains the new token, which is itself a bearer OAuth access token that is valid for one minute. The token happens to be a JWT; however, its structure and format are opaque to the client so the "issued_token_type" indicates only that it is an access token.

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

{
  "access_token": "eyJhbGciOiJIJFUzI1NiIsImtpZCI6IjllciJ9.eyJhdWQiOiJo
dHRwc3ovL2JhY2t1bmQuZXhhbXBsZS5jb20iLCJpc3MiOiJodHRwc3ovL2FzLmV
4YW1wbGUuY29tIiwiaXNwIjo6NDQxOTE3NTkzLCJpYXQiOjE0NDE1MTc1MzMzIn
NlYiOi6ImJkY0BleGFtcGxlLmNvbSI6InNjb3BlIjoieXBpIn0.40y3ZgQedw6rx
f59WlwHDD9jryFOr0_Wh3CGozQBihNBhnXEQgU85AI9x3KmsPottVMLPIWvmDCM
y5-kdXjwhw",
  "issued_token_type":
    "urn:ietf:params:oauth:token-type:access_token",
  "token_type": "Bearer",
  "expires_in": 60
}
```

Figure 3: Token Exchange Response

The resource server can then use the newly acquired access token in making a request to the backend server as illustrated in Figure 4.

```
GET /api HTTP/1.1
Host: backend.example.com
Authorization: Bearer eyJhbGciOiJIUzI1NiIsImtpZCI6IjllciJ9.eyJhdWQiOiJodHRwczovL2JhY2t1bmQuZXhhbXBsZS5jb20iLCJpc3MiOiJodHRwczovL2FzLmV4YW1wbGUuYy29tIiwiaXNjaWJoxNDQxOTE3NTkzLCJpYXQiOiJEONDE5MTclMzMsInN1YiI6ImUuYy29tIiwiaWF0IjoiYXBpLn0.40y3ZgQedw6rxsf59lwhDD9jrYfOrO_Wh3CGozQBihNBhnXEQgU85AI9x3KmsPottVMLPIWvmDCMy5-kdXjwhw
```

Figure 4: Backend Protected Resource Request

Additional examples can be found in Appendix A.

3. Token Type Identifiers

Several parameters in this specification utilize an identifier as the value to describe the token in question. Specifically, they are the "requested_token_type", "subject_token_type", "actor_token_type" parameters of the request and the "issued_token_type" member of the response. Token type identifiers are URIs. Token Exchange can work with both tokens issued by other parties and tokens from the given authorization server. For the former the token type identifier indicates the syntax (e.g., JWT or SAML 2.0) so the authorization server can parse it; for the latter it indicates what the given authorization server issued it for (e.g., access_token or refresh token).

The following token type identifiers are defined by this specification. Other URIs MAY be used to indicate other token types.

`urn:ietf:params:oauth:token-type:access_token`

Indicates that the token is an OAuth 2.0 access token issued by the given authorization server.

`urn:ietf:params:oauth:token-type:refresh_token`

Indicates that the token is an OAuth 2.0 refresh token issued by the given authorization server.

`urn:ietf:params:oauth:token-type:id_token`

Indicates that the token is an ID Token, as defined in Section 2 of [OpenID.Core].

`urn:ietf:params:oauth:token-type:saml1`

Indicates that the token is a base64url-encoded SAML 1.1 [OASIS.saml-core-1.1] assertion.

`urn:ietf:params:oauth:token-type:saml2`

Indicates that the token is a base64url-encoded SAML 2.0 [OASIS.saml-core-2.0-os] assertion.

The value `"urn:ietf:params:oauth:token-type:jwt"`, which is defined in Section 9 of [JWT], indicates that the token is a JWT.

The distinction between an access token and a JWT is subtle. An access token represents a delegated authorization decision, whereas JWT is a token format. An access token can be formatted as a JWT but doesn't necessarily have to be. And a JWT might well be an access token but not all JWTs are access tokens. The intent of this specification is that `"urn:ietf:params:oauth:token-type:access_token"` be an indicator that the token is a typical OAuth access token issued by the authorization server in question, opaque to the client, and usable the same manner as any other access token obtained from that authorization server. (It could well be a JWT, but the client isn't and needn't be aware of that fact.) Whereas, `"urn:ietf:params:oauth:token-type:jwt"` is to indicate specifically that a JWT is being requested or sent (perhaps in a cross-domain use-case where the JWT is used as an authorization grant to obtain an access token from a different authorization server as is facilitated by [RFC7523]).

Note that for tokens which are binary in nature, the URI used for conveying them needs to be associated with the semantics of a base64 or other encoding suitable for usage with HTTP and OAuth.

4. JSON Web Token Claims and Introspection Response Parameters

It is useful to have defined mechanisms to express delegation within a token as well as to express authorization to delegate or impersonate. Although the token exchange protocol described herein can be used with any type of token, this section defines claims to express such semantics specifically for JWTs and in an OAuth 2.0 Token Introspection [RFC7662] response. Similar definitions for other types of tokens are possible but beyond the scope of this specification.

Note that the claims not established herein but used in examples and descriptions, such as "iss", "sub", "exp", etc., are defined by [JWT].

4.1. "act" (Actor) Claim

The "act" (actor) claim provides a means within a JWT to express that delegation has occurred and identify the acting party to whom authority has been delegated. The "act" claim value is a JSON object and members in the JSON object are claims that identify the actor. The claims that make up the "act" claim identify and possibly provide additional information about the actor. For example, the combination of the two claims "iss" and "sub" might be necessary to uniquely identify an actor.

However, claims within the "act" claim pertain only to the identity of the actor and are not relevant to the validity of the containing JWT in the same manner as the top-level claims. Consequently, non-identity claims (e.g., "exp", "nbf", and "aud") are not meaningful when used within an "act" claim, and therefore are not used.

Figure 5 illustrates the "act" (actor) claim within a JWT Claims Set. The claims of the token itself are about user@example.com while the "act" claim indicates that admin@example.com is the current actor.

```
{
  "aud": "https://consumer.example.com",
  "iss": "https://issuer.example.com",
  "exp": 1443904177,
  "nbf": 1443904077,
  "sub": "user@example.com",
  "act": {
    "sub": "admin@example.com"
  }
}
```

Figure 5: Actor Claim

A chain of delegation can be expressed by nesting one "act" claim within another. The outermost "act" claim represents the current actor while nested "act" claims represent prior actors. The least recent actor is the most deeply nested. The nested "act" claims serve as a history trail that connects the initial request and subject through the various delegation steps undertaken before reaching the current actor. In this sense, the current actor is considered to include the entire authorization/delegation history, leading naturally to the nested structure described here.

For the purpose of applying access control policy, the consumer of a token MUST only consider the token's top-level claims and the party identified as the current actor by the "act" claim. Prior actors identified by any nested "act" claims are informational only and are not to be considered in access control decisions.

The following example in Figure 6 illustrates nested "act" (actor) claims within a JWT Claims Set. The claims of the token itself are about user@example.com while the "act" claim indicates that the system https://service16.example.com is the current actor and https://service77.example.com was a prior actor. Such a token might come about as the result of service16 receiving a token in a call from service77 and exchanging it for a token suitable to call service26 while the authorization server notes the situation in the newly issued token.

```
{
  "aud": "https://service26.example.com",
  "iss": "https://issuer.example.com",
  "exp": 1443904100,
  "nbf": 1443904000,
  "sub": "user@example.com",
  "act": {
    {
      "sub": "https://service16.example.com",
      "act": {
        {
          "sub": "https://service77.example.com"
        }
      }
    }
  }
}
```

Figure 6: Nested Actor Claim

When included as a top-level member of an OAuth token introspection response, "act" has the same semantics and format as the claim of the same name.

4.2. "scope" (Scopes) Claim

The value of the "scope" claim is a JSON string containing a space-separated list of scopes associated with the token, in the format described in Section 3.3 of [RFC6749].

Figure 7 illustrates the "scope" claim within a JWT Claims Set.

```
{
  "aud": "https://consumer.example.com",
  "iss": "https://issuer.example.com",
  "exp": 1443904177,
  "nbf": 1443904077,
  "sub": "dgaf4mvfs75Fci_FL3heQA",
  "scope": "email profile phone address"
}
```

Figure 7: Scopes Claim

OAuth 2.0 Token Introspection [RFC7662] already defines the "scope" parameter to convey the scopes associated with the token.

4.3. "client_id" (Client Identifier) Claim

The "client_id" claim carries the client identifier of the OAuth 2.0 [RFC6749] client that requested the token.

The following example in Figure 8 illustrates the "client_id" claim within a JWT Claims Set indicating an OAuth 2.0 client with "s6BhdRkqt3" as its identifier.

```
{
  "aud": "https://consumer.example.com",
  "iss": "https://issuer.example.com",
  "exp": 1443904177,
  "sub": "user@example.com",
  "client_id": "s6BhdRkqt3"
}
```

Figure 8: Client Identifier Claim

OAuth 2.0 Token Introspection [RFC7662] already defines the "client_id" parameter as the client identifier for the OAuth 2.0 client that requested the token.

4.4. "may_act" (Authorized Actor) Claim

The "may_act" claim makes a statement that one party is authorized to become the actor and act on behalf of another party. The claim might be used, for example, when a "subject_token" is presented to the token endpoint in a token exchange request and "may_act" claim in the subject token can be used by the authorization server to determine whether the client (or party identified in the "actor_token") is authorized to engage in the requested delegation or impersonation.

The claim value is a JSON object and members in the JSON object are claims that identify the party that is asserted as being eligible to act for the party identified by the JWT containing the claim. The claims that make up the "may_act" claim identify and possibly provide additional information about the authorized actor. For example, the combination of the two claims "iss" and "sub" are sometimes necessary to uniquely identify an authorized actor, while the "email" claim might be used to provide additional useful information about that party.

However, claims within the "may_act" claim pertain only to the identity of that party and are not relevant to the validity of the containing JWT in the same manner as top-level claims. Consequently, claims such as "exp", "nbf", and "aud" are not meaningful when used within a "may_act" claim, and therefore are not used.

Figure 9 illustrates the "may_act" claim within a JWT Claims Set. The claims of the token itself are about user@example.com while the "may_act" claim indicates that admin@example.com is authorized to act on behalf of user@example.com.

```
{
  "aud": "https://consumer.example.com",
  "iss": "https://issuer.example.com",
  "exp": 1443904177,
  "nbf": 1443904077,
  "sub": "user@example.com",
  "may_act": {
    "sub": "admin@example.com"
  }
}
```

Figure 9: Authorized Actor Claim

When included as a top-level member of an OAuth token introspection response, "may_act" has the same semantics and format as the claim of the same name.

5. Security Considerations

Much of the guidance from Section 10 of [RFC6749], the Security Considerations in The OAuth 2.0 Authorization Framework, is also applicable here. Furthermore, [RFC6819] provides additional security considerations for OAuth and [I-D.ietf-oauth-security-topics] has updated security guidance based on deployment experience and new threats that have emerged since OAuth 2.0 was originally published.

All of the normal security issues that are discussed in [JWT], especially in relationship to comparing URIs and dealing with unrecognized values, also apply here.

In addition, both delegation and impersonation introduce unique security issues. Any time one principal is delegated the rights of another principal, the potential for abuse is a concern. The use of the "scope" claim (in addition to other typical constraints such as a limited token lifetime) is suggested to mitigate potential for such abuse, as it restricts the contexts in which the delegated rights can be exercised.

6. Privacy Considerations

Tokens employed in the context of the functionality described herein may contain privacy-sensitive information and, to prevent disclosure of such information to unintended parties, MUST only be transmitted over encrypted channels, such as Transport Layer Security (TLS). In cases where it is desirable to prevent disclosure of certain information to the client, the token MUST be encrypted to its intended recipient. Deployments SHOULD determine the minimally necessary amount of data and only include such information in issued tokens. In some cases, data minimization may include representing only an anonymous or pseudonymous user.

7. IANA Considerations

7.1. OAuth URI Registration

This specification registers the following values in the IANA "OAuth URI" registry [IANA.OAuth.Parameters] established by [RFC6755].

7.1.1. Registry Contents

- o URN: urn:ietf:params:oauth:grant-type:token-exchange
- o Common Name: Token exchange grant type for OAuth 2.0
- o Change controller: IESG
- o Specification Document: Section 2.1 of [[this specification]]

- o URN: urn:ietf:params:oauth:token-type:access_token
- o Common Name: Token type URI for an OAuth 2.0 access token
- o Change controller: IESG
- o Specification Document: Section 3 of [[this specification]]

- o URN: urn:ietf:params:oauth:token-type:refresh_token
- o Common Name: Token type URI for an OAuth 2.0 refresh token
- o Change controller: IESG
- o Specification Document: Section 3 of [[this specification]]

- o URN: urn:ietf:params:oauth:token-type:id_token
- o Common Name: Token type URI for an ID Token
- o Change controller: IESG
- o Specification Document: Section 3 of [[this specification]]

- o URN: urn:ietf:params:oauth:token-type:saml1
- o Common Name: Token type URI for a base64url-encoded SAML 1.1 assertion
- o Change Controller: IESG
- o Specification Document: Section 3 of [[this specification]]

- o URN: urn:ietf:params:oauth:token-type:saml2
- o Common Name: Token type URI for a base64url-encoded SAML 2.0 assertion
- o Change Controller: IESG
- o Specification Document: Section 3 of [[this specification]]

7.2. OAuth Parameters Registration

This specification registers the following values in the IANA "OAuth Parameters" registry [IANA.OAuth.Parameters] established by [RFC6749].

7.2.1. Registry Contents

- o Parameter name: resource
- o Parameter usage location: token request
- o Change controller: IESG
- o Specification document(s): Section 2.1 of [[this specification]]

- o Parameter name: audience
- o Parameter usage location: token request
- o Change controller: IESG
- o Specification document(s): Section 2.1 of [[this specification]]

- o Parameter name: requested_token_type
- o Parameter usage location: token request
- o Change controller: IESG
- o Specification document(s): Section 2.1 of [[this specification]]

- o Parameter name: subject_token
- o Parameter usage location: token request
- o Change controller: IESG
- o Specification document(s): Section 2.1 of [[this specification]]

- o Parameter name: subject_token_type
- o Parameter usage location: token request
- o Change controller: IESG

- o Specification document(s): Section 2.1 of [[this specification]]
- o Parameter name: actor_token
- o Parameter usage location: token request
- o Change controller: IESG
- o Specification document(s): Section 2.1 of [[this specification]]
- o Parameter name: actor_token_type
- o Parameter usage location: token request
- o Change controller: IESG
- o Specification document(s): Section 2.1 of [[this specification]]
- o Parameter name: issued_token_type
- o Parameter usage location: token response
- o Change controller: IESG
- o Specification document(s): Section 2.2.1 of [[this specification]]

7.3. OAuth Access Token Type Registration

This specification registers the following access token type in the IANA "OAuth Access Token Types" registry [IANA.OAuth.Parameters] established by [RFC6749].

7.3.1. Registry Contents

- o Type name: N_A
- o Additional Token Endpoint Response Parameters: (none)
- o HTTP Authentication Scheme(s): (none)
- o Change controller: IESG
- o Specification document(s): Section 2.2.1 of [[this specification]]

7.4. JSON Web Token Claims Registration

This specification registers the following Claims in the IANA "JSON Web Token Claims" registry [IANA.JWT.Claims] established by [JWT].

7.4.1. Registry Contents

- o Claim Name: "act"
- o Claim Description: Actor
- o Change Controller: IESG
- o Specification Document(s): Section 4.1 of [[this specification]]
- o Claim Name: "scope"
- o Claim Description: Scope Values
- o Change Controller: IESG

- o Specification Document(s): Section 4.2 of [[this specification]]
- o Claim Name: "client_id"
- o Claim Description: Client Identifier
- o Change Controller: IESG
- o Specification Document(s): Section 4.3 of [[this specification]]
- o Claim Name: "may_act"
- o Claim Description: Authorized Actor - the party that is authorized to become the actor
- o Change Controller: IESG
- o Specification Document(s): Section 4.4 of [[this specification]]

7.5. OAuth Token Introspection Response Registration

This specification registers the following values in the IANA "OAuth Token Introspection Response" registry [IANA.OAuth.Parameters] established by [RFC7662].

7.5.1. Registry Contents

- o Claim Name: "act"
- o Claim Description: Actor
- o Change Controller: IESG
- o Specification Document(s): Section 4.1 of [[this specification]]
- o Claim Name: "may_act"
- o Claim Description: Authorized Actor - the party that is authorized to become the actor
- o Change Controller: IESG
- o Specification Document(s): Section 4.4 of [[this specification]]

7.6. OAuth Extensions Error Registration

This specification registers the following values in the IANA "OAuth Extensions Error" registry [IANA.OAuth.Parameters] established by [RFC6749].

7.6.1. Registry Contents

- o Error Name: "invalid_target"
- o Error Usage Location: token error response
- o Related Protocol Extension: OAuth 2.0 Token Exchange
- o Change Controller: IETF
- o Specification Document(s): Section 2.2.2 of [[this specification]]

8. References

8.1. Normative References

- [IANA.JWT.Claims] IANA, "JSON Web Token Claims", <<http://www.iana.org/assignments/jwt>>.
- [IANA.OAuth.Parameters] IANA, "OAuth Parameters", <<http://www.iana.org/assignments/oauth-parameters>>.
- [JWT] Jones, M., Bradley, J., and N. Sakimura, "JSON Web Token (JWT)", RFC 7519, DOI 10.17487/RFC7519, May 2015, <<http://tools.ietf.org/html/rfc7519>>.
- [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>>.
- [RFC3986] Berners-Lee, T., Fielding, R., and L. Masinter, "Uniform Resource Identifier (URI): Generic Syntax", STD 66, RFC 3986, DOI 10.17487/RFC3986, January 2005, <<https://www.rfc-editor.org/info/rfc3986>>.
- [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>>.
- [RFC7662] Richer, J., Ed., "OAuth 2.0 Token Introspection", RFC 7662, DOI 10.17487/RFC7662, October 2015, <<https://www.rfc-editor.org/info/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/info/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/info/rfc8259>>.

8.2. Informative References

- [I-D.ietf-oauth-resource-indicators]
Campbell, B., Bradley, J., and H. Tschofenig, "Resource Indicators for OAuth 2.0", draft-ietf-oauth-resource-indicators-02 (work in progress), January 2019.
- [I-D.ietf-oauth-security-topics]
Lodderstedt, T., Bradley, J., Labunets, A., and D. Fett, "OAuth 2.0 Security Best Current Practice", draft-ietf-oauth-security-topics-13 (work in progress), July 2019.
- [OASIS.saml-core-1.1]
Maler, E., Mishra, P., and R. Philpott, "Assertions and Protocol for the OASIS Security Assertion Markup Language (SAML) V1.1", OASIS Standard oasis-sstc-saml-core-1.1, September 2003.
- [OASIS.saml-core-2.0-os]
Cantor, S., Kemp, J., Philpott, R., and E. Maler, "Assertions and Protocol for the OASIS Security Assertion Markup Language (SAML) V2.0", OASIS Standard saml-core-2.0-os, March 2005.
- [OpenID.Core]
Sakimura, N., Bradley, J., Jones, M., de Medeiros, B., and C. Mortimore, "OpenID Connect Core 1.0", November 2014, <http://openid.net/specs/openid-connect-core-1_0.html>.
- [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/info/rfc6750>>.
- [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>>.
- [RFC6819] Lodderstedt, T., Ed., McGloin, M., and P. Hunt, "OAuth 2.0 Threat Model and Security Considerations", RFC 6819, DOI 10.17487/RFC6819, January 2013, <<https://www.rfc-editor.org/info/rfc6819>>.
- [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/info/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/info/rfc7523>>.

[WS-Trust]

Nadalin, A., Goodner, M., Gudgin, M., Barbir, A., and H. Granqvist, "WS-Trust 1.4", February 2012, <<http://docs.oasis-open.org/ws-sx/ws-trust/v1.4/ws-trust.html>>.

Appendix A. Additional Token Exchange Examples

Two example token exchanges are provided in the following sections illustrating impersonation and delegation, respectively (with extra line breaks and indentation for display purposes only).

A.1. Impersonation Token Exchange Example

A.1.1. Token Exchange Request

In the following token exchange request, a client is requesting a token with impersonation semantics (with only a "subject_token" and no "actor_token", delegation is impossible). The client tells the authorization server that it needs a token for use at the target service with the logical name "urn:example:cooperation-context".

```
POST /as/token.oauth2 HTTP/1.1
Host: as.example.com
Content-Type: application/x-www-form-urlencoded

grant_type=urn%3Aietf%3Aparams%3Aoauth%3Agrant-type%3Atoken-exchange
&audience=urn%3Aexample%3Acooperation-context
&subject_token=eyJhbGciOiJIUzI1NiIsImtpZCI6IjE2In0.eyJhdWQiOiJodHRwczovL2FzLmV4YW1wbGUuY29tIiwiaXNzIjoiaHR0cHM6Ly9vcmlnaW5hbC1pc3N1ZXIuZXhhbXBsZS5uZXQiLCJleHAiOjE0NDE5MTA2MDAsIm5iZiI6MTQ0MTkwOTAwMCwic3ViIjoiYmRjQGV4YW1wbGUubmV0Iiwic2NvcGUiOiJvcmlcnMgcHJvZmlsZSBoaXN0b3J5In0.PRBg-jXn4cJujlgmYXFiGkZzRuzbXZ_sDxdE98ddW44ufsbWLKd3JJ1VZhF64pbTtfjy4VXFVBDAQpKjn5JzAw
&subject_token_type=urn%3Aietf%3Aparams%3Aoauth%3Atoken-type%3Ajwt
```

Figure 10: Token Exchange Request

A.1.2. Subject Token Claims

The "subject_token" in the prior request is a JWT and the decoded JWT Claims Set is shown here. The JWT is intended for consumption by the authorization server within a specific time window. The subject of

the JWT ("bdc@example.net") is the party on behalf of whom the new token is being requested.

```
{
  "aud": "https://as.example.com",
  "iss": "https://original-issuer.example.net",
  "exp": 1441910600,
  "nbf": 1441909000,
  "sub": "bdc@example.net",
  "scope": "orders profile history"
}
```

Figure 11: Subject Token Claims

A.1.3. Token Exchange Response

The "access_token" parameter of the token exchange response shown below contains the new token that the client requested. The other parameters of the response indicate that the token is a bearer access token that expires in an hour.

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

```
{
  "access_token": "eyJhbGciOiJIJFZlIiwiaXNzIjoiYmRjQGV4YW1wbGUubmV0Iiwic2NvcGUiOiJvcmlcnMgcHJvZmlsZSBoaXN0b3J5In0.rMdWpSGNACTvnFuOL74sYZ6MVuld2Z2WkGLmQeR9ztj6w2OXraQlkJmGjyiCq24kCB7AI2VqVx13wSWnVKh85A",
  "issued_token_type":
    "urn:ietf:params:oauth:token-type:access_token",
  "token_type": "Bearer",
  "expires_in": 3600
}
```

Figure 12: Token Exchange Response

A.1.4. Issued Token Claims

The decoded JWT Claims Set of the issued token is shown below. The new JWT is issued by the authorization server and intended for consumption by a system entity known by the logical name "urn:example:cooperation-context" any time before its expiration. The subject ("sub") of the JWT is the same as the subject the token used to make the request, which effectively enables the client to

impersonate that subject at the system entity known by the logical name of "urn:example:cooperation-context" by using the token.

```
{
  "aud": "urn:example:cooperation-context",
  "iss": "https://as.example.com",
  "exp": 1441913610,
  "sub": "bdc@example.net",
  "scope": "orders profile history"
}
```

Figure 13: Issued Token Claims

A.2. Delegation Token Exchange Example

A.2.1. Token Exchange Request

In the following token exchange request, a client is requesting a token and providing both a "subject_token" and an "actor_token". The client tells the authorization server that it needs a token for use at the target service with the logical name "urn:example:cooperation-context". Policy at the authorization server dictates that the issued token be a composite.

```
POST /as/token.oauth2 HTTP/1.1
Host: as.example.com
Content-Type: application/x-www-form-urlencoded

grant_type=urn%3Aietf%3Aparams%3Aoauth%3Agrant-type%3Atoken-exchange
&audience=urn%3Aexample%3Acooperation-context
&subject_token=eyJhbGciOiJIUzI1NiIsImtpZCI6IjE2In0.eyJhdWQiOiJodHRwczovL2FzLmV4YWlwbGUuY29tIiwiaXNzIjoiaHR0cHM6Ly9vcmlnaW5hbC1pc3N1ZXIuZSxhhbXBsZS5uZXQlLCJleHAiOjE0NDE5MTAwNjAsInNjb3BlIjoic3Rh dHVzIGZlZWQiLCJzdWIiOiJlc2VyQG V4YWlwbGUubmV0IiwibWF5X2FjdCI6eyJzdWIiOiJhZG1pbkBLEGFtcGxlLm5ldCJ9fQ.4rPRSWihQbpMIgAmAoqaJoJAxj-p2X8_fAtAGTXrvMxU-eEZhnXqY0_AOZgLdxw5DyLzua8H_I10MCcckF-Q_g
&subject_token_type=urn%3Aietf%3Aparams%3Aoauth%3Atoken-type%3Ajwt
&actor_token=eyJhbGciOiJIUzI1NiIsImtpZCI6IjE2In0.eyJhdWQiOiJodHRwczovL2FzLmV4YWlwbGUuY29tIiwiaXNzIjoiaHR0cHM6Ly9vcmlnaW5hbC1pc3N1ZXIuZSxhhbXBsZS5uZXQlLCJleHAiOjE0NDE5MTAwNjAsInN1YiI6ImFkbWluQG V4YWlwbGUubmV0In0.7YQ-3zpFfhUvzje5oqw8COCvN5uP6NsKik9CVV6cAO f4QKgM-tKfiOwcgZoUuD L2tEs6tqPlcB lMjiSzEjm3yBg
&actor_token_type=urn%3Aietf%3Aparams%3Aoauth%3Atoken-type%3Ajwt
```

Figure 14: Token Exchange Request

A.2.2. Subject Token Claims

The "subject_token" in the prior request is a JWT and the decoded JWT Claims Set is shown here. The JWT is intended for consumption by the authorization server before a specific expiration time. The subject of the JWT ("user@example.net") is the party on behalf of whom the new token is being requested.

```
{
  "aud": "https://as.example.com",
  "iss": "https://original-issuer.example.net",
  "exp": 1441910060,
  "scope": "status feed",
  "sub": "user@example.net",
  "may_act": {
    "sub": "admin@example.net"
  }
}
```

Figure 15: Subject Token Claims

A.2.3. Actor Token Claims

The "actor_token" in the prior request is a JWT and the decoded JWT Claims Set is shown here. This JWT is also intended for consumption by the authorization server before a specific expiration time. The subject of the JWT ("admin@example.net") is the actor that will wield the security token being requested.

```
{
  "aud": "https://as.example.com",
  "iss": "https://original-issuer.example.net",
  "exp": 1441910060,
  "sub": "admin@example.net"
}
```

Figure 16: Actor Token Claims

A.2.4. Token Exchange Response

The "access_token" parameter of the token exchange response shown below contains the new token that the client requested. The other parameters of the response indicate that the token is a JWT that expires in an hour and that the access token type is not applicable since the issued token is not an access token.

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

{
  "access_token": "eyJhbGciOiJIJFUiI1NiIsImtpZCI6IjcyIn0.eyJhdWQiOiJ1cm46ZXhnbXBsZTpjb29wZXJhdGlvbiljb250ZXh0IiwiaXNzIjoiaHR0cHM6Ly9hcy5leGFtcGxlLmNvbSIsImV4cCI6MTQ0MTkxMzYxMCwic2NvcGUiOiJzdGF0dXMgZmVlZCIsInN1YiI6InVzZXJAZXhnbXBsZS5uZXQlLCJhY3QiOiOnsic3ViIjojYWRtaW5AZXhnbXBsZS5uZXQifX0.3paKl9UySKYB5ng6_cUtQ2ql08Rc_y7Mea7IwEXTcYbNdwG9-G1EKCFe5fw3H0hwX-MSZ49Wpcb1SiAZaOQBtw",
  "issued_token_type": "urn:ietf:params:oauth:token-type:jwt",
  "token_type": "N_A",
  "expires_in": 3600
}
```

Figure 17: Token Exchange Response

A.2.5. Issued Token Claims

The decoded JWT Claims Set of the issued token is shown below. The new JWT is issued by the authorization server and intended for consumption by a system entity known by the logical name "urn:example:cooperation-context" any time before its expiration. The subject ("sub") of the JWT is the same as the subject of the "subject_token" used to make the request. The actor ("act") of the JWT is the same as the subject of the "actor_token" used to make the request. This indicates delegation and identifies "admin@example.net" as the current actor to whom authority has been delegated to act on behalf of "user@example.net".

```
{
  "aud": "urn:example:cooperation-context",
  "iss": "https://as.example.com",
  "exp": 1441913610,
  "scope": "status feed",
  "sub": "user@example.net",
  "act":
  {
    "sub": "admin@example.net"
  }
}
```

Figure 18: Issued Token Claims

Appendix B. Acknowledgements

This specification was developed within the OAuth Working Group, which includes dozens of active and dedicated participants. It was produced under the chairmanship of Hannes Tschofenig, Derek Atkins, and Rifaat Shekh-Yusef with Kathleen Moriarty, Stephen Farrell, Eric Rescorla, Roman Danyliw, and Benjamin Kaduk serving as Security Area Directors. The following individuals contributed ideas, feedback, and wording to this specification:

Caleb Baker, Vittorio Bertocci, Mike Brown, Thomas Broyer, Roman Danyliw, William Denniss, Vladimir Dzhuvinov, Eric Fazendin, Phil Hunt, Benjamin Kaduk, Jason Keglovitz, Torsten Lodderstedt, Barry Leiba, Adam Lewis, James Manger, Nov Mataka, Matt Miller, Hilarie Orman, Matthew Perry, Eric Rescorla, Justin Richer, Adam Roach, Rifaat Shekh-Yusef, Scott Tomilson, and Hannes Tschofenig.

Appendix C. Document History

[[to be removed by the RFC Editor before publication as an RFC]]

-19

- o Fix-up changes introduced in -18.
- o Fix invalid JSON in the Nested Actor Claim example.
- o Reference figure numbers in text when introducing the examples in Section 2 and 4.
- o Editorial updates from additional IESG evaluation comments.
- o Add an informational reference to ietf-oauth-resource-indicators
- o Update ietf-oauth-security-topics ref to 13

-18

- o Editorial updates based on a few more IESG evaluation comments.

-17

- o Editorial improvements and example fixes resulting from IESG evaluation comments.
- o Added a pointer to RFC6749's Appendix B. on the "Use of application/x-www-form-urlencoded Media Type" as a way of providing a normative citation (by reference) for the media type.
- o Strengthened some of the wording in the privacy considerations to bring it inline with RFC 7519 Sec. 12 and RFC 6749 Sec. 10.8.

-16

- o Fixed typo and added an AD to Acknowledgements.

-15

- o Updated the nested actor claim example to (hopefully) be more straightforward.
- o Reworked Privacy Considerations to say to use TLS in transit, minimize the amount of information in the token, and encrypt the token if disclosure of its information to the client is a concern per https://mailarchive.ietf.org/arch/msg/secdir/KJhx4aq_U5uk3k6zpYP-CEHbpVM
- o Moved the Security and Privacy Considerations sections to before the IANA Considerations.

-14

- o Added text in Section 4.1 about the "act" claim stating that only the top-level claims and the current actor are to be considered in applying access control decisions.

-13

- o Updated the claim name and value syntax for scope to be consistent with the treatment of scope in RFC 7662 OAuth 2.0 Token Introspection.
- o Updated the client identifier claim name to be consistent with the treatment of client id in RFC 7662 OAuth 2.0 Token Introspection.

-12

- o Updated to use the boilerplate from RFC 8174.

-11

- o Added new WG chair and AD to the Acknowledgements.
- o Applied clarifications suggested during AD review by EKR.

-10

- o Defined token type URIs for base64url-encoded SAML 1.1 and SAML 2.0 assertions.
- o Applied editorial fixes.

-09

- o Changed "security tokens obtained could be used in a number of contexts" to "security tokens obtained may be used in a number of contexts" per a WGLC suggestion.

- o Clarified that the validity of the subject or actor token have no impact on the validity of the issued token after the exchange has occurred per a WGLC comment.
- o Changed use of `invalid_target` error code to a SHOULD per a WGLC comment.
- o Clarified text about non-identity claims within the "act" claim being meaningless per a WGLC comment.
- o Added brief Privacy Considerations section per WGLC comments.

-08

- o Use the bibxml reference for OpenID.Core rather than defining it inline.
- o Added editor role for Campbell.
- o Minor clarification of the text for `actor_token`.

-07

- o Fixed typo (desecration -> discretion).
- o Added an explanation of the relationship between scope, audience and resource in the request and added an "invalid_target" error code enabling the AS to tell the client that the requested audiences/resources were too broad.

-06

- o Drop "An STS for the REST of Us" from the title.
- o Drop "heavyweight" and "lightweight" from the abstract and introduction.
- o Clarifications on the language around `xxxxxx_token_type`.
- o Remove the `want_composite` parameter.
- o Add a short mention of proof-of-possession style tokens to the introduction and remove the respective open issue.

-05

- o Defined the JWT claim "cid" to express the OAuth 2.0 client identifier of the client that requested the token.
- o Defined and requested registration for "act" and "may_act" as Token introspection response parameters (in addition to being JWT claims).
- o Loosen up the language about `refresh_token` in the response to OPTIONAL from NOT RECOMMENDED based on feedback from real world deployment experience.
- o Add clarifying text about the distinction between JWT and access token URIs.
- o Close out (remove) some of the Open Issues bullets that have been resolved.

-04

- o Clarified that the "resource" and "audience" request parameters can be used at the same time (via <http://www.ietf.org/mail-archive/web/oauth/current/msg15335.html>).
- o Clarified subject/actor token validity after token exchange and explained a bit more about the recommendation to not issue refresh tokens (via <http://www.ietf.org/mail-archive/web/oauth/current/msg15318.html>).
- o Updated the examples appendix to use an issuer value that doesn't imply that the client issued and signed the tokens and used "Bearer" and "urn:ietf:params:oauth:token-type:access_token" in one of the responses (via <http://www.ietf.org/mail-archive/web/oauth/current/msg15335.html>).
- o Defined and registered urn:ietf:params:oauth:token-type:id_token, since some use cases perform token exchanges for ID Tokens and no URI to indicate that a token is an ID Token had previously been defined.

-03

- o Updated the document editors (adding Campbell, Bradley, and Mortimore).
- o Added to the title.
- o Added to the abstract and introduction.
- o Updated the format of the request to use application/x-www-form-urlencoded request parameters and the response to use the existing token endpoint JSON parameters defined in OAuth 2.0.
- o Changed the grant type identifier to urn:ietf:params:oauth:grant-type:token-exchange.
- o Added RFC 6755 registration requests for urn:ietf:params:oauth:token-type:refresh_token, urn:ietf:params:oauth:token-type:access_token, and urn:ietf:params:oauth:grant-type:token-exchange.
- o Added RFC 6749 registration requests for request/response parameters.
- o Removed the Implementation Considerations and the requirement to support JWTs.
- o Clarified many aspects of the text.
- o Changed "on_behalf_of" to "subject_token", "on_behalf_of_token_type" to "subject_token_type", "act_as" to "actor_token", and "act_as_token_type" to "actor_token_type".
- o Added an "audience" request parameter used to indicate the logical names of the target services at which the client intends to use the requested security token.
- o Added a "want_composite" request parameter used to indicate the desire for a composite token rather than trying to infer it from the presence/absence of token(s) in the request.

- o Added a "resource" request parameter used to indicate the URLs of resources at which the client intends to use the requested security token.
- o Specified that multiple "audience" and "resource" request parameter values may be used.
- o Defined the JWT claim "act" (actor) to express the current actor or delegation principal.
- o Defined the JWT claim "may_act" to express that one party is authorized to act on behalf of another party.
- o Defined the JWT claim "scp" (scopes) to express OAuth 2.0 scope-token values.
- o Added the "N_A" (not applicable) OAuth Access Token Type definition for use in contexts in which the token exchange syntax requires a "token_type" value, but in which the token being issued is not an access token.
- o Added examples.

-02

- o Enabled use of Security Token types other than JWTs for "act_as" and "on_behalf_of" request values.
- o Referenced the JWT and OAuth Assertions RFCs.

-01

- o Updated references.

-00

- o Created initial working group draft from draft-jones-oauth-token-exchange-01.

Authors' Addresses

Michael B. Jones
Microsoft

Email: mbj@microsoft.com
URI: <http://self-issued.info/>

Anthony Nadalin
Microsoft

Email: tonynad@microsoft.com

Brian Campbell (editor)
Ping Identity

Email: brian.d.campbell@gmail.com

John Bradley
Yubico

Email: ve7jtb@ve7jtb.com

Chuck Mortimore
Salesforce

Email: cmortimore@salesforce.com

OAuth Working Group
Internet-Draft
Intended status: Standards Track
Expires: September 12, 2017

N. Sakimura
Nomura Research Institute
K. Li
Alibaba Group
J. Bradley
Ping Identity
March 11, 2017

The OAuth 2.0 Authorization Framework: JWT Pop Token Usage
draft-sakimura-oauth-jpop-01

Abstract

This specification describes how to use JWT POP (Jpop) tokens that were obtained through [POPKD] in HTTP requests to access OAuth 2.0 protected resources. Only the party in possession of a corresponding cryptographic key with the Jpop token can use it to get access to the associated resources unlike in the case of the bearer token described in [RFC6750] where any party in possession of the access token can access the resource.

Requirements Language

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119 [RFC2119].

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 <http://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 September 12, 2017.

Copyright Notice

Copyright (c) 2017 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<http://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1. Introduction	3
1.1. Notational Conventions	3
2. Terms and definitions	3
3. JWT POP Token	3
4. Sender Constrained Token	4
4.1. CN Constrained Token	4
4.2. Client ID Constrained Token	5
5. Key Constrained Token	5
6. Resource access method	7
6.1. Mutual TLS access method	7
6.2. Signature method	8
7. Authorization Error	9
8. IANA Considerations	10
8.1. Jpop Authentication Scheme	10
8.2. JWT Confirmation Methods	10
9. Security Considerations	10
9.1. Certificate validation	10
9.2. Key protection	11
9.3. Audiance Restriction	11
9.4. Dynamic client registration elements	11
10. Acknowledgements	11
11. References	11
11.1. Normative References	11
11.2. Informative References	13
Appendix A. Document History	13
Authors' Addresses	13

1. Introduction

This document specifies the method for the client to use a proof-of-possession token against a protected resource. The format of such token is defined in section 3 of [RFC7800].

1.1. Notational Conventions

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 RFC 2119 [RFC2119].

Unless otherwise noted, all the protocol parameter names and values are case sensitive.

2. Terms and definitions

For the purpose of this document, the terms defined in [RFC6749] and [RFC7800] are used.

3. JWT POP Token

JWT PoP token is a JWS signed JWT whose payload is a JWT Claims Set. The JWT claims set MUST include the following:

iss The issuer identifier of the authorization server.

aud The identifier of the resource server.

iat The issuance time of this token.

exp The expiry time of this token.

cnf The confirmation method.

Their semantics are defined in [RFC7519] and [RFC7800].

Following is an example of such.

```
{
  "iss": "https://server.example.com",
  "aud": "https://resource.example.org",
  "iat": "1360189224",
  "exp": "1361398868",
  "cnf": {...}
}
```

Figure 1: Example of JWT PoP Token.

4. Sender Constrained Token

There are several varieties of sender constrained token. Namely:

1. CN Constrained Token
2. Client ID Constrained Token

4.1. CN Constrained Token

CN constrained token is typically used when X.509 client certificate authentication is used at the token endpoint. In this case, the constraint is expressed by including the following member at the top level of cnf claim.

cn The Common Name of the client certificate that the client used in the authorization request.

The authorization server finds the relevant CN from the X.509 client certificate authentication that is performed at the token endpoint.

```
{
  "iss": "https://server.example.com",
  "sub": "joe@example.com",
  "aud": "https://resource.example.org",
  "exp": "1361398824",
  "nbf": "1360189224",
  "cnf": {
    "cn": "client.example.com"
  }
}
```

Figure 2: Example of CN Constrained JWT.

4.2. Client ID Constrained Token

The constraint in the Client ID constrained token is expressed by including the following member at the top level of cnf claim.

cid The client_id of the client that the client used in the authorization request. The combination of the "iss" of the access token and this value forms a globally unique identifier for the client.

The authorization server finds the client ID from the client ID used in the client authentication at the token endpoint.

5. Key Constrained Token

Methods to express key constraints are extensively described in the section 3 of [RFC7800]. Such cnf claim is used in the access token described in section 3 to form a key constrained token. [RFC7800] defines 4 confirmation methods.

jwk JSON Web Key Representing a Public Key

jwe Encrypted JSON Web Key

jwt#s256 [RFC7638] Thumbprint of a JWK using the SHA-256 hash function.

x5t#s256 [RFC7515] X.509 Certificate SHA-256 Thumbprint

jku JWK Set URL

Following is an example of a JWT payload containing a JWK with a raw key.

```
{
  "iss": "https://server.example.com",
  "sub": "joe@example.com",
  "aud": "https://resource.example.org",
  "exp": "1361398824",
  "nbf": "1360189224",
  "cnf": {
    "jwk": {
      "kty": "EC",
      "use": "sig",
      "crv": "P-256",
      "x": "18wHLeIgW9wVN6VD1Txgpqy2LszYkMf6J8njVAibvhM",
      "y": "-V4dS4UaLMgP_4fY4j8ir7cl1TXlFdAgcx55o7TkcSA"
    }
  }
}
```

Figure 3: Example of a JWK Key Constrained JWT.

Following is an example of a JWT payload containing a jku URI.

```
{
  "iss": "https://server.example.com",
  "sub": "joe@example.com",
  "aud": "https://resource.example.org",
  "exp": "1361398824",
  "nbf": "1360189224",
  "cnf": {
    "jku": "https://client.example.com/keys/client123-jwks"
  }
}
```

Figure 4: Example of a jku Constrained JWT.

Following is an example of a JWT payload containing a x5t#s256 Certificate Thumbprint of a x509 certificate. .

```
{
  "iss": "https://server.example.com",
  "sub": "joe@example.com",
  "aud": "https://resource.example.org",
  "exp": "1361398824",
  "nbf": "1360189224",
  "cnf": {
    "x5t#s256": "w5cK0ebwmCZUYDB2Y5SlESsXE8o9yZg05O89jdNidgI"
  }
}
```

Figure 5: Example of a x5t#s256 Certificate Thumbprint Constrained JWT.

6. Resource access method

The resource server that supports this specification **MUST** authenticate the Client by having it demonstrate that it is the holder of the key associated with the access token being used. The confirmation method can be broadly categorized in two forms.

Mutual TLS method A method leveraging on the X.509 client certificate authentication of the TLS connection. `cn`, `x5t#s256`, and `jku` confirmation methods can be used with this access method. (The JWKS referenced by the `jku` **MUST** contain JWK with `x5c` certificate elements for this access method)

Signature method A method leveraging the signature on the nonce. `cid`, `jku`, `jwk`, `jwe`, and, `jwt#S256` confirmation methods can be used with this access method.

6.1. Mutual TLS access method

CN cnf method Under this method, X.509 client certificate authentication at the resource endpoint is being leveraged. The resource endpoint **MUST** obtain the CN of the client certificate used for the authentication and **MUST** verify that the value of the `cn` member in the `cnf` member matches with it.

If it does not match, the process stops here and the resource access **MUST** be denied.

If it is valid, then the resource server **MUST** verify the access token. If it is valid, the resource **SHOULD** be returned as HTTP response.

x5t#s256 cnf method Under this method, X.509 client certificate authentication at the resource endpoint is being leveraged. The resource endpoint MUST obtain the client certificate used for the authentication and MUST verify that the base64url-encoded SHA-256 thumbprint of the DER encoded X.509 client certificate. The **x5t#s256** member in the **cnf** member MUST exactly match the calculated thumbprint.

If it does not match, the process stops here and the resource access MUST be denied.

If it is valid, then the resource server MUST verify the access token. If it is valid, the resource SHOULD be returned as HTTP response.

jku cnf method Under this method, X.509 client certificate authentication at the resource endpoint is being leveraged. The resource endpoint MUST obtain the client certificate used for the authentication and MUST verify that the certificate matches one of the **x5c** elements retrieved from the [RFC7517]JWKS. Each **x5c** element may contain a chain of base64-encoded certificates. The client certificate MUST only be compared with the last certificate in the chain.

If it does not match, the process stops here and the resource access MUST be denied.

If it is valid, then the resource server MUST verify the access token. If it is valid, the resource SHOULD be returned as HTTP response.

6.2. Signature method

For this, the following steps are taken:

1. The client prepares a nonce.
2. The client creates JWS compact serialization over the nonce.

To obtain it, first create a JSON with a name "nonce" and the value being what was received in the previous step. The JWS MUST contain a kid header element if the client has more than one signing key published via JWKS URI e.g.,

```
{
  "nonce": "dcd98b7102dd2f0e8b11d0f600bfb0c093"
}
```

Then, "jws-on-nonce" is obtained by creating a compact serialization of JWS on this JSON.

3. The client sends the request to the resource server, this time with Authorization Request Header as defined in section 4.2 of [RFC7235] with the credential as follows:

```
credentials      = "Jpop" jpop-response
jpop-response    = at-response "," s-response
at-response      = "at" "=" access-token; As specified by [POPKD]
s-response       = "s" "=" jws-on-nonce; Created in the step 3.
access-token     = quoted-string
jws-on-nonce     = quoted-string
```

In the following example, the access token and the jws-on-nonce are represented as access.token.jwt and jws.of.nonce for the sake of brevity.

```
GET /resource/1234 HTTP/1.0
Host: server.example.com
Authorization: Jpop at="access.token.jwt", s="jws.of.nonce"
```

Figure 6: Example resource request

4. The resource server finds the client's public key from the access token through the methods described in [RFC7800].

5. The resource server MUST verify the value of "s" of the Authorization header. If it fails, the process stops here and the resource access MUST be denied.

6. The resource server MUST verify the access token. If it is valid, the resource SHOULD be returned as HTTP response.

7. Authorization Error

If the client requests the resource without the proper authentication header, the resource server returns a HTTP 401 response with "WWW-Authenticate" header as defined in section 4.1 of [RFC7235] with the challenge as follows:

```
challenge        = "Jpop" jpop-challenge
jpop-challenge    = "nonce" "=" nonce-value
nonce-value       = quoted-string
```

Following example depicts what the response would look like.

```
HTTP/1.0 401 Unauthorized
Server: HTTPd/0.9
Date: Wed, 14 March 2017 09:26:53 GMT
WWW-Authenticate: Jpop nonce="dcd98b7102dd2f0e8b11d0f600bfb0c093"
```

Figure 7: Example error response.

8. IANA Considerations

8.1. Jpop Authentication Scheme

A new scheme has been registered in the HTTP Authentication Scheme Registry as follows:

Authentication Scheme Name: Jpop

Reference: Section 3 of this specification

Notes (optional): The Named Authentication scheme is intended to be used only with OAuth Resource Access, and thus does not support proxy authentication.

8.2. JWT Confirmation Methods

- o Confirmation Method Value: "cn"
- o Confirmation Method Description: CN match with the TLS client auth.
- o Change Controller: IESG
- o Specification Document(s): This document.
- o Confirmation Method Value: "cid"
- o Confirmation Method Description: Client ID Confirmation
- o Change Controller: IESG
- o Specification Document(s): This document.

9. Security Considerations

9.1. Certificate validation

The "cn" JWT confirmation method relies its security property on the X.509 client certificate authentication. In particular, the validity of the certificate needs to be verified properly. It involves the

traversal of all the certificate chain and the certificate validation (e.g., with OCSP).

9.2. Key protection

The client's secret key must be kept securely. Otherwise, the notion of PoP breaks down.

It should be noted that JWE confirmation method is significantly weaker form of the PoP, as the resource server and the authorization server can masquerade as the client.

9.3. Audience Restriction

When using the signature method the client must specify to the AS the aud it intends to send the token to, so that it can be included in the AT.

A malicious RS could receive a AT with no aud or a logical audience and then replay the AT and jws-on-nonce to the actual server.

NOTE another approach would be to include the resource in the jws-on-nonce

9.4. Dynamic client registration elements

When a AS uses dynamic client registration it may accept software statements supplied by a federation operator. Those software statements can contain a JWKS-URI that is hosted by the federation operator or protected by a certificate provisioned from a trusted root. These methods would allow the federation operator to administratively revoke the keys at the JWKS-URI without requiring the JWKS to contain x5c elements with CA issued certificates and having to have the RS perform full certificate validation for each request.

10. Acknowledgements

The authors thank the following people for providing valuable feedback to this document. Nov Mataka (YAuth).

11. References

11.1. Normative References

- [POPKD] Bradley, J., Hunt, P., Jones, M., and H. Tschofenig, "OAuth 2.0 Proof-of-Possession: Authorization Server to Client Key Distribution", March 2017, <<https://tools.ietf.org/html/draft-ietf-oauth-pop-key-distribution-03>>.
- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <<http://www.rfc-editor.org/info/rfc2119>>.
- [RFC2617] Franks, J., Hallam-Baker, P., Hostetler, J., Lawrence, S., Leach, P., Luotonen, A., and L. Stewart, "HTTP Authentication: Basic and Digest Access Authentication", RFC 2617, DOI 10.17487/RFC2617, June 1999, <<http://www.rfc-editor.org/info/rfc2617>>.
- [RFC6749] Hardt, D., Ed., "The OAuth 2.0 Authorization Framework", RFC 6749, DOI 10.17487/RFC6749, October 2012, <<http://www.rfc-editor.org/info/rfc6749>>.
- [RFC6750] Jones, M. and D. Hardt, "The OAuth 2.0 Authorization Framework: Bearer Token Usage", RFC 6750, DOI 10.17487/RFC6750, October 2012, <<http://www.rfc-editor.org/info/rfc6750>>.
- [RFC7235] Fielding, R., Ed. and J. Reschke, Ed., "Hypertext Transfer Protocol (HTTP/1.1): Authentication", RFC 7235, DOI 10.17487/RFC7235, June 2014, <<http://www.rfc-editor.org/info/rfc7235>>.
- [RFC7515] Jones, M., Bradley, J., and N. Sakimura, "JSON Web Signature (JWS)", RFC 7515, DOI 10.17487/RFC7515, May 2015, <<http://www.rfc-editor.org/info/rfc7515>>.
- [RFC7517] Jones, M., "JSON Web Key (JWK)", RFC 7517, DOI 10.17487/RFC7517, May 2015, <<http://www.rfc-editor.org/info/rfc7517>>.
- [RFC7519] Jones, M., Bradley, J., and N. Sakimura, "JSON Web Token (JWT)", RFC 7519, DOI 10.17487/RFC7519, May 2015, <<http://www.rfc-editor.org/info/rfc7519>>.
- [RFC7638] Jones, M. and N. Sakimura, "JSON Web Key (JWK) Thumbprint", RFC 7638, DOI 10.17487/RFC7638, September 2015, <<http://www.rfc-editor.org/info/rfc7638>>.

- [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, <<http://www.rfc-editor.org/info/rfc7800>>.

11.2. Informative References

- [PKCE] Sakimura, N., "Proof Key for Code Exchange by OAuth Public Clients", July 2015.
- [POPA] Hunt, P., Ed., "OAuth 2.0 Proof-of-Possession (PoP) Security Architecture", March 2015, <<https://tools.ietf.org/html/draft-ietf-oauth-pop-architecture-08>>.
- [TINTRO] Richer, J., "OAuth 2.0 Token Introspection", July 2015.

Appendix A. Document History

-00 Initial Version.

Authors' Addresses

Nat Sakimura
Nomura Research Institute
Otemachi Financial City Grand Cube, 1-9-2 Otemachi
Chiyoda-ku, Tokyo 100-0004
Japan

Phone: +81-3-5533-2111
Email: n-sakimura@nri.co.jp
URI: <https://nat.sakimura.org/>

Kepeng Li
Alibaba Group

Email: kepeng.lkp@alibaba-inc.com

John Bradley
Ping Identity

Email: ve7jtb@ve7jtb.com
URI: <http://www.thread-safe.com/>

OAuth Working Group
Internet-Draft
Intended status: Standards Track
Expires: September 12, 2017

W. Denniss
Google
March 11, 2017

OAuth 2.0 Device Posture Signals
draft-wdenniss-oauth-device-posture-00

Abstract

Enterprise and security focused OAuth providers typically want additional signals to confirm user presence when users return to previously authorized apps. Rather than requiring a full reauthentication, or require enrollment in a mobile device management solution, some authorization servers may be willing to accept device posture signals from the app, like the fact that device has a lock screen, as confirmation of user presence. This document details how OAuth native app clients can communicate device posture signals to OAuth providers.

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 <http://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 September 12, 2017.

Copyright Notice

Copyright (c) 2017 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<http://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect

to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1. Introduction	2
2. Notational Conventions	3
3. Device Posture Signal Dictionary	3
4. Authorization Request Device Posture Hint	3
5. Token Endpoint Device Posture Enforcement	4
6. Security Considerations	5
6.1. Device Posture Scope	5
6.2. Spoofed Devices	5
6.3. App Trustworthiness	5
7. IANA Considerations	5
7.1. OAuth Parameters Registration	5
7.1.1. Registry Contents	6
7.2. OAuth Extensions Error Registration	6
7.2.1. Registry Contents	6
7.3. Device Posture Keys Registry	6
7.3.1. Registration Template	7
7.3.2. Initial Registry Contents	7
8. References	8
8.1. Normative References	8
8.2. Informative References	8
Appendix A. Acknowledgements	8
Author's Address	9

1. Introduction

Users who follow strong security practices on their devices - such as configuring screen locks, and not enabling admin privileges (commonly known as "rooting" or "jailbreaking") - shouldn't need to reauthenticate frequently to the individual apps on their device.

This specification details how apps can send device posture signals to the OAuth Token Endpoint, enabling it to enforce device policy compliance, and avoid the need for reauthentication in some cases.

It is designed to provide a mechanism for honest apps to communicate device posture. By itself it doesn't protect against malicious users, dishonest apps, or compromised devices, but the signal format described could carry signals that do.

2. Notational Conventions

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 Key words for use in RFCs to Indicate Requirement Levels [RFC2119]. If these words are used without being spelled in uppercase then they are to be interpreted with their normal natural language meanings.

3. Device Posture Signal Dictionary

The device posture is a dictionary of signals asserted by the app about the device. The structure is sent as an added parameter in several places during the OAuth flow, as documented in the subsequent sections.

All device posture keys are OPTIONAL and MUST only be set when the attribute can be obtained by the app. The standard attribute keys are as follows:

screen_lock

Boolean. True if the user has a screen lock, such as a pin, pattern biometric, etc.

root_privileges

Boolean. True if user apps can access root device privileges. For mobile operating systems, known as "jailbreaking" on iOS and "rooting" on Android.

device_attestation

Dictionary. An attestation from the operating system, containing a signed-statement about the device and/or the app. The format is a dictionary, the specifics of which depends on the operating system.

An example device posture dictionary:

```
{
  "screen_lock": true,
  "root_privileges": false
}
```

4. Authorization Request Device Posture Hint

Clients MAY send the device posture signal dictionary to the authorization server in the authorization request. These signals, except for those that are signed and bound to the device are susceptible to client-side modification by end-users. While untrusted, such signals can still be used as hints by the

authorization server to present a better user experience, like informing the user they need a lock screen.

Error encountered during authorization can be displayed to the user in the browser making this a more user friendly way to instruct the user on how to move their device into conformance. The token endpoint (on which errors are less user-friendly as there's no user agent), can then enforce the restrictions per Section 5.

The following parameters are added to the OAuth 2.0 Authorization Request:

`device_posture_hint`

JSON String. URL-encoded JSON dictionary, contains the Device Posture Signals defined in Section 3.

5. Token Endpoint Device Posture Enforcement

Clients that follow this specification **MUST** send the device posture signals on every request to the token endpoint.

Token Endpoints **SHOULD** verify that the posture conforms to their requirements and act accordingly.

The following parameters are added to all requests to the Token Endpoint:

`device_posture`

JSON String. URL-encoded JSON dictionary, contains the Device Posture Signals defined in Section 3.

The app **MUST** obtain fresh device posture information before every request to the Token Endpoint, and **MUST NOT** include stale information (rather, it should drop any signals it cannot freshly obtain).

For token refresh requests, where the device posture has been previously communicated, if an attribute is missing, the Token Endpoint may choose to use the previous value, based on it's own policy and freshness requirements.

If the policy does not meet requirements, the Token Endpoint **SHOULD** return the following error code:

`device_posture_invalid`

Error indicating that the device posture does not meet requirements. The error description **SHOULD** contain details on why this is is the case.

6. Security Considerations

6.1. Device Posture Scope

This specification is designed to help authorization servers enforce security policy (like requiring a lock screen) on end-users. The intent is to enforce restrictions on honest users, to force them to follow security practices set out by the authorization server. By itself, it offers no protection against malicious users, dishonest apps, or compromised devices.

Combined with other technologies like device-based attestations and token binding may enable such protection, and this specification could be used to transmit secure signals, but that topic is out of scope for this specification.

6.2. Spoofed Devices

It is possible to at a device level completely spoof the device posture. Even statements signed by the operating system are vulnerable to spoofing, as it's possible a statement from the real device can be replayed on a spoofed device, unless such statements include a binding to the device itself. Per Section 6.1, this topic is out of scope for this specification.

6.3. App Trustworthiness

This specification is designed to allow trusted apps to report device posture to the authorization server to help the server enforce security policy on end-users. It does not by itself force apps to be honest, or genuine. Genuine apps (i.e. apps not lying about their client ID) might be dishonest about the device posture, and apps that are normally honest, could be spoofed, unless anti-spoofing countermeasures that are out of scope of this specification are employed.

7. IANA Considerations

7.1. OAuth Parameters Registration

This specification registers the following value in the IANA "OAuth Parameters" registry [IANA.OAuth.Parameters] established by [RFC6749].

7.1.1.1. Registry Contents

- o Parameter name: `device_posture_hint`
- o Parameter usage location: authorization request
- o Change controller: IESG
- o Specification document(s): Section 4 of [[this specification]]

- o Parameter name: `device_posture`
- o Parameter usage location: token request
- o Change controller: IESG
- o Specification document(s): Section 5 of [[this specification]]

7.2. OAuth Extensions Error Registration

This specification registers the following error in the IANA "OAuth Extensions Error Registry" [IANA.OAuth.Parameters] established by [RFC6749].

7.2.1. Registry Contents

- o Error name: `device_posture_invalid`
- o Error usage location: authorization response, token error response
- o Related protocol extension: resource parameter
- o Change controller: IESG
- o Specification document(s): Section 5 of [[this specification]]

7.3. Device Posture Keys Registry

This specification establishes the IANA "Device Posture Keys" registry for Device Posture Dictionary keys. The registry records the Device Posture key and a reference to the specification that defines it. This specification registers the Device Posture keys defined in Section 3.

Keys are registered on an Expert Review [RFC5226] basis after a three-week review period on the `oauth-reg-review@ietf.org` mailing list, on the advice of one or more Designated Experts.

Registration requests sent to the mailing list for review should use an appropriate subject (e.g., "Request to register Device Posture Key: `screen_lock`").

Within the review period, the Designated Experts will either approve or deny the registration request, communicating this decision to the review list and IANA. Denials should include an explanation and, if applicable, suggestions as to how to make the request successful. Registration requests that are undetermined for a period longer than

21 days can be brought to the IESG's attention (using the iesg@ietf.org mailing list) for resolution.

Criteria that should be applied by the Designated Experts includes determining whether the proposed registration duplicates existing functionality, whether it is likely to be of general applicability or whether it is useful only for a single application, whether the value is actually being used, and whether the registration description is clear.

IANA must only accept registry updates from the Designated Experts and should direct all requests for registration to the review mailing list.

It is suggested that the same Designated Experts evaluate these registration requests as those who evaluate registration requests for the IANA "OAuth Parameters" registry [IANA.OAuth.Parameters].

7.3.1. Registration Template

Device Posture Signal Key:

The key name requested (e.g., "screen_lock"). 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.

Device Posture Signal Key Description:

Brief description of the device posture signal (e.g., "Screen lock active").

Change Controller:

For Standards Track RFCs, state "IESG". 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.

7.3.2. Initial Registry Contents

- o Device Posture Signal Key: "screen_lock"
- o Device Posture Signal Key Description: Boolean. 'true' when the device has a screen lock enabled.
- o Change Controller: IESG
- o Specification Document(s): Section 3 of [[this specification]]

- o Device Posture Signal Key: "root_privileges"

- o Device Posture Signal Key Description: Boolean. True if user apps can access root device privileges.
- o Change Controller: IESG
- o Specification Document(s): Section 3 of [[this specification]]

- o Device Posture Signal Key: "device_attestation"
- o Device Posture Signal Key Description: Dictionary. An attestation from the operating system, containing a signed-statement about the device and/or the app.
- o Change Controller: IESG
- o Specification Document(s): Section 3 of [[this specification]]

8. References

8.1. Normative References

- [IANA.OAuth.Parameters]
IANA, "OAuth Parameters",
<<http://www.iana.org/assignments/oauth-parameters>>.
- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <<http://www.rfc-editor.org/info/rfc2119>>.
- [RFC5226] Narten, T. and H. Alvestrand, "Guidelines for Writing an IANA Considerations Section in RFCs", BCP 26, RFC 5226, DOI 10.17487/RFC5226, May 2008, <<http://www.rfc-editor.org/info/rfc5226>>.
- [RFC6749] Hardt, D., Ed., "The OAuth 2.0 Authorization Framework", RFC 6749, DOI 10.17487/RFC6749, October 2012, <<http://www.rfc-editor.org/info/rfc6749>>.

8.2. Informative References

- [RFC6819] Lodderstedt, T., Ed., McGloin, M., and P. Hunt, "OAuth 2.0 Threat Model and Security Considerations", RFC 6819, DOI 10.17487/RFC6819, January 2013, <<http://www.rfc-editor.org/info/rfc6819>>.

Appendix A. Acknowledgements

The following individuals contributed ideas, feedback, and wording that shaped and formed the final specification:

Eric Sachs, John Bradley, and Andy Zmolek.

Author's Address

William Denniss
Google
1600 Amphitheatre Pkwy
Mountain View, CA 94043
USA

Phone: +1 650-253-0000
Email: wdenniss@google.com
URI: <http://google.com/>