

OAuth Working Group
Internet-Draft
Intended status: Standards Track
Expires: October 10, 2021

N. Sakimura
NAT.Consulting
J. Bradley
Yubico
M. Jones
Microsoft
April 8, 2021

The OAuth 2.0 Authorization Framework: JWT Secured Authorization Request
(JAR)
draft-ietf-oauth-jwsreq-34

Abstract

The authorization request in OAuth 2.0 described in RFC 6749 utilizes query parameter serialization, which means that Authorization Request parameters are encoded in the URI of the request and sent through user agents such as web browsers. While it is easy to implement, it means that (a) the communication through the user agents is not integrity protected and thus the parameters can be tainted, (b) the source of the communication is not authenticated, and (c) the communication through the user agents can be monitored. Because of these weaknesses, several attacks to the protocol have now been put forward.

This document introduces the ability to send request parameters in a JSON Web Token (JWT) instead, which allows the request to be signed with JSON Web Signature (JWS) and encrypted with JSON Web Encryption (JWE) so that the integrity, source authentication and confidentiality property of the Authorization Request is attained. The request can be sent by value or by reference.

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 October 10, 2021.

Copyright Notice

Copyright (c) 2021 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 Language	5
2. Terminology	6
2.1. Request Object	6
2.2. Request Object URI	6
3. Symbols and abbreviated terms	6
4. Request Object	7
5. Authorization Request	9
5.1. Passing a Request Object by Value	10
5.2. Passing a Request Object by Reference	10
5.2.1. URI Referencing the Request Object	11
5.2.2. Request using the "request_uri" Request Parameter	12
5.2.3. Authorization Server Fetches Request Object	12
6. Validating JWT-Based Requests	13
6.1. JWE Encrypted Request Object	13
6.2. JWS Signed Request Object	13
6.3. Request Parameter Assembly and Validation	14
7. Authorization Server Response	14
8. TLS Requirements	14
9. IANA Considerations	15
9.1. OAuth Parameters Registration	15
9.2. OAuth Authorization Server Metadata Registry	16
9.3. OAuth Dynamic Client Registration Metadata Registry	17
9.4. Media Type Registration	17
9.4.1. Registry Contents	17
10. Security Considerations	18
10.1. Choice of Algorithms	18
10.2. Request Source Authentication	18
10.3. Explicit Endpoints	19

10.4.	Risks Associated with request_uri	19
10.4.1.	DDoS Attack on the Authorization Server	20
10.4.2.	Request URI Rewrite	20
10.5.	Downgrade Attack	20
10.6.	TLS Security Considerations	21
10.7.	Parameter Mismatches	21
10.8.	Cross-JWT Confusion	21
11.	Privacy Considerations	22
11.1.	Collection limitation	22
11.2.	Disclosure Limitation	23
11.2.1.	Request Disclosure	23
11.2.2.	Tracking using Request Object URI	23
12.	Acknowledgements	24
13.	Revision History	24
14.	References	32
14.1.	Normative References	32
14.2.	Informative References	34
	Authors' Addresses	35

1. Introduction

The Authorization Request in OAuth 2.0 [RFC6749] utilizes query parameter serialization and is typically sent through user agents such as web browsers.

For example, the parameters "response_type", "client_id", "state", and "redirect_uri" are encoded in the URI of the request:

```
GET /authorize?response_type=code&client_id=s6BhdRkqt3&state=xyz
&redirect_uri=https%3A%2F%2Fclient%2Eexample%2Ecom%2Fcb HTTP/1.1
Host: server.example.com
```

While it is easy to implement, the encoding in the URI does not allow application layer security to be used to provide confidentiality and integrity protection. While TLS is used to offer communication security between the Client and the user-agent as well as the user-agent and the Authorization Server, TLS sessions are terminated in the user-agent. In addition, TLS sessions may be terminated prematurely at some middlebox (such as a load balancer).

As the result, the Authorization Request of [RFC6749] has shortcomings in that:

- (a) the communication through the user agents is not integrity protected and thus the parameters can be tainted (integrity protection failure)

- (b) the source of the communication is not authenticated (source authentication failure)
- (c) the communication through the user agents can be monitored (containment / confidentiality failure).

Due to these inherent weaknesses, several attacks against the protocol, such as Redirection URI rewriting, have been identified.

The use of application layer security mitigates these issues.

The use of application layer security allows requests to be prepared by a trusted third party so that a client application cannot request more permissions than previously agreed.

Furthermore, passing the request by reference allows the reduction of over-the-wire overhead.

The JWT [RFC7519] encoding has been chosen because of

- (1) its close relationship with JSON, which is used as OAuth's response format
- (2) its developer friendliness due to its textual nature
- (3) its relative compactness compared to XML
- (4) its development status as a Proposed Standard, along with the associated signing and encryption methods [RFC7515] [RFC7516]
- (5) the relative ease of JWS and JWE compared to XML Signature and Encryption.

The parameters "request" and "request_uri" are introduced as additional authorization request parameters for the OAuth 2.0 [RFC6749] flows. The "request" parameter is a JSON Web Token (JWT) [RFC7519] whose JWT Claims Set holds the JSON encoded OAuth 2.0 authorization request parameters. Note that, in contrast to RFC 7519, the elements of the Claims Set are encoded OAuth Request Parameters [IANA.OAuth.Parameters], supplemented with only a few of the IANA-managed JSON Web Token Claims [IANA.JWT.Claims] - in particular "iss" and "aud". The JWT in the "request" parameter is integrity protected and source authenticated using JWS.

The JWT [RFC7519] can be passed to the authorization endpoint by reference, in which case the parameter "request_uri" is used instead of the "request".

Using JWT [RFC7519] as the request encoding instead of query parameters has several advantages:

- (a) (integrity protection) The request can be signed so that the integrity of the request can be checked.
- (b) (source authentication) The request can be signed so that the signer can be authenticated.
- (c) (confidentiality protection) The request can be encrypted so that end-to-end confidentiality can be provided even if the TLS connection is terminated at one point or another (including at and before user-agents).
- (d) (collection minimization) The request can be signed by a trusted third party attesting that the authorization request is compliant with a certain policy. For example, a request can be pre-examined by a trusted third party that all the personal data requested is strictly necessary to perform the process that the end-user asked for, and signed by that trusted third party. The authorization server then examines the signature and shows the conformance status to the end-user, who would have some assurance as to the legitimacy of the request when authorizing it. In some cases, it may even be desirable to skip the authorization dialogue under such circumstances.

There are a few cases that request by reference is useful such as:

1. When it is desirable to reduce the size of transmitted request. The use of application layer security increases the size of the request, particularly when public key cryptography is used.
2. When the client does not want to do the application level cryptography. The Authorization Server may provide an endpoint to accept the Authorization Request through direct communication with the Client so that the Client is authenticated and the channel is TLS protected.

This capability is in use by OpenID Connect [OpenID.Core].

1.1. Requirements Language

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

2. Terminology

For the purposes of this specification, the following terms and definitions in addition to what is defined in OAuth 2.0 Framework [RFC6749], JSON Web Signature [RFC7515], and JSON Web Encryption [RFC7519] apply.

2.1. Request Object

JSON Web Token (JWT) [RFC7519] whose JWT Claims Set holds the JSON encoded OAuth 2.0 authorization request parameters.

2.2. Request Object URI

Absolute URI that references the set of parameters comprising an OAuth 2.0 authorization request. The contents of the resource referenced by the URI are a Request Object (Section 2.1), unless the URI was provided to the client by the same Authorization Server, in which case the content is an implementation detail at the discretion of the Authorization Server. The former is to ensure interoperability in cases where the provider of the request_uri is a separate entity from the consumer, such as when a client provides a URI referencing a Request Object stored on the client's backend service and made accessible via HTTPS. In the latter case where the Authorization Server is both provider and consumer of the URI, such as when it offers an endpoint that provides a URI in exchange for a Request Object, this interoperability concern does not apply.

3. Symbols and abbreviated terms

The following abbreviations are common to this specification.

JSON JavaScript Object Notation

JWT JSON Web Token

JWS JSON Web Signature

JWE JSON Web Encryption

URI Uniform Resource Identifier

URL Uniform Resource Locator

4. Request Object

A Request Object (Section 2.1) is used to provide authorization request parameters for an OAuth 2.0 authorization request. It MUST contain all the parameters (including extension parameters) used to process the OAuth 2.0 [RFC6749] authorization request except the "request" and "request_uri" parameters that are defined in this document. The parameters are represented as the JWT claims of the object. Parameter names and string values MUST be included as JSON strings. Since Request Objects are handled across domains and potentially outside of a closed ecosystem, per section 8.1 of [RFC8259], these JSON strings MUST be encoded using UTF-8 [RFC3629]. Numerical values MUST be included as JSON numbers. It MAY include any extension parameters. This JSON [RFC8259] object constitutes the JWT Claims Set defined in JWT [RFC7519]. The JWT Claims Set is then signed or signed and encrypted.

To sign, JSON Web Signature (JWS) [RFC7515] is used. The result is a JWS signed JWT [RFC7519]. If signed, the Authorization Request Object SHOULD contain the Claims "iss" (issuer) and "aud" (audience) as members, with their semantics being the same as defined in the JWT [RFC7519] specification. The value of "aud" should be the value of the Authorization Server (AS) "issuer" as defined in RFC8414 [RFC8414].

To encrypt, JWE [RFC7516] is used. When both signature and encryption are being applied, the JWT MUST be signed then encrypted as described in Section 11.2 of [RFC7519]. The result is a Nested JWT, as defined in [RFC7519].

The client determines the algorithms used to sign and encrypt Request Objects. The algorithms chosen need to be supported by both the client and the authorization server. The client can inform the authorization server of the algorithms that it supports in its dynamic client registration metadata [RFC7591], specifically, the metadata values "request_object_signing_alg", "request_object_encryption_alg", and "request_object_encryption_enc". Likewise, the authorization server can inform the client of the algorithms that it supports in its authorization server metadata [RFC8414], specifically, the metadata values "request_object_signing_alg_values_supported", "request_object_encryption_alg_values_supported", and "request_object_encryption_enc_values_supported".

The Request Object MAY be sent by value as described in Section 5.1 or by reference as described in Section 5.2. "request" and "request_uri" parameters MUST NOT be included in Request Objects.

A Request Object (Section 2.1) has the media type [RFC2046] "application/oauth-authz-req+jwt". Note that some existing deployments may alternatively be using the type "application/jwt".

The following is an example of the Claims in a Request Object before base64url [RFC7515] encoding and signing. Note that it includes the extension parameters "nonce" and "max_age".

```
{
  "iss": "s6BhdRkqt3",
  "aud": "https://server.example.com",
  "response_type": "code id_token",
  "client_id": "s6BhdRkqt3",
  "redirect_uri": "https://client.example.org/cb",
  "scope": "openid",
  "state": "af0ifjsldkj",
  "nonce": "n-0S6_WzA2Mj",
  "max_age": 86400
}
```

Signing it with the "RS256" algorithm [RFC7518] results in this Request Object value (with line wraps within values for display purposes only):

```
eyYhbGciOiJSUzI1NiIsImtpZCI6ImSyYmRjIn0.ewogICAgImIzcyI6ICJzNkJoZmFJrcXQzIiwKICAgICJhdWQiOiAiaHR0cHM6Ly9zZXJ2ZXIuZXhhbXBsZS5jb20iLAogICAgInJlc3BvbmlX3R5cGUOiAiY29kZSBpZF90b2t1biIsCiAgICAiY2xpZW50X2lkIjogInM2QmhhUmtxdDMLAogICAgInJlZGlyZWNOX3VyaSI6ICJodHRwczovL2NsYWVudC5leGFtcGxlLm9yZy9jYiIsCiAgICAIic2NvcGUOiAib3BlbmklIiwKICAgICJzdGF0ZSI6ICJhZjBpZmpzbGRraiiIsCiAgICAIbm9uY2UiOiAibi0wUzZfV3pBMklqIiwKICAgICJtYXhfYWdlIjogODY0MDAKfQ.Nsxa_18VUElVaPjqW_ToI1yrEJ67Bgkb5xsuZRVqzGkfKrOIX7BCx0biSxYGmjK9KJPctH1OC0iQJWxu5YVY-vnW0_PLJb1C2HG-ztVzcnKpZcgle4i0vgQcpkH00CpW3SEYXnyWnKzuKzqSblwAZALo5f89B_p6QA6j6JwBSRvdVsDPdulw8lKxGTbh82CpCaQ50rLAg3EYLYaCb41k4I1zGXE4fvim9FIMs80CmmzwIB5S-ujfFzwFjoyuPEV4hJnoVUMXR_W9tppPf846lGwA8h9G9oNTIuX8ft2jfpnZdFmLq3_wr3Wa5q3a-lfbqF3S9H_8nN3jli7tLR_5Nz-q
```


The following RSA public key, represented in JWK format, can be used to validate the Request Object signature in this and subsequent Request Object examples (with line wraps within values for display purposes only):

```
{
  "kty": "RSA",
  "kid": "k2bdc",
  "n": "x5RbkAZkmpRxia65qRQ1wwSMSxQUnS7gcpVTV_cdHmfmg21td2yabEO9XadD8
    pJNZubINPpmgHh3J1aD9WRwS05ucmFq3CfFsluLt13_7oX5yDRSKX7poXmT_5
    ko8k4NJZPMAO8fPToDTH7kHYbONSE2FYa5GZ60CUsFhSonI-dcMDJ0Ary91xI
    w5k2z4TAdARVWcs7sD07VhlMMshrwsPHBQgTatlkxyIHxbYdtak8fqvNAwr7O
    lVEvM_Ipf5OfmdB8Sd-wjzaBsyP4VhJKoi_qdgSzpC694XZeYPq45Sw-q51iF
    Ulc0lTCI7z6jltUtnR6ySn6XDGFnzH5Fe5ypw",
  "e": "AQAB"
}
```

5. Authorization Request

The client constructs the authorization request URI by adding the following parameters to the query component of the authorization endpoint URI using the "application/x-www-form-urlencoded" format:

request

REQUIRED unless "request_uri" is specified. The Request Object (Section 2.1) that holds authorization request parameters stated in section 4 of OAuth 2.0 [RFC6749]. If this parameter is present in the authorization request, "request_uri" MUST NOT be present.

request_uri

REQUIRED unless "request" is specified. The absolute URI as defined by RFC3986 [RFC3986] that is the Request Object URI (Section 2.2) referencing the authorization request parameters stated in section 4 of OAuth 2.0 [RFC6749]. If this parameter is present in the authorization request, "request" MUST NOT be present.

client_id

REQUIRED. OAuth 2.0 [RFC6749] "client_id". The value MUST match the "request" or "request_uri" Request Object's (Section 2.1) "client_id".

The client directs the resource owner to the constructed URI using an HTTP redirection response, or by other means available to it via the user-agent.

For example, the client directs the end user's user-agent to make the following HTTPS request:

```
GET /authz?client_id=s6BhdRkqt3&request=eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXZW40In0= HTTP/1.1
Host: server.example.com
```

The value for the request parameter is abbreviated for brevity.

The authorization request object MUST be one of the following:

- (a) JWS signed
- (b) JWS signed and JWE encrypted

The client MAY send the parameters included in the request object duplicated in the query parameters as well for the backward compatibility etc. However, the authorization server supporting this specification MUST only use the parameters included in the request object.

5.1. Passing a Request Object by Value

The Client sends the Authorization Request as a Request Object to the Authorization Endpoint as the "request" parameter value.

The following is an example of an Authorization Request using the "request" parameter (with line wraps within values for display purposes only):

```
https://server.example.com/authorize?client_id=s6BhdRkqt3&
request=eyJhbGciOiJSUzI1NiIsImtpZCI6Im9ybmRjIn0.ewogICAgImVzcyI6
ICJzNkJoZjZJrcXQzIiwKIICAgICJhdWQiOiAiAiaHR0cHM6Ly9zZXJ2ZXIuZXhhbXBs
ZS5jb20iLAogICAgInJlc3BvbmlX3R5cGUiOiAiY29kZSBpZF90b2t1biIsCiAg
ICAiY2xpZW50X2lkIjogInM2QmhhkUmtxdDMLAogICAgInJlZGlyZWNOX3VyaSI6
ICJodHRwczovL2NsaWVudC5leGFtcGxlLm9yZy9jYiIsCiAgICAic2NvcGUiOiAi
b3BlbmklIiwKIICAgICJzdGF0ZSI6ICJhZjBpZmpzbGRraiIsCiAgICAibm9uY2Ui
OiAibi0wUzZfV3pBMklqIiwKIICAgICJtYXhfYWdlIjogODY0MDAKfQ.Nsxa_18VU
ElVaPjqW_ToIlyrEJ67BgKb5xsuZRVqzGkfKrOIX7BCx0biSxYGmjK9KJPctH1OC
0iQJwXu5YVY-vnW0_PLJb1C2HG-ztVzcnKZC2gE4i0vgQcpkUOCpW3SEYXnyWnKz
uKzqSblwAZALo5f89B_p6QA6j6JwBSRvdVsDPdulW8lKxGTbH82czCaQ50rLAg3E
YLyaCb4ik4I1zGXE4fvim9FIMs8OCmzwIB5S-uJfFzwFjoyuPEV4hJnoVUmXR_W
9t9pPf846lGwA8h9G9oNTIuX8Ft2jfpnZdFmLg3_wr3Wa5q3a-lfbgF3S9H_8nN3
jli7tLR_5Nz-q
```

5.2. Passing a Request Object by Reference

The "request_uri" Authorization Request parameter enables OAuth authorization requests to be passed by reference, rather than by value. This parameter is used identically to the "request" parameter, other than that the Request Object value is retrieved from

the resource identified by the specified URI rather than passed by value.

The entire Request URI SHOULD NOT exceed 512 ASCII characters. There are two reasons for this restriction:

1. Many phones in the market as of this writing still do not accept large payloads. The restriction is typically either 512 or 1024 ASCII characters.
2. On a slow connection such as 2G mobile connection, a large URL would cause the slow response and therefore the use of such is not advisable from the user experience point of view.

The contents of the resource referenced by the "request_uri" MUST be a Request Object and MUST be reachable by the Authorization Server unless the URI was provided to the client by the Authorization Server. In the first case, the "request_uri" MUST be an "https" URI, as specified in Section 2.7.2 of RFC7230 [RFC7230]. In the second case, it MUST be a URN, as specified in RFC8141 [RFC8141].

The following is an example of the contents of a Request Object resource that can be referenced by a "request_uri" (with line wraps within values for display purposes only):

```
eyJhbGciOiJSUzI1NiIsImtpZCI6Im9yZy9jYiIsCiAgICAic2NvcGUiOiAib3BlbmklIiwKICAgICJzdGF0ZSI6ICJhZjBpZmpzbGRraiiIsCiAgICAibm9uY2UiOiAibi0wUzZfV3pBMklqIiwKICAgICJtYXhfYWdlIjogODY0MDAKfQ.Nsxa_18VUElVaPjqW_ToIlyrEJ67BgKb5xsuZRVqzGkfKrOIX7BCx0biSxYGmjK9KJPctH1OC0iQJwXu5YVY-vnW0_PLJb1C2HG-ztVzcnKZC2gE4i0vgQcpkUOCpW3SEYXnyWnKzuKzqSblwAZALo5f89B_p6QA6j6JwBSRvdVsDPdulW8lKxGTbH82czCaQ50rLag3EYLYaCb4ik4I1zGXE4fvim9FIMs80CMmzwIB5S-ujFfzwFjoyuPEV4hJnoVUmXR_W9typPf846lGwA8h9G9oNTIuX8Ft2jfpnZdFmLg3_wr3Wa5q3a-lfbgF3S9H_8nN3j1i7tLR_5Nz-g
```

5.2.1. URI Referencing the Request Object

The Client stores the Request Object resource either locally or remotely at a URI the Authorization Server can access. Such facility may be provided by the authorization server or a trusted third party. For example, the authorization server may provide a URL to which the client POSTs the request object and obtains the Request URI. This URI is the Request Object URI, "request_uri".

It is possible for the Request Object to include values that are to be revealed only to the Authorization Server. As such, the "request_uri" MUST have appropriate entropy for its lifetime so that the URI is not guessable if publicly retrievable. For the guidance, refer to 5.1.4.2.2 of [RFC6819] and Good Practices for Capability URLs [CapURLs]. It is RECOMMENDED that it be removed after a reasonable timeout unless access control measures are taken.

The following is an example of a Request Object URI value (with line wraps within values for display purposes only). In this example, a trusted third-party service hosts the Request Object.

```
https://tfp.example.org/request.jwt/  
GkurKxf5T0Y-mnPFCHqWOMiZi4VS138cQO_V7PZHAdM
```

5.2.2. Request using the "request_uri" Request Parameter

The Client sends the Authorization Request to the Authorization Endpoint.

The following is an example of an Authorization Request using the "request_uri" parameter (with line wraps within values for display purposes only):

```
https://server.example.com/authorize?  
client_id=s6BhdRkqt3  
&request_uri=https%3A%2F%2Ftfp.example.org%2Frequest.jwt  
%2FGkurKxf5T0Y-mnPFCHqWOMiZi4VS138cQO_V7PZHAdM
```

5.2.3. Authorization Server Fetches Request Object

Upon receipt of the Request, the Authorization Server MUST send an HTTP "GET" request to the "request_uri" to retrieve the referenced Request Object, unless it is stored in a way so that it can retrieve it through other mechanism securely, and parse it to recreate the Authorization Request parameters.

The following is an example of this fetch process. In this example, a trusted third-party service hosts the Request Object.

```
GET /request.jwt/GkurKxf5T0Y-mnPFCHqWOMiZi4VS138cQO_V7PZHAdM HTTP/1.1  
Host: tfp.example.org
```

The following is an example of the fetch response:

```
HTTP/1.1 200 OK
Date: Thu, 20 Aug 2020 23:52:39 GMT
Server: Apache/2.4.43 (tfp.example.org)
Content-type: application/oauth-authz-req+jwt
Content-Length: 797
Last-Modified: Wed, 19 Aug 2020 23:52:32 GMT

eyJhbGciOiJSUzI1NiIsImtpZCI6Im9yYmRjIn0.ewogICAgImIzcyI6ICJzNkJoZFJrcXQzIiwKICAgICJhdWQiOiAiaHR0cHM6Ly9zZXJ2ZXIuZXhhbXBsZS5jb20iLAogICAgInJlc3Bvb3R5cGUiOiAiY29kZSBpZF90b2t1biIsCiAgICAiY2xpZW50X2lkIjogInM2QmhkUmtxdDMiLAogICAgInJlZGlyZWNOX3VyaSI6ICJodHRwczovL2NsYWVudC5leGFtcGxlLm9yZy9jYiIsCiAgICAic2NvcGUiOiAib3BlbmlkIiwKICAgICJzdGF0ZSI6ICJhZjBpZmpzbGRraiiIsCiAgICAibm9uY2UiOiAibi0wUzZfV3pBMk1qIiwKICAgICJtYXhfYWdlIjogODY0MDAKfQ.Nsxa_18VUELvaPjqW_ToIlyrEJ67BgKb5xsuZRVqzGkfKrOIX7BCx0biSxYGmJk9KJPctH1OC0iQJwXu5YVY-vnW0_PLJb1C2HG-ztVzcnKZC2gE4i0vgQcpkUOCpW3SEYXnyWnKzuKzqSblwAZALo5f89B_p6QA6j6JwBSRvdVsDPdulW8lKxGTbH82czCaQ50rLAg3EYLYaCb4ik4I1zGXE4fvim9FIMS8OCMmzwIB5S-ujFfzwFjoyuPEV4hJnoVUmXR_W9typPf846lGwA8h9G9oNTIuX8Ft2jfpnZdFmLg3_wr3Wa5q3a-lfbgF3S9H_8nN3j1i7tLR_5Nz-g
```

6. Validating JWT-Based Requests

6.1. JWE Encrypted Request Object

If the request object is encrypted, the Authorization Server MUST decrypt the JWT in accordance with the JSON Web Encryption [RFC7516] specification.

The result is a signed request object.

If decryption fails, the Authorization Server MUST return an "invalid_request_object" error to the client in response to the authorization request.

6.2. JWS Signed Request Object

The Authorization Server MUST validate the signature of the JSON Web Signature [RFC7515] signed Request Object. If a "kid" Header Parameter is present, the key identified MUST be the key used, and MUST be a key associated with the client. The signature MUST be validated using a key associated with the client and the algorithm specified in the "alg" Header Parameter. Algorithm verification MUST be performed, as specified in Sections 3.1 and 3.2 of [RFC8725].

If the key is not associated with the client or if signature validation fails, the Authorization Server MUST return an

"invalid_request_object" error to the client in response to the authorization request.

6.3. Request Parameter Assembly and Validation

The Authorization Server MUST extract the set of Authorization Request parameters from the Request Object value. The Authorization Server MUST only use the parameters in the Request Object even if the same parameter is provided in the query parameter. The Client ID values in the "client_id" request parameter and in the Request Object "client_id" claim MUST be identical. The Authorization Server then validates the request as specified in OAuth 2.0 [RFC6749].

If the Client ID check or the request validation fails, then the Authorization Server MUST return an error to the client in response to the authorization request, as specified in Section 5.2 of OAuth 2.0 [RFC6749].

7. Authorization Server Response

Authorization Server Response is created and sent to the client as in Section 4 of OAuth 2.0 [RFC6749].

In addition, this document uses these additional error values:

invalid_request_uri

The "request_uri" in the Authorization Request returns an error or contains invalid data.

invalid_request_object

The request parameter contains an invalid Request Object.

request_not_supported

The Authorization Server does not support the use of the "request" parameter.

request_uri_not_supported

The Authorization Server does not support the use of the "request_uri" parameter.

8. TLS Requirements

Client implementations supporting the Request Object URI method MUST support TLS following Recommendations for Secure Use of Transport Layer Security (TLS) and Datagram Transport Layer Security (DTLS) [BCP195].

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

HTTP clients MUST also verify the TLS server certificate, using DNS-ID [RFC6125], to avoid man-in-the-middle attacks. The rules and guidelines defined in [RFC6125] apply here, with the following considerations:

- o Support for DNS-ID identifier type (that is, the `dNSName` identity in the `subjectAltName` extension) is REQUIRED. Certification authorities which issue server certificates MUST support the DNS-ID identifier type, and the DNS-ID identifier type MUST be present in server certificates.
- o DNS names in server certificates MAY contain the wildcard character `"*"`.
- o Clients MUST NOT use CN-ID identifiers; a CN field may be present in the server certificate's subject name, but MUST NOT be used for authentication within the rules described in [BCP195].
- o SRV-ID and URI-ID as described in Section 6.5 of [RFC6125] MUST NOT be used for comparison.

9. IANA Considerations

9.1. OAuth Parameters Registration

Since the request object is a JWT, the core JWT claims cannot be used for any purpose in the request object other than for what JWT dictates. Thus, they need to be registered as OAuth Authorization Request parameters to avoid future OAuth extensions using them with different meanings.

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

- o Name: `"iss"`
- o Parameter Usage Location: authorization request
- o Change Controller: IETF
- o Specification Document(s): Section 4.1.1 of [RFC7519] and this document.
- o Name: `"sub"`
- o Parameter Usage Location: authorization request
- o Change Controller: IETF

- o Specification Document(s): Section 4.1.2 of [RFC7519] and this document.
- o Name: "aud"
- o Parameter Usage Location: authorization request
- o Change Controller: IETF
- o Specification Document(s): Section 4.1.3 of [RFC7519] and this document.
- o Name: "exp"
- o Parameter Usage Location: authorization request
- o Change Controller: IETF
- o Specification Document(s): Section 4.1.4 of [RFC7519] and this document.
- o Name: "nbf"
- o Parameter Usage Location: authorization request
- o Change Controller: IETF
- o Specification Document(s): Section 4.1.5 of [RFC7519] and this document.
- o Name: "iat"
- o Parameter Usage Location: authorization request
- o Change Controller: IETF
- o Specification Document(s): Section 4.1.6 of [RFC7519] and this document.
- o Name: "jti"
- o Parameter Usage Location: authorization request
- o Change Controller: IETF
- o Specification Document(s): Section 4.1.7 of [RFC7519] and this document.

9.2. OAuth Authorization Server Metadata Registry

This specification adds the following value to the "OAuth Authorization Server Metadata" registry [IANA.OAuth.Parameters] established by [RFC8414].

- o Metadata Name: "require_signed_request_object"
- o Metadata Description: Indicates where authorization request needs to be protected as Request Object and provided through either "request" or "request_uri" parameter.
- o Change Controller: IETF
- o Specification Document(s): Section 10.5 of this document.

9.3. OAuth Dynamic Client Registration Metadata Registry

This specification adds the following value to the "OAuth Dynamic Client Registration Metadata" registry [IANA.OAuth.Parameters] established by [RFC7591].

- o Metadata Name: "require_signed_request_object"
- o Metadata Description: Indicates where authorization request needs to be protected as Request Object and provided through either "request" or "request_uri" parameter.
- o Change Controller: IETF
- o Specification Document(s): Section 10.5 of this document.

9.4. Media Type Registration

9.4.1. Registry Contents

This section registers the "application/oauth-authz-req+jwt" media type [RFC2046] in the "Media Types" registry [IANA.MediaTypes] in the manner described in [RFC6838], which can be used to indicate that the content is a JWT containing Request Object claims.

- o Type name: application
- o Subtype name: oauth-authz-req+jwt
- o Required parameters: n/a
- o Optional parameters: n/a
- o Encoding considerations: binary; A Request Object is a JWT; JWT values are encoded as a series of base64url-encoded values (some of which may be the empty string) separated by period ('.') characters.
- o Security considerations: See Section 10 of [[this specification]]
- o Interoperability considerations: n/a
- o Published specification: Section 4 of [[this specification]]
- o Applications that use this media type: Applications that use Request Objects to make an OAuth 2.0 Authorization Request
- o Fragment identifier considerations: n/a
- o Additional information:

Magic number(s): n/a

File extension(s): n/a

Macintosh file type code(s): n/a

- o Person & email address to contact for further information:
Nat Sakimura, nat@nat.consulting
- o Intended usage: COMMON
- o Restrictions on usage: none
- o Author: Nat Sakimura, nat@nat.consulting

- o Change controller: IETF
- o Provisional registration? No

10. Security Considerations

In addition to the all the security considerations discussed in OAuth 2.0 [RFC6819], the security considerations in [RFC7515], [RFC7516], [RFC7518], and [RFC8725] need to be considered. Also, there are several academic papers such as [BASIN] that provide useful insight into the security properties of protocols like OAuth.

In consideration of the above, this document advises taking the following security considerations into account.

10.1. Choice of Algorithms

When sending the authorization request object through "request" parameter, it MUST either be signed using JWS [RFC7515] or signed then encrypted using JWS [RFC7515] and JWE [RFC7516] respectively, with then considered appropriate algorithms.

10.2. Request Source Authentication

The source of the Authorization Request MUST always be verified. There are several ways to do it:

- (a) Verifying the JWS Signature of the Request Object.
- (b) Verifying that the symmetric key for the JWE encryption is the correct one if the JWE is using symmetric encryption. Note however, that if public key encryption is used, no source authentication is enabled by the encryption, as any party can encrypt content to the public key.
- (c) Verifying the TLS Server Identity of the Request Object URI. In this case, the Authorization Server MUST know out-of-band that the Client uses Request Object URI and only the Client is covered by the TLS certificate. In general, it is not a reliable method.
- (d) When an Authorization Server implements a service that returns a Request Object URI in exchange for a Request Object, the Authorization Server MUST perform Client Authentication to accept the Request Object and bind the Client Identifier to the Request Object URI it is providing. It MUST validate the signature, per (a). Since Request Object URI can be replayed, the lifetime of the Request Object URI MUST be short and preferably one-time use. The entropy of the Request Object URI

MUST be sufficiently large. The adequate shortness of the validity and the entropy of the Request Object URI depends on the risk calculation based on the value of the resource being protected. A general guidance for the validity time would be less than a minute and the Request Object URI is to include a cryptographic random value of 128bit or more at the time of the writing of this specification.

- (e) When a trusted third-party service returns a Request Object URI in exchange for a Request Object, it MUST validate the signature, per (a). In addition, the Authorization Server MUST be trusted by the third-party service and MUST know out-of-band that the client is also trusted by it.

10.3. Explicit Endpoints

Although this specification does not require them, research such as [BASIN] points out that it is a good practice to explicitly state the intended interaction endpoints and the message position in the sequence in a tamper evident manner so that the intent of the initiator is unambiguous. The following endpoints defined in [RFC6749], [RFC6750], and [RFC8414] are RECOMMENDED by this specification to use this practice :

- (a) Protected Resources ("protected_resources")
- (b) Authorization Endpoint ("authorization_endpoint")
- (c) Redirection URI ("redirect_uri")
- (d) Token Endpoint ("token_endpoint")

Further, if dynamic discovery is used, then this practice also applies to the discovery related endpoints.

In [RFC6749], while Redirection URI is included in the Authorization Request, others are not. As a result, the same applies to Authorization Request Object.

10.4. Risks Associated with request_uri

The introduction of "request_uri" introduces several attack possibilities. Consult the security considerations in Section 7 of RFC3986 [RFC3986] for more information regarding risks associated with URIs.

10.4.1. DDoS Attack on the Authorization Server

A set of malicious client can launch a DoS attack to the authorization server by pointing the "request_uri" to a URI that returns extremely large content or extremely slow to respond. Under such an attack, the server may use up its resource and start failing.

Similarly, a malicious client can specify the "request_uri" value that itself points to an authorization request URI that uses "request_uri" to cause the recursive lookup.

To prevent such attack to succeed, the server should (a) check that the value of "request_uri" parameter does not point to an unexpected location, (b) check the media type of the response is "application/oauth-authz-req+jwt", (c) implement a time-out for obtaining the content of "request_uri", and (d) not perform recursive GET on the "request_uri".

10.4.2. Request URI Rewrite

The value of "request_uri" is not signed thus it can be tampered by Man-in-the-browser attacker. Several attack possibilities rise because of this, e.g., (a) attacker may create another file that the rewritten URI points to making it possible to request extra scope (b) attacker launches a DoS attack to a victim site by setting the value of "request_uri" to be that of the victim.

To prevent such attack to succeed, the server should (a) check that the value of "request_uri" parameter does not point to an unexpected location, (b) check the media type of the response is "application/oauth-authz-req+jwt", and (c) implement a time-out for obtaining the content of "request_uri".

10.5. Downgrade Attack

Unless the protocol used by client and the server is locked down to use OAuth JAR, it is possible for an attacker to use RFC6749 requests to bypass all the protection provided by this specification.

To prevent it, this specification defines a new client metadata and server metadata "require_signed_request_object" whose value is a boolean.

When the value of it as a client metadata is "true", then the server MUST reject the authorization request from the client that does not conform to this specification. It MUST also reject the request if the request object uses "alg":"none" when this client metadata value is "true". If omitted, the default value is "false".

When the value of it as a server metadata is "true", then the server MUST reject the authorization request from any client that does not conform to this specification. It MUST also reject the request if the request object uses "alg":"none" when this server metadata value is "true". If omitted, the default value is "false".

Note that even if "require_signed_request_object" metadata values are not present, the client MAY use signed request objects, provided that there are signing algorithms mutually supported by the client and the server. Use of signing algorithm metadata is described in Section 4.

10.6. TLS Security Considerations

Current security considerations can be found in Recommendations for Secure Use of TLS and DTLS [BCP195]. This supersedes the TLS version recommendations in OAuth 2.0 [RFC6749].

10.7. Parameter Mismatches

Given that OAuth parameter values are being sent in two different places, as normal OAuth parameters and as Request Object claims, implementations must guard against attacks that could use mismatching parameter values to obtain unintended outcomes. That is the reason that the two Client ID values MUST match, the reason that only the parameter values from the Request Object are to be used, and the reason that neither "request" nor "request_uri" can appear in a Request Object.

10.8. Cross-JWT Confusion

As described in Section 2.8 of [RFC8725], attackers may attempt to use a JWT issued for one purpose in a context that it was not intended for. The mitigations described for these attacks can be applied to Request Objects.

One way that an attacker might attempt to repurpose a Request Object is to try to use it as a client authentication JWT, as described in Section 2.2 of [RFC7523]. A simple way to prevent this is to never use the Client ID as the "sub" value in a Request Object.

Another way to prevent cross-JWT confusion is to use explicit typing, as described in Section 3.11 of [RFC8725]. One would explicitly type a Request Object by including a "typ" Header Parameter with the value "oauth-authz-req+jwt" (which is registered in Section 9.4.1. Note however, that requiring explicitly typed Requests Objects at existing authorization servers will break most existing deployments, as existing clients are already commonly using untyped Request Objects, especially with OpenID Connect [OpenID.Core]. However, requiring

explicit typing would be a good idea for new OAuth deployment profiles where compatibility with existing deployments is not a consideration.

Finally, yet another way to prevent cross-JWT confusion is to use a key management regime in which keys used to sign Request Objects are identifiably distinct from those used for other purposes. Then, if an adversary attempts to repurpose the Request Object in another context, a key mismatch will occur, thwarting the attack.

11. Privacy Considerations

When the Client is being granted access to a protected resource containing personal data, both the Client and the Authorization Server need to adhere to Privacy Principles. RFC 6973 Privacy Considerations for Internet Protocols [RFC6973] gives excellent guidance on the enhancement of protocol design and implementation. The provision listed in it should be followed.

Most of the provision would apply to The OAuth 2.0 Authorization Framework [RFC6749] and The OAuth 2.0 Authorization Framework: Bearer Token Usage [RFC6750] and are not specific to this specification. In what follows, only the specific provisions to this specification are noted.

11.1. Collection limitation

When the Client is being granted access to a protected resource containing personal data, the Client SHOULD limit the collection of personal data to that which is within the bounds of applicable law and strictly necessary for the specified purpose(s).

It is often hard for the user to find out if the personal data asked for is strictly necessary. A trusted third-party service can help the user by examining the Client request and comparing to the proposed processing by the Client and certifying the request. After the certification, the Client, when making an Authorization Request, can submit Authorization Request to the trusted third-party service to obtain the Request Object URI. This process is two steps:

- (1) (Certification Process) The trusted third-party service examines the business process of the client and determines what claims they need: This is the certification process. Once the client is certified, then they are issued a client credential to authenticate against to push request objects to the trusted third-party service to get the "request_uri".

- (2) (Translation Process) The client uses the client credential that it got to push the request object to the trusted third-party service to get the "request_uri". The trusted third-party service also verifies that the Request Object is consistent with the claims that the client is eligible for, per prior step.

Upon receiving such Request Object URI in the Authorization Request, the Authorization Server first verifies that the authority portion of the Request Object URI is a legitimate one for the trusted third-party service. Then, the Authorization Server issues HTTP GET request to the Request Object URI. Upon connecting, the Authorization Server MUST verify the server identity represented in the TLS certificate is legitimate for the Request Object URI. Then, the Authorization Server can obtain the Request Object, which includes the "client_id" representing the Client.

The Consent screen MUST indicate the Client and SHOULD indicate that the request has been vetted by the trusted third-party service for adherence to the Collection Limitation principle.

11.2. Disclosure Limitation

11.2.1. Request Disclosure

This specification allows extension parameters. These may include potentially sensitive information. Since URI query parameter may leak through various means but most notably through referrer and browser history, if the authorization request contains a potentially sensitive parameter, the Client SHOULD JWE [RFC7516] encrypt the request object.

Where Request Object URI method is being used, if the request object contains personally identifiable or sensitive information, the "request_uri" SHOULD be used only once, have a short validity period, and MUST have large enough entropy deemed necessary with applicable security policy unless the Request Object itself is JWE [RFC7516] Encrypted. The adequate shortness of the validity and the entropy of the Request Object URI depends on the risk calculation based on the value of the resource being protected. A general guidance for the validity time would be less than a minute and the Request Object URI is to include a cryptographic random value of 128bit or more at the time of the writing of this specification.

11.2.2. Tracking using Request Object URI

Even if the protected resource does not include a personally identifiable information, it is sometimes possible to identify the user through the Request Object URI if persistent static per-user

Request Object URIs are used. A third party may observe it through browser history etc. and start correlating the user's activity using it. In a way, it is a data disclosure as well and should be avoided.

Therefore, per-user persistent Request Object URIs should be avoided. Single-use Request Object URIs are one alternative.

12. Acknowledgements

The following people contributed to the creation of this document in the OAuth working group and other IETF roles. (Affiliations at the time of the contribution are used.)

Annabelle Backman (Amazon), Dirk Balfanz (Google), Sergey Beryozkin, Ben Campbell (as AD), Brian Campbell (Ping Identity), Roman Danyliw (as AD), Martin Duke (as AD), Vladimir Dzhuvinov (Connect2id), Lars Eggert (as AD), Joel Halpern (as GENART), Benjamin Kaduk (as AD), Stephen Kent (as SECDIR), Murray Kucherawy (as AD), Warren Kumari (as OPSDIR), Watson Ladd (as SECDIR), Torsten Lodderstedt (yes.com), Jim Manico, Axel Nennker (Deutsche Telecom), Hannes Tschofenig (ARM), James H. Manger (Telstra), Kathleen Moriarty (as AD), John Panzer (Google), Francesca Palombini (as AD), David Recordon (Facebook), Marius Scurtescu (Google), Luke Shepard (Facebook), Filip Skokan (Auth0), Eric Vyncke (as AD), and Robert Wilton (as AD).

The following people contributed to creating this document through the OpenID Connect Core 1.0 [OpenID.Core].

Brian Campbell (Ping Identity), George Fletcher (AOL), Ryo Itou (Mixi), Edmund Jay (Illumila), Breno de Medeiros (Google), Hideki Nara (TACT), Justin Richer (MITRE).

13. Revision History

Note to the RFC Editor: Please remove this section from the final RFC.

-34

- o Addressed additional IESG comments by Murray Kucherawy and Francesca Palombini.

-33

- o Addressed IESG comments prior to 8-Apr-21 telechat. Thanks to Martin Duke, Lars Eggert, Benjamin Kaduk, Francesca Palombini, and Eric Vyncke for their reviews.

-32

- o Removed outdated JSON reference.

-31

- o Addressed SecDir review comments by Watson Ladd.

-30

- o Changed the MIME Type from "oauth.authz.req+jwt" to "oauth-authz-req+jwt", per advice from the designated experts.

-29

- o Uniformly use the Change Controller "IETF".

-28

- o Removed unused references, as suggested by Roman Danyliw.

-27

- o Edits by Mike Jones to address IESG and working group review comments, including:
- o Added Security Considerations text saying not to use the Client ID as the "sub" value to prevent Cross-JWT Confusion.
- o Added Security Considerations text about using explicit typing to prevent Cross-JWT Confusion.
- o Addressed Eric Vyncke's review comments.
- o Addressed Robert Wilton's review comments.
- o Addressed Murray Kucherawy's review comments.
- o Addressed Benjamin Kaduk's review comments.
- o Applied spelling and grammar corrections.

-20

- o BK comments
- o Section 3 Removed WAP

- o Section 4. Clarified authorization request object parameters, removed extension parameters from examples
- o Section 4. Specifies application/oauth.authz.req+jwt as mime-type for request objects
- o Section 5.2.1 Added reference to Capability URLs
- o Section 5.2.3. Added entropy fragment to example request
- o Section 8. Replaced "subjectAltName dnsName" with "DNS-ID"
- o Section 9. Registers authorization request parameters in JWT Claims Registry.
- o Section 9. Registers application/oauth.authz.req in IANA mime-types registry
- o Section 10.1. Clarified encrypted request objects are "signed then encrypted" to maintain consistency
- o Section 10.2. Clarifies trust between AS and TFP
- o Section 10.3. Clarified endpoints subject to the practice
- o Section 10.4 Replaced "redirect_uri" to "request_uri"
- o Section 10.4. Added reference to RFC 3986 for risks
- o Section 10.4.1.d Deleted "do" to maintain grammar flow
- o Section 10.4.1, 10.4.2 Replaced "application/jose" to "application/jwt"
- o Section 12.1. Extended description for submitting authorization request to TFP to obtain request object
- o Section 12.2.2. Replaced per-user Request Object URI with static per-user Request URIs
- o Section 13. Combined OAuth WG contributors together
- o Section Whole doc Replaced application/jwt with application/oauth.authz.req+jwt

-19

- o AD comments

- o Section 5.2.1. s/Requiest URI/Request URI/
- o Section 8 s/[BCP195] ./[BCP195]./
- o Section 10.3. s/sited/cited/
- o Section 11. Typo. s/Curent/Current/

-17

- o #78 Typos in content-type

-16

- o Treated remaining Ben Campbell comments.

-15

- o Removed further duplication

-14

- o #71 Reiterate dynamic params are included.
- o #70 Made clear that AS must return error.
- o #69 Inconsistency of the need to sign.
- o Fixed Mime-type.
- o #67 Inconsistence in requiring HTTPS in request URI.
- o #66 Dropped ISO 29100 reference.
- o #25 Removed Encrypt only option.
- o #59 Same with #25.

-13

- o add TLS Security Consideration section
- o replace RFC7525 reference with BCP195
- o moved front tag in FETT reference to fix XML structure
- o changes reference from SoK to FETT

-12

- o fixes #62 - Alexey Melnikov Discuss
- o fixes #48 - OPSDIR Review : General - delete semicolons after list items
- o fixes #58 - DP Comments for the Last Call
- o fixes #57 - GENART - Remove "non-normative ... " from examples.
- o fixes #45 - OPSDIR Review : Introduction - are attacks discovered or already opened
- o fixes #49 - OPSDIR Review : Introduction - Inconsistent colons after initial sentence of list items.
- o fixes #53 - OPSDIR Review : 6.2 JWS Signed Request Object - Clarify JOSE Header
- o fixes #42 - OPSDIR Review : Introduction - readability of 'and' is confusing
- o fixes #50 - OPSDIR Review : Section 4 Request Object - Clarify 'signed, encrypted, or signed and encrypted'
- o fixes #39 - OPSDIR Review : Abstract - Explain/Clarify JWS and JWE
- o fixed #50 - OPSDIR Review : Section 4 Request Object - Clarify 'signed, encrypted, or signed and encrypted'
- o fixes #43 - OPSDIR Review : Introduction - 'properties' sounds awkward and are not exactly 'properties'
- o fixes #56 - OPSDIR Review : 12 Acknowledgements - 'contribution is' => 'contribution are'
- o fixes #55 - OPSDIR Review : 11.2.2 Privacy Considerations - ' It is in a way' => 'In a way, it is'
- o fixes #54 - OPSDIR Review : 11 Privacy Considerations - 'and not specific' => 'and are not specific'
- o fixes #51 - OPSDIR Review : Section 4 Request Object - 'It is fine' => 'It is recommended'
- o fixes #47 - OPSDIR Review : Introduction - 'over- the- wire' => 'over-the-wire'

- o fixes #46 - OPSDIR Review : Introduction - 'It allows' => 'The use of application security' for
- o fixes #44 - OPSDIR Review : Introduction - 'has' => 'have'
- o fixes #41 - OPSDIR Review : Introduction - missing 'is' before 'typically sent'
- o fixes #38 - OPSDIR Review : Section 11 - Delete 'freely accessible' regarding ISO 29100

-11

- o s/bing/being/
- o Added history for -10

-10

- o #20: KM1 -- some wording that is awkward in the TLS section.
- o #21: KM2 - the additional attacks against OAuth 2.0 should also have a pointer
- o #22: KM3 -- Nit: in the first line of 10.4:
- o #23: KM4 -- Mention RFC6973 in Section 11 in addition to ISO 29100
- o #24: SECDIR review: Section 4 -- Confusing requirements for sign+encrypt
- o #25: SECDIR review: Section 6 -- authentication and integrity need not be provided if the requestor encrypts the token?
- o #26: SECDIR Review: Section 10 -- why no reference for JWS algorithms?
- o #27: SECDIR Review: Section 10.2 - how to do the agreement between client and server "a priori"?
- o #28: SECDIR Review: Section 10.3 - Indication on "large entropy" and "short lifetime" should be indicated
- o #29: SECDIR Review: Section 10.3 - Typo
- o #30: SECDIR Review: Section 10.4 - typos and missing articles

- o #31: SECDIR Review: Section 10.4 - Clearer statement on the lack of endpoint identifiers needed
- o #32: SECDIR Review: Section 11 - ISO29100 needs to be moved to normative reference
- o #33: SECDIR Review: Section 11 - Better English and Entropy language needed
- o #34: Section 4: Typo
- o #35: More Acknowledgment
- o #36: DP - More precise qualification on Encryption needed.

-09

- o Minor Editorial Nits.
- o Section 10.4 added.
- o Explicit reference to Security consideration (10.2) added in section 5 and section 5.2.
- o , (add yourself) removed from the acknowledgment.

-08

- o Applied changes proposed by Hannes on 2016-06-29 on IETF OAuth list recorded as <https://bitbucket.org/Nat/oauth-jwsreq/issues/12/>.
- o TLS requirements added.
- o Security Consideration reinforced.
- o Privacy Consideration added.
- o Introduction improved.

-07

- o Changed the abbrev to OAuth JAR from oauth-jar.
- o Clarified sig and enc methods.
- o Better English.

- o Removed claims from one of the example.
- o Re-worded the URI construction.
- o Changed the example to use request instead of request_uri.
- o Clarified that Request Object parameters take precedence regardless of request or request_uri parameters were used.
- o Generalized the language in 4.2.1 to convey the intent more clearly.
- o Changed "Server" to "Authorization Server" as a clarification.
- o Stopped talking about request_object_signing_alg.
- o IANA considerations now reflect the current status.
- o Added Brian Campbell to the contributors list. Made the lists alphabetic order based on the last names. Clarified that the affiliation is at the time of the contribution.
- o Added "older versions of " to the reference to IE URI length limitations.
- o Stopped talking about signed or unsigned JWS etc.
- o 1.Introduction improved.

-06

- o Added explanation on the 512 chars URL restriction.
- o Updated Acknowledgements.

-05

- o More alignment with OpenID Connect.

-04

- o Fixed typos in examples. (request_url -> request_uri, cliend_id -> client_id)
- o Aligned the error messages with the OAuth IANA registry.
- o Added another rationale for having request object.

-03

- o Fixed the non-normative description about the advantage of static signature.
- o Changed the requirement for the parameter values in the request itself and the request object from 'MUST MATCH' to 'Req Obj takes precedence'.

-02

- o Now that they are RFCs, replaced JWS, JWE, etc. with RFC numbers.

-01

- o Copy Edits.

14. References

14.1. 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, May 2015.
- [IANA.MediaTypes] IANA, "Media Types", <<http://www.iana.org/assignments/media-types>>.
- [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>>.
- [RFC3629] Yergeau, F., "UTF-8, a transformation format of ISO 10646", STD 63, RFC 3629, DOI 10.17487/RFC3629, November 2003, <<https://www.rfc-editor.org/info/rfc3629>>.
- [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>>.

- [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>>.
- [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>>.
- [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>>.
- [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>>.
- [RFC7516] Jones, M. and J. Hildebrand, "JSON Web Encryption (JWE)", RFC 7516, DOI 10.17487/RFC7516, May 2015, <<https://www.rfc-editor.org/info/rfc7516>>.
- [RFC7518] Jones, M., "JSON Web Algorithms (JWA)", RFC 7518, DOI 10.17487/RFC7518, May 2015, <<https://www.rfc-editor.org/info/rfc7518>>.
- [RFC7519] Jones, M., Bradley, J., and N. Sakimura, "JSON Web Token (JWT)", RFC 7519, DOI 10.17487/RFC7519, May 2015, <<https://www.rfc-editor.org/info/rfc7519>>.
- [RFC8141] Saint-Andre, P. and J. Klensin, "Uniform Resource Names (URNs)", RFC 8141, DOI 10.17487/RFC8141, April 2017, <<https://www.rfc-editor.org/info/rfc8141>>.
- [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>>.
- [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>>.

14.2. Informative References

- [BASIN] Basin, D., Cremers, C., and S. Meier, "Provably Repairing the ISO/IEC 9798 Standard for Entity Authentication", Journal of Computer Security - Security and Trust Principles Volume 21 Issue 6, Pages 817-846, November 2013, <<https://www.cs.ox.ac.uk/people/cas.cremers/downloads/papers/BCM2012-iso9798.pdf>>.
- [CapURLs] Tennison, J., "Good Practices for Capability URLs", W3C Working Draft, February 2014, <<https://www.w3.org/TR/capability-urls/>>.
- [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.Core] Sakimura, N., Bradley, J., Jones, M., de Medeiros, B., and C. Mortimore, "OpenID Connect Core 1.0", OpenID Foundation Standards, February 2014, <http://openid.net/specs/openid-connect-core-1_0.html>.
- [RFC2046] Freed, N. and N. Borenstein, "Multipurpose Internet Mail Extensions (MIME) Part Two: Media Types", RFC 2046, DOI 10.17487/RFC2046, November 1996, <<https://www.rfc-editor.org/info/rfc2046>>.
- [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>>.

- [RFC6838] Freed, N., Klensin, J., and T. Hansen, "Media Type Specifications and Registration Procedures", BCP 13, RFC 6838, DOI 10.17487/RFC6838, January 2013, <<https://www.rfc-editor.org/info/rfc6838>>.
- [RFC6973] Cooper, A., Tschofenig, H., Aboba, B., Peterson, J., Morris, J., Hansen, M., and R. Smith, "Privacy Considerations for Internet Protocols", RFC 6973, DOI 10.17487/RFC6973, July 2013, <<https://www.rfc-editor.org/info/rfc6973>>.
- [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>>.
- [RFC8725] Sheffer, Y., Hardt, D., and M. Jones, "JSON Web Token Best Current Practices", BCP 225, RFC 8725, DOI 10.17487/RFC8725, February 2020, <<https://www.rfc-editor.org/info/rfc8725>>.

Authors' Addresses

Nat Sakimura
NAT.Consulting
2-22-17 Naka
Kunitachi, Tokyo 186-0004
Japan

Phone: +81-42-580-7401
Email: nat@nat.consulting
URI: <http://nat.sakimura.org/>

John Bradley
Yubico
Casilla 177, Sucursal Talagante
Talagante, RM
Chile

Phone: +1.202.630.5272
Email: ve7jtb@ve7jtb.com
URI: <http://www.thread-safe.com/>

Michael B. Jones
Microsoft
One Microsoft Way
Redmond, Washington 98052
United States of America

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

OAuth
Internet-Draft
Intended status: Informational
Expires: January 9, 2017

P. Hunt, Ed.
Oracle Corporation
J. Richer

W. Mills

P. Mishra
Oracle Corporation
H. Tschofenig
ARM Limited
July 8, 2016

OAuth 2.0 Proof-of-Possession (PoP) Security Architecture
draft-ietf-oauth-pop-architecture-08.txt

Abstract

The OAuth 2.0 bearer token specification, as defined in RFC 6750, allows any party in possession of a bearer token (a "bearer") to get access to the associated resources (without demonstrating possession of a cryptographic key). To prevent misuse, bearer tokens must be protected from disclosure in transit and at rest.

Some scenarios demand additional security protection whereby a client needs to demonstrate possession of cryptographic keying material when accessing a protected resource. This document motivates the development of the OAuth 2.0 proof-of-possession security mechanism.

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 January 9, 2017.

Copyright Notice

Copyright (c) 2016 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
2. Terminology	3
3. Use Cases	3
3.1. Preventing Access Token Re-Use by the Resource Server . .	4
3.2. TLS and DTLS Channel Binding Support	4
3.3. Access to a Non-TLS Protected Resource	4
3.4. Offering Application Layer End-to-End Security	5
4. Security and Privacy Threats	5
5. Requirements	6
6. Threat Mitigation	10
6.1. Confidentiality Protection	11
6.2. Sender Constraint	11
6.3. Key Confirmation	12
6.4. Summary	13
7. Architecture	14
7.1. Client and Authorization Server Interaction	15
7.1.1. Symmetric Keys	15
7.1.2. Asymmetric Keys	16
7.2. Client and Resource Server Interaction	17
7.3. Resource and Authorization Server Interaction (Token Introspection)	18
8. Security Considerations	19
9. IANA Considerations	19
10. Acknowledgments	19
11. References	20
11.1. Normative References	20
11.2. Informative References	21
Authors' Addresses	22

1. Introduction

The OAuth 2.0 protocol family ([RFC6749], [RFC6750], and [RFC6819]) offer a single token type known as the "bearer" token to access protected resources. RFC 6750 [RFC6750] specifies the bearer token mechanism and defines it as follows:

"A security token with the property that any party in possession of the token (a "bearer") can use the token in any way that any other party in possession of it can. Using a bearer token does not require a bearer to prove possession of cryptographic key material."

The bearer token meets the security needs of a number of use cases the OAuth 2.0 protocol had originally been designed for. There are, however, other scenarios that require stronger security properties and ask for active participation of the OAuth client in form of cryptographic computations when presenting an access token to a resource server.

This document outlines additional use cases requiring stronger security protection in Section 3, identifies threats in Section 4, proposes different ways to mitigate those threats in Section 6, outlines an architecture for a solution that builds on top of the existing OAuth 2.0 framework in Section 7, and concludes with a requirements list 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 [RFC2119], with the important qualification that, unless otherwise stated, these terms apply to the design of the protocol, not its implementation or application.

3. Use Cases

The main use case that motivates improvement upon "bearer" token security is the desire of resource servers to obtain additional assurance that the client is indeed authorized to present an access token. The expectation is that the use of additional credentials (symmetric or asymmetric keying material) will encourage developers to take additional precautions when transferring and storing access token in combination with these credentials.

Additional use cases listed below provide further requirements for the solution development. Note that a single solution does not necessarily need to offer support for all use cases.

3.1. Preventing Access Token Re-Use by the Resource Server

In a scenario where a resource server receives a valid access token, the resource server then re-uses it with other resource server. The reason for re-use may be malicious or may well be legitimate. In a legitimate case, the intent is to support chaining of computations whereby a resource server needs to consult other third party resource servers to complete a requested operation. In both cases it may be assumed that the scope and audience of the access token is sufficiently defined that to allow such a re-use. For example, imagine a case where a company operates email services as well as picture sharing services and that company had decided to issue access tokens with a scope and audience that allows access to both services.

With this use case the desire is to prevent such access token re-use. This also implies that the legitimate use cases require additional enhancements for request chaining.

3.2. TLS and DTLS Channel Binding Support

In this use case we consider the scenario where an OAuth 2.0 request to a protected resource is secured using TLS or DTLS (see [RFC4347]), but the client and the resource server demand that the underlying TLS/DTLS exchange is bound to additional application layer security to prevent cases where the TLS/DTLS connection is terminated at a TLS/DTLS intermediary, which splits the TLS/DTLS connection into two separate connections.

In this use case additional information should be conveyed to the resource server to ensure that no entity entity has tampered with the TLS/DTLS connection.

3.3. Access to a Non-TLS Protected Resource

This use case is for a web client that needs to access a resource that makes data available (such as videos) without offering integrity and confidentiality protection using TLS. Still, the initial resource request using OAuth, which includes the access token, must be protected against various threats (e.g., token replay, token modification).

While it is possible to utilize bearer tokens in this scenario with TLS protection when the request to the protected resource is made, as described in [RFC6750], there may be the desire to avoid using TLS

between the client and the resource server at all. In such a case the bearer token approach is not possible since it relies on TLS for ensuring integrity and confidentiality protection of the access token exchange since otherwise replay attacks are possible: First, an eavesdropper may steal an access token and present it at a different resource server. Second, an eavesdropper may steal an access token and replay it against the same resource server at a later point in time. In both cases, if the attack is successful, the adversary gets access to the resource owners data or may perform an operation selected by the adversary (e.g., sending a message). Note that the adversary may obtain the access token (if the recommendations in [RFC6749] and [RFC6750] are not followed) using a number of ways, including eavesdropping the communication on the wireless link.

Consequently, the important assumption in this use case is that a resource server does not have TLS support and the security solution should work in such a scenario. Furthermore, it may not be necessary to provide authentication of the resource server towards the client.

3.4. Offering Application Layer End-to-End Security

In Web deployments resource servers are often placed behind load balancers, which are deployed by the same organization that operates the resource servers. These load balancers may terminate the TLS connection setup and HTTP traffic is transmitted without TLS protection from the load balancer to the resource server. With application layer security in addition to the underlying TLS security it is possible to allow application servers to perform cryptographic verification on an end-to-end basis.

The key aspect in this use case is therefore to offer end-to-end security in the presence of load balancers via application layer security. Enterprise networks also deploy proxies that inspect traffic and thereby break TLS.

4. Security and Privacy Threats

The following list presents several common threats against protocols utilizing some form of token. This list of threats is based on NIST Special Publication 800-63 [NIST800-63]. We exclude a discussion of threats related to any form of identity proofing and authentication of the resource owner to the authorization server since these procedures are not part of the OAuth 2.0 protocol specification itself.

Token manufacture/modification:

An attacker may generate a bogus token or modify the token content (such as authentication or attribute statements) of an existing token, causing resource server to grant inappropriate access to the client. For example, an attacker may modify the token to extend the validity period. A client, which MAY be a normal client or MAY be assumed to be constrained (see [RFC7252]), may modify the token to have access to information that they should not be able to view.

Token disclosure:

Tokens may contain personal data, such as real name, age or birthday, payment information, etc.

Token redirect:

An attacker uses the token generated for consumption by the resource server to obtain access to another resource server.

Token reuse:

An attacker attempts to use a token that has already been used once with a resource server. The attacker may be an eavesdropper who observes the communication exchange or, worse, one of the communication end points. A client may, for example, leak access tokens because it cannot keep secrets confidential. A client may also reuse access tokens for some other resource servers. Finally, a resource server may use a token it had obtained from a client and use it with another resource server that the client interacts with. A resource server, offering relatively unimportant application services, may attempt to use an access token obtained from a client to access a high-value service, such as a payment service, on behalf of the client using the same access token.

Token repudiation:

Token repudiation refers to a property whereby a resource server is given an assurance that the authorization server cannot deny to have created a token for the client.

5. Requirements

RFC 4962 [RFC4962] gives useful guidelines for designers of authentication and key management protocols. While RFC 4962 was written with the AAA framework used for network access authentication in mind the offered suggestions are useful for the design of other key management systems as well. The following requirements list

applies OAuth 2.0 terminology to the requirements outlined in RFC 4962.

These requirements include

Cryptographic Algorithm Independent:

The key management protocol MUST be cryptographic algorithm independent.

Strong, fresh session keys:

Session keys MUST be strong and fresh. Each session deserves an independent session key, i.e., one that is generated specifically for the intended use. In context of OAuth this means that keying material is created in such a way that can only be used by the combination of a client instance, protected resource, and authorization scope.

Limit Key Scope:

Following the principle of least privilege, parties MUST NOT have access to keying material that is not needed to perform their role. Any protocol that is used to establish session keys MUST specify the scope for session keys, clearly identifying the parties to whom the session key is available.

Replay Detection Mechanism:

The key management protocol exchanges MUST be replay protected. Replay protection allows a protocol message recipient to discard any message that was recorded during a previous legitimate dialogue and presented as though it belonged to the current dialogue.

Authenticate All Parties:

Each party in the key management protocol MUST be authenticated to the other parties with whom they communicate. Authentication mechanisms MUST maintain the confidentiality of any secret values used in the authentication process. Secrets MUST NOT be sent to another party without confidentiality protection.

Authorization:

Client and resource server authorization MUST be performed. These entities MUST demonstrate possession of the appropriate keying material, without disclosing it. Authorization is REQUIRED

whenever a client interacts with an authorization server.
Authorization checking prevents an elevation of privilege attack.

Keying Material Confidentiality and Integrity:

While preserving algorithm independence, confidentiality and integrity of all keying material MUST be maintained.

Confirm Cryptographic Algorithm Selection:

The selection of the "best" cryptographic algorithms SHOULD be securely confirmed. The mechanism SHOULD detect attempted roll-back attacks.

Uniquely Named Keys:

Key management proposals require a robust key naming scheme, particularly where key caching is supported. The key name provides a way to refer to a key in a protocol so that it is clear to all parties which key is being referenced. Objects that cannot be named cannot be managed. All keys MUST be uniquely named, and the key name MUST NOT directly or indirectly disclose the keying material.

Prevent the Domino Effect:

Compromise of a single client MUST NOT compromise keying material held by any other client within the system, including session keys and long-term keys. Likewise, compromise of a single resource server MUST NOT compromise keying material held by any other Resource Server within the system. In the context of a key hierarchy, this means that the compromise of one node in the key hierarchy must not disclose the information necessary to compromise other branches in the key hierarchy. Obviously, the compromise of the root of the key hierarchy will compromise all of the keys; however, a compromise in one branch MUST NOT result in the compromise of other branches. There are many implications of this requirement; however, two implications deserve highlighting. First, the scope of the keying material must be defined and understood by all parties that communicate with a party that holds that keying material. Second, a party that holds keying material in a key hierarchy must not share that keying material with parties that are associated with other branches in the key hierarchy.

Bind Key to its Context:

Keying material MUST be bound to the appropriate context. The context includes the following.

- * The manner in which the keying material is expected to be used.
- * The other parties that are expected to have access to the keying material.
- * The expected lifetime of the keying material. Lifetime of a child key SHOULD NOT be greater than the lifetime of its parent in the key hierarchy.

Any party with legitimate access to keying material can determine its context. In addition, the protocol MUST ensure that all parties with legitimate access to keying material have the same context for the keying material. This requires that the parties are properly identified and authenticated, so that all of the parties that have access to the keying material can be determined. The context will include the client and the resource server identities in more than one form.

Authorization Restriction:

If client authorization is restricted, then the client SHOULD be made aware of the restriction.

Client Identity Confidentiality:

A client has identity confidentiality when any party other than the resource server and the authorization server cannot sufficiently identify the client within the anonymity set. In comparison to anonymity and pseudonymity, identity confidentiality is concerned with eavesdroppers and intermediaries. A key management protocol SHOULD provide this property.

Resource Owner Identity Confidentiality:

Resource servers SHOULD be prevented from knowing the real or pseudonymous identity of the resource owner, since the authorization server is the only entity involved in verifying the resource owner's identity.

Collusion:

Resource servers that collude can be prevented from using information related to the resource owner to track the individual. That is, two different resource servers can be prevented from determining that the same resource owner has authenticated to both

of them. Authorization servers MUST bind different keying material to access tokens used for resource servers from different origins (or similar concepts in the app world).

AS-to-RS Relationship Anonymity:

For solutions using asymmetric key cryptography the client MAY conceal information about the resource server it wants to interact with. The authorization server MAY reject such an attempt since it may not be able to enforce access control decisions.

Channel Binding:

A solution MUST enable support for channel bindings. The concept of channel binding, as defined in [RFC5056], allows applications to establish that the two end-points of a secure channel at one network layer are the same as at a higher layer by binding authentication at the higher layer to the channel at the lower layer.

There are performance concerns with the use of asymmetric cryptography. Although symmetric key cryptography offers better performance asymmetric cryptography offers additional security properties. A solution MUST therefore offer the capability to support both symmetric as well as asymmetric keys.

There are threats that relate to the experience of the software developer as well as operational practices. Verifying the servers identity in TLS is discussed at length in [RFC6125].

A number of the threats listed in Section 4 demand protection of the access token content and a standardized solution, for example, in the form of a JSON-based format, is available with the JWT [RFC7519].

6. Threat Mitigation

A large range of threats can be mitigated by protecting the content of the token, for example using a digital signature or a keyed message digest. Alternatively, the content of the token could be passed by reference rather than by value (requiring a separate message exchange to resolve the reference to the token content).

To simplify discussion in the following example we assume that the token itself cannot be modified by the client, either due to cryptographic protection (such as signature or encryption) or use of a reference value with sufficient entropy and associated secure lookup. The token remains opaque to the client. These are characteristics shared with bearer tokens and more information on

best practices can be found in [RFC6819] and in the security considerations section of [RFC6750].

To deal with token redirect it is important for the authorization server to include the identifier of the intended recipient - the resource server. A resource server must not be allowed to accept access tokens that are not meant for its consumption.

To provide protection against token disclosure two approaches are possible, namely (a) not to include sensitive information inside the token or (b) to ensure confidentiality protection. The latter approach requires at least the communication interaction between the client and the authorization server as well as the interaction between the client and the resource server to experience confidentiality protection. As an example, TLS with a ciphersuite that offers confidentiality protection has to be applied as per [RFC7525]. Encrypting the token content itself is another alternative. In our scenario the authorization server would, for example, encrypt the token content with a symmetric key shared with the resource server.

To deal with token reuse more choices are available.

6.1. Confidentiality Protection

In this approach confidentiality protection of the exchange is provided on the communication interfaces between the client and the resource server, and between the client and the authorization server. No eavesdropper on the wire is able to observe the token exchange. Consequently, a replay by a third party is not possible. An authorization server wants to ensure that it only hands out tokens to clients it has authenticated first and who are authorized. For this purpose, authentication of the client to the authorization server will be a requirement to ensure adequate protection against a range of attacks. This is, however, true for the description in Section 6.2 and Section 6.3 as well. Furthermore, the client has to make sure it does not distribute (or leak) the access token to entities other than the intended the resource server. For that purpose the client will have to authenticate the resource server before transmitting the access token.

6.2. Sender Constraint

Instead of providing confidentiality protection, the authorization server could also put the identifier of the client into the protected token with the following semantic: 'This token is only valid when presented by a client with the following identifier.' When the access token is then presented to the resource server how does it

know that it was provided by the client? It has to authenticate the client! There are many choices for authenticating the client to the resource server, for example by using client certificates in TLS [RFC5246], or pre-shared secrets within TLS [RFC4279]. The choice of the preferred authentication mechanism and credential type may depend on a number of factors, including

- o security properties
- o available infrastructure
- o library support
- o credential cost (financial)
- o performance
- o integration into the existing IT infrastructure
- o operational overhead for configuration and distribution of credentials

This long list hints to the challenge of selecting at least one mandatory-to-implement client authentication mechanism.

6.3. Key Confirmation

A variation of the mechanism of sender authentication, described in Section 6.2, is to replace authentication with the proof-of-possession of a specific (session) key, i.e., key confirmation. In this model the resource server would not authenticate the client itself but would rather verify whether the client knows the session key associated with a specific access token. Examples of this approach can be found with the OAuth 1.0 MAC token [RFC5849], and Kerberos [RFC4120] when utilizing the AP_REQ/AP_REP exchange (see also [I-D.hardjono-oauth-kerberos] for a comparison between Kerberos and OAuth).

To illustrate key confirmation, the first example is borrowed from Kerberos and use symmetric key cryptography. Assume that the authorization server shares a long-term secret with the resource server, called $K(\text{Authorization Server-Resource Server})$. This secret would be established between them out-of-band. When the client requests an access token the authorization server creates a fresh and unique session key K_s and places it into the token encrypted with the long term key $K(\text{Authorization Server-Resource Server})$. Additionally, the authorization server attaches K_s to the response message to the client (in addition to the access token itself) over a

confidentiality protected channel. When the client sends a request to the resource server it has to use K_s to compute a keyed message digest for the request (in whatever form or whatever layer). The resource server, when receiving the message, retrieves the access token, verifies it and extracts $K(\text{Authorization Server-Resource Server})$ to obtain K_s . This key K_s is then used to verify the keyed message digest of the request message.

Note that in this example one could imagine that the mechanism to protect the token itself is based on a symmetric key based mechanism to avoid any form of public key infrastructure but this aspect is not further elaborated in the scenario.

A similar mechanism can also be designed using asymmetric cryptography. When the client requests an access token the authorization server creates an ephemeral public / privacy key pair (PK/SK) and places the public key PK into the protected token. When the authorization server returns the access token to the client it also provides the PK/SK key pair over a confidentiality protected channel. When the client sends a request to the resource server it has to use the privacy key SK to sign the request. The resource server, when receiving the message, retrieves the access token, verifies it and extracts the public key PK. It uses this ephemeral public key to verify the attached signature.

6.4. Summary

As a high level message, there are various ways the threats can be mitigated. While the details of each solution are somewhat different, they all accomplish the goal of mitigating the threats.

The three approaches are:

Confidentiality Protection:

The weak point with this approach, which is briefly described in Section 6.1, is that the client has to be careful to whom it discloses the access token. What can be done with the token entirely depends on what rights the token entitles the presenter and what constraints it contains. A token could encode the identifier of the client but there are scenarios where the client is not authenticated to the resource server or where the identifier of the client rather represents an application class rather than a single application instance. As such, it is possible that certain deployments choose a rather liberal approach to security and that everyone who is in possession of the access token is granted access to the data.

Sender Constraint:

The weak point with this approach, which is briefly described in Section 6.2, is to setup the authentication infrastructure such that clients can be authenticated towards resource servers. Additionally, the authorization server must encode the identifier of the client in the token for later verification by the resource server. Depending on the chosen layer for providing client-side authentication there may be additional challenges due to Web server load balancing, lack of API access to identity information, etc.

Key Confirmation:

The weak point with this approach, see Section 6.3, is the increased complexity: a complete key distribution protocol has to be defined.

In all cases above it has to be ensured that the client is able to keep the credentials secret.

7. Architecture

The proof-of-possession security concept assumes that the authorization server acts as a trusted third party that binds keys to access tokens. These keys are then used by the client to demonstrate the possession of the secret to the resource server when accessing the resource. The resource server, when receiving an access token, needs to verify that the key used by the client matches the one included in the access token.

There are slight differences between the use of symmetric keys and asymmetric keys when they are bound to the access token and the subsequent interaction between the client and the authorization server when demonstrating possession of these keys. Figure 1 shows the symmetric key procedure and Figure 2 illustrates how asymmetric keys are used. While symmetric cryptography provides better performance properties the use of asymmetric cryptography allows the client to keep the private key locally and never expose it to any other party.

For example, with the JSON Web Token (JWT) [RFC7519] a standardized format for access tokens is available. The necessary elements to bind symmetric or asymmetric keys to a JWT are described in [I-D.ietf-oauth-proof-of-possession].

Note: The negotiation of cryptographic algorithms between the client and the authorization server is not shown in the examples below and

assumed to be present in a protocol solution to meet the requirements for crypto-agility.

7.1. Client and Authorization Server Interaction

7.1.1. Symmetric Keys

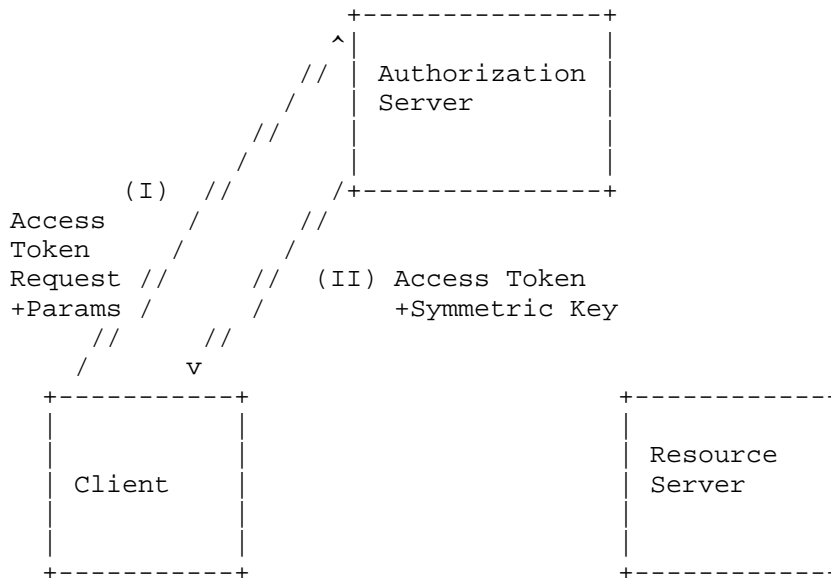


Figure 1: Interaction between the Client and the Authorization Server (Symmetric Keys).

In order to request an access token the client interacts with the authorization server as part of the a normal grant exchange, as shown in Figure 1. However, it needs to include additional information elements for use with the PoP security mechanism, as depicted in message (I). In message (II) the authorization server then returns the requested access token. In addition to the access token itself, the symmetric key is communicated to the client. This symmetric key is a unique and fresh session key with sufficient entropy for the given lifetime. Furthermore, information within the access token ties it to this specific symmetric key.

Note: For this security mechanism to work the client as well as the resource server need to have access to the session key. While the key transport mechanism from the authorization server to the client has been explained in the previous paragraph there are three ways for communicating this session key from the authorization server to the resource server, namely

Embedding the symmetric key inside the access token itself. This requires that the symmetric key is confidentiality protected.

The resource server queries the authorization server for the symmetric key. This is an approach envisioned by the token introspection endpoint [RFC7662].

The authorization server and the resource server both have access to the same back-end database. Smaller, tightly coupled systems might prefer such a deployment strategy.

7.1.2. Asymmetric Keys

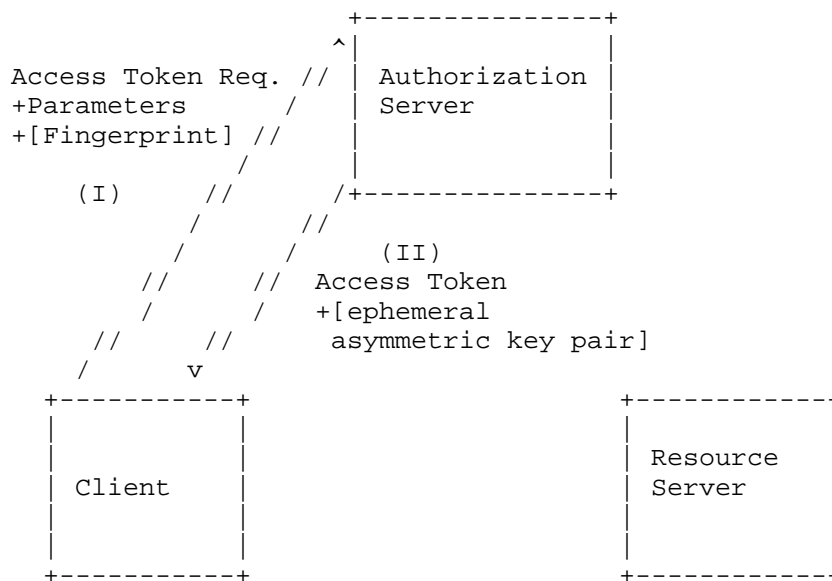


Figure 2: Interaction between the Client and the Authorization Server (Asymmetric Keys).

The use of asymmetric keys is slightly different since the client or the server could be involved in the generation of the ephemeral key pair. This exchange is shown in Figure 1. If the client generates the key pair it either includes a fingerprint of the public key or the public key in the request to the authorization server. The authorization server would include this fingerprint or public key in the confirmation claim inside the access token and thereby bind the asymmetric key pair to the token. If the client did not provide a fingerprint or a public key in the request then the authorization server is asked to create an ephemeral asymmetric key pair, binds the fingerprint of the public key to the access token, and returns the

asymmetric key pair (public and private key) to the client. Note that there is a strong preference for generating the private/public key pair locally at the client rather than at the server.

7.2. Client and Resource Server Interaction

The specification describing the interaction between the client and the authorization server, as shown in Figure 1 and in Figure 2, can be found in [I-D.ietf-oauth-pop-key-distribution].

Once the client has obtained the necessary access token and keying material it can start to interact with the resource server. To demonstrate possession of the key bound to the access token it needs to apply this key to the request by computing a keyed message digest (i.e., a symmetric key-based cryptographic primitive) or a digital signature (i.e., an asymmetric cryptographic computation). When the resource server receives the request it verifies it and decides whether access to the protected resource can be granted. This exchange is shown in Figure 3.

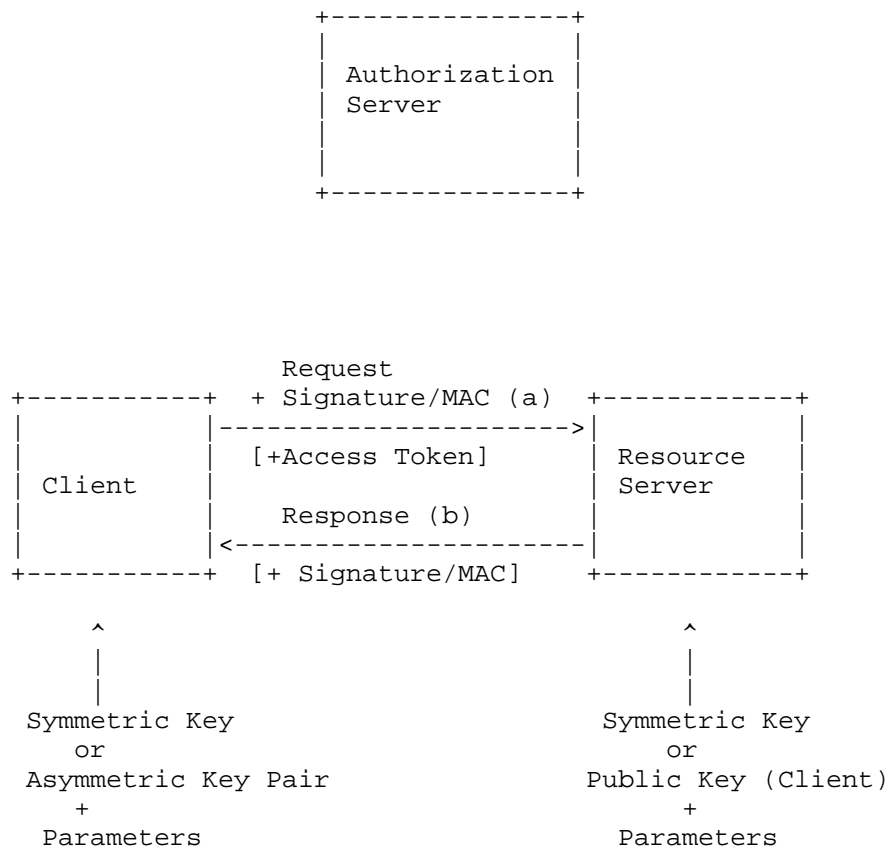


Figure 3: Client Demonstrates PoP.

The specification describing the ability to sign the HTTP request from the client to the resource server can be found in [I-D.ietf-oauth-signed-http-request].

7.3. Resource and Authorization Server Interaction (Token Introspection)

So far the examples talked about access tokens that are passed by value and allow the resource server to make authorization decisions immediately after verifying the request from the client. In some deployments a real-time interaction between the authorization server and the resource server is envisioned that lowers the need to pass self-contained access tokens around. In that case the access token merely serves as a handle or a reference to state stored at the authorization server. As a consequence, the resource server cannot autonomously make an authorization decision when receiving a request

from a client but has to consult the authorization server. This can, for example, be done using the token introspection endpoint (see [RFC7662]). Figure 4 shows the protocol interaction graphically. Despite the additional token exchange previous descriptions about associating symmetric and asymmetric keys to the access token are still applicable to this scenario.

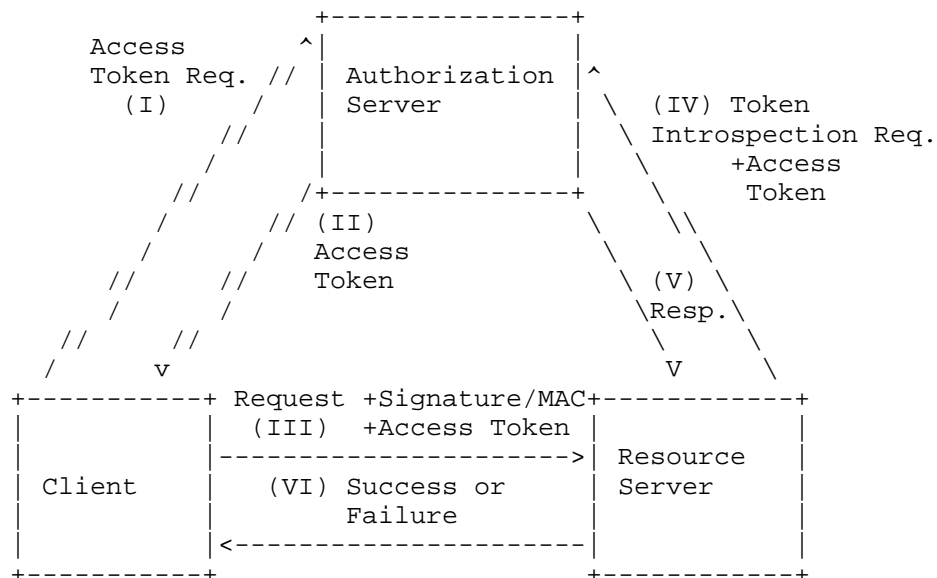


Figure 4: Token Introspection and Access Token Handles.

8. Security Considerations

The purpose of this document is to provide use cases, requirements, and motivation for developing an OAuth security solution extending Bearer Tokens. As such, this document is only about security.

9. IANA Considerations

This document does not require actions by IANA.

10. Acknowledgments

This document is the result of conference calls late 2012/early 2013 and in design team conference calls February 2013 of the IETF OAuth working group. The following persons (in addition to the OAuth WG chairs, Hannes Tschofenig, and Derek Atkins) provided their input during these calls: Bill Mills, Justin Richer, Phil Hunt, Prateek Mishra, Mike Jones, George Fletcher, Leif Johansson, Lucy Lynch, John

Bradley, Tony Nadalin, Klaas Wierenga, Thomas Hardjono, Brian Campbell

In the appendix of this document we reuse content from [RFC4962] and the authors would like thank Russ Housely and Bernard Aboba for their work on RFC 4962.

We would like to thank Reddy Tirumaleswar for his review.

11. References

11.1. Normative References

- [I-D.ietf-oauth-pop-key-distribution]
Bradley, J., Hunt, P., Jones, M., and H. Tschofenig, "OAuth 2.0 Proof-of-Possession: Authorization Server to Client Key Distribution", draft-ietf-oauth-pop-key-distribution-02 (work in progress), October 2015.
- [I-D.ietf-oauth-proof-of-possession]
Jones, M., Bradley, J., and H. Tschofenig, "Proof-of-Possession Key Semantics for JSON Web Tokens (JWTs)", draft-ietf-oauth-proof-of-possession-11 (work in progress), December 2015.
- [I-D.ietf-oauth-signed-http-request]
Richer, J., Bradley, J., and H. Tschofenig, "A Method for Signing HTTP Requests for OAuth", draft-ietf-oauth-signed-http-request-02 (work in progress), February 2016.
- [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>>.
- [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>>.
- [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>>.

- [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, <<http://www.rfc-editor.org/info/rfc7525>>.
- [RFC7662] Richer, J., Ed., "OAuth 2.0 Token Introspection", RFC 7662, DOI 10.17487/RFC7662, October 2015, <<http://www.rfc-editor.org/info/rfc7662>>.

11.2. Informative References

- [I-D.hardjono-oauth-kerberos]
Hardjono, T., "OAuth 2.0 support for the Kerberos V5 Authentication Protocol", draft-hardjono-oauth-kerberos-01 (work in progress), December 2010.
- [NIST800-63]
Burr, W., Dodson, D., Perlner, R., Polk, T., Gupta, S., and E. Nabbus, "NIST Special Publication 800-63-1, INFORMATION SECURITY", December 2008.
- [RFC4120] Neuman, C., Yu, T., Hartman, S., and K. Raeburn, "The Kerberos Network Authentication Service (V5)", RFC 4120, DOI 10.17487/RFC4120, July 2005, <<http://www.rfc-editor.org/info/rfc4120>>.
- [RFC4279] Eronen, P., Ed. and H. Tschofenig, Ed., "Pre-Shared Key Ciphersuites for Transport Layer Security (TLS)", RFC 4279, DOI 10.17487/RFC4279, December 2005, <<http://www.rfc-editor.org/info/rfc4279>>.
- [RFC4347] Rescorla, E. and N. Modadugu, "Datagram Transport Layer Security", RFC 4347, DOI 10.17487/RFC4347, April 2006, <<http://www.rfc-editor.org/info/rfc4347>>.
- [RFC4962] Housley, R. and B. Aboba, "Guidance for Authentication, Authorization, and Accounting (AAA) Key Management", BCP 132, RFC 4962, DOI 10.17487/RFC4962, July 2007, <<http://www.rfc-editor.org/info/rfc4962>>.
- [RFC5056] Williams, N., "On the Use of Channel Bindings to Secure Channels", RFC 5056, DOI 10.17487/RFC5056, November 2007, <<http://www.rfc-editor.org/info/rfc5056>>.
- [RFC5849] Hammer-Lahav, E., Ed., "The OAuth 1.0 Protocol", RFC 5849, DOI 10.17487/RFC5849, April 2010, <<http://www.rfc-editor.org/info/rfc5849>>.

- [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, <<http://www.rfc-editor.org/info/rfc6125>>.
- [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>>.
- [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>>.
- [RFC7252] Shelby, Z., Hartke, K., and C. Bormann, "The Constrained Application Protocol (CoAP)", RFC 7252, DOI 10.17487/RFC7252, June 2014, <<http://www.rfc-editor.org/info/rfc7252>>.

Authors' Addresses

Phil Hunt (editor)
Oracle Corporation

Email: phil.hunt@yahoo.com

Justin Richer

Email: ietf@justin.richer.org

William Mills

Email: wmills@yahoo-inc.com

Prateek Mishra
Oracle Corporation

Email: prateek.mishra@oracle.com

Hannes Tschofenig
ARM Limited
Hall in Tirol 6060
Austria

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

Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: September 28, 2019

J. Bradley
Ping Identity
P. Hunt
Oracle Corporation
M. Jones
Microsoft
H. Tschofenig
Arm Ltd.
M. Meszaros
GITDA
March 27, 2019

OAuth 2.0 Proof-of-Possession: Authorization Server to Client Key
Distribution
draft-ietf-oauth-pop-key-distribution-07

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.

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 28, 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	2
2. Terminology	4
3. Processing Instructions	4
4. Examples	5
4.1. Symmetric Key Transport	5
4.1.1. Client-to-AS Request	5
4.1.2. Client-to-AS Response	6
4.2. Asymmetric Key Transport	9
4.2.1. Client-to-AS Request	9
4.2.2. Client-to-AS Response	10
5. Security Considerations	11
6. IANA Considerations	13
6.1. OAuth Access Token Types	13
6.2. OAuth Parameters Registration	13
6.3. OAuth Extensions Error Registration	13
7. Acknowledgements	13
8. References	14
8.1. Normative References	14
8.2. Informative References	15
Authors' Addresses	16

1. Introduction

The work on proof-of-possession tokens, an extended token security mechanisms for OAuth 2.0, is motivated in [22]. This document defines the ability for the client request and to obtain PoP tokens from the authorization server. After successfully completing the exchange the client is in possession of a PoP token and the keying material bound to it. Clients that access protected resources then need to demonstrate knowledge of the secret key that is bound to the PoP 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). To demonstrate possession of the private/secret key to the resource server protocol mechanisms outside the scope of this document are used.

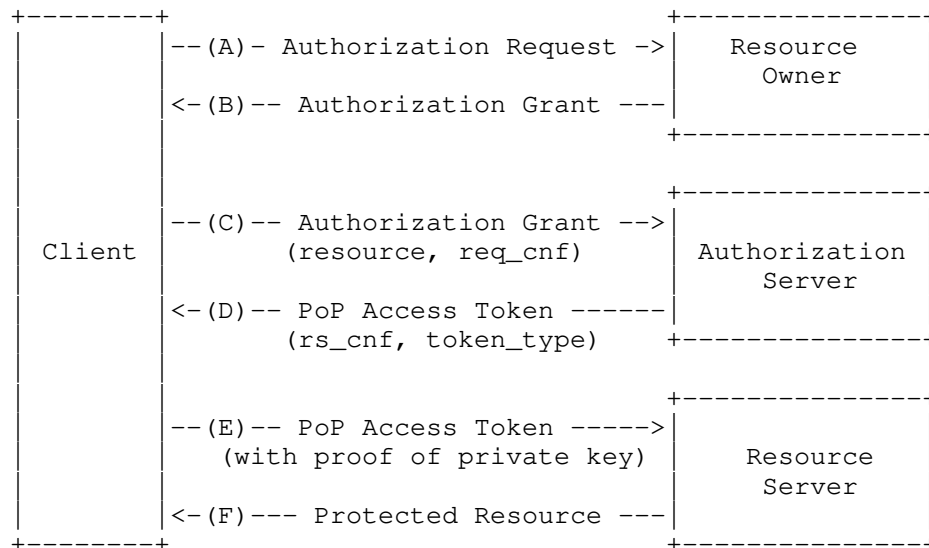


Figure 1: Augmented OAuth 2.0 Protocol Flow

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 [19] 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 specification 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 4.1 and the procedure for dealing with asymmetric keys is described in Section 4.2.

This document describes how the client requests and obtains a PoP access token from the authorization server for use with HTTPS-based

transport. The use of alternative transports, such as Constrained Application Protocol (CoAP), is described in [24].

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:

In the context of this specification '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[7]

JWT: JSON Web Token[9]

JWS: JSON Web Signature[6]

JWK: JSON Web Key[5]

JWE: JSON Web Encryption[8]

CWT: CBOR Web Token[13]

COSE: CBOR Object Signing and Encryption[14]

3. 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 use the resource identifier parameter, defined in [16], or the audience parameter, defined in [15], when symmetric PoP tokens are used. For asymmetric PoP tokens the use of resource indicators and audience is optional but

RECOMMENDED. The parameters 'audience' and 'resource' both allow the client to express the location of the target service and the difference between the two is described in [15]. As a summary, 'audience' allows expressing a logical name while 'resource' contains an absolute URI. More details about the 'resource' parameter can be found in [16].

Step (3): The authorization server parses the request from the server and determines the suitable response based on OAuth 2.0 and the PoP token credential procedures.

Note that PoP access tokens may be encoded in a variety of ways:

JWT The access token may be encoded using the JSON Web Token (JWT) format [9]. The proof-of-possession token functionality is described in [10]. A JWT encoded PoP token MUST be protected against modification by either using a digital signature or a keyed message digest, as described in [6]. The JWT may also be encrypted using [8].

CWT [13] defines an alternative token format based on CBOR. The proof-of-possession token functionality is defined in [12]. A CWT encoded PoP token MUST be protected against modification by either using a digital signature or a keyed message digest, as described in [12].

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 [23].

Note that the OAuth 2.0 framework nor this specification does not mandate a specific PoP token format but using a standardized format will improve interoperability and will lead to better code re-use.

Application layer interactions between the client and the resource server are beyond the scope of this document.

4. Examples

This section provides a number of examples.

4.1. Symmetric Key Transport

4.1.1. Client-to-AS Request

The client starts with a request to the authorization server indicating that it is interested to obtain a token for <https://resource.example.com>


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

grant_type=authorization_code
&code=Sp1xl0BeZQQYbYS6WxSbIA
&scope=calendar%20contacts
&redirect_uri=https%3A%2F%2Fclient%2Eexample%2Ecom%2Fcb
&resource=https%3A%2F%2Fresource.example.com
```

Example Request to the Authorization Server

4.1.2. 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.

Figure 2 shows a response containing a token and a "cnf" parameter with a symmetric proof-of-possession key both encoded in a JSON-based serialization format. The "cnf" parameter contains the RFC 7517 [5] encoded key element.

```
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",
  "cnf":{
    {"keys":
      [
        {"kty":"oct",
          "alg":"A128KW",
          "k":"GawgguFyGrWKav7AX4VKUg"
        }
      ]
    }
  }
}
```

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

Note that the cnf payload in Figure 2 is not encrypted at the application layer since Transport Layer Security is used between the AS and the client and the content of the cnf payload is consumed by the client itself. Alternatively, a JWE could be used to encrypt the key distribution, as shown in Figure 3.

```

{
  "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",
  "cnf":{
    "jwe":
      "eyJhbGciOiJSU0EtT0FFUCIsImVuYyI6IkJExMjhdQkMtSFMyNTYifQ.
      (remainder of JWE omitted for brevity)"
    }
  }
}

```

Figure 3: Example: Encrypted Symmetric Key

The content of the 'access_token' in JWT format contains the 'cnf' (confirmation) claim. 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 has to be present in a real deployment to mitigate a number of security threats.

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",
  "exp": 1311281970,
  "iat": 1311280970,
  "cnf":{
    "jwe":
      "eyJhbGciOiJSU0EtT0FFUCIsImVuYyI6IkJExMjhdQkMtSFMyNTYifQ.
      (remainder of 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 since content is meant for consumption by the selected resource server only. The details are described in [22].

4.2. Asymmetric Key Transport

4.2.1. Client-to-AS Request

This example illustrates the case where an asymmetric key shall be bound to an access token. The client makes the following HTTPS request shown in Figure 5. 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=Sp1x10BeZQQYbYS6WxSbIA
&redirect_uri=https%3A%2F%2Fclient%2Eexample%2Ecom%2Fcb
&token_type=pop
&req_cnf=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 'req_cnf' parameter contains the ECC public key the client would like to associate with the access token (in JSON format).

```
{
  "jwk": {
    "kty": "EC",
    "use": "sig",
    "crv": "P-256",
    "x": "18wHLeIgW9wVN6VD1Txgpqy2LszYkMf6J8njVAibvhM",
    "y": "-V4dS4UaLMgP_4fY4j8ir7cl1TXlFdAgcx55o7TkcSA"
  }
}
```

Figure 6: Client Providing Public Key to Authorization Server

4.2.2. 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. 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 "pop" 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",
  "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, as shown in Figure 8. The digital signature covering the access token offering authenticity and integrity protection is not shown below (but must be present).

```
{
  "iss": "https://authz.example.com",
  "aud": "https://resource.example.com",
  "exp": "1361398824",
  "nbf": "1360189224",
  "cnf": {
    "jwk" : {
      "kty" : "EC",
      "crv" : "P-256",
      "x" : "usWxHK2PmfHnHKwXPS54m0kTcGJ90UiglWiGahtagnv8",
      "y" : "IBOL+C3BttVivg+1SreASjpkttcsz+1rb7btKLv8EX4"
    }
  }
}
```

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 key (as well as the public key).

5. Security Considerations

[22] 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 [17] 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.

6. IANA Considerations

6.1. OAuth Access Token Types

This specification registers the following error in the IANA "OAuth Access Token Types" [25] established by [17].

- o Name: pop
- o Change controller: IESG
- o Specification document(s): [[this specification]]

6.2. OAuth Parameters Registration

This specification registers the following value in the IANA "OAuth Parameters" registry [25] established by [2].

- o Parameter name: cnf_req
- o Parameter usage location: authorization request, token request
- o Change controller: IESG
- o Specification document(s): [[this specification]]

- o Parameter name: cnf
- o Parameter usage location: authorization response, token response
- o Change controller: IESG
- o Specification document(s): [[this specification]]

- o Parameter name: rs_cnf
- o Parameter usage location: token response
- o Change controller: IESG
- o Specification document(s): [[this specification]]

6.3. OAuth Extensions Error Registration

This specification registers the following error in the IANA "OAuth Extensions Error Registry" [25] established by [2].

- o Error name: invalid_token_type
- o Error usage location: implicit grant error response, token error response
- o Related protocol extension: token_type parameter
- o Change controller: IESG
- o Specification document(s): [[this specification]]

7. Acknowledgements

We would like to thank Chuck Mortimore and James Manger for their review comments.

8. References

8.1. 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, <<https://www.rfc-editor.org/info/rfc2119>>.
- [2] Hardt, D., Ed., "The OAuth 2.0 Authorization Framework", RFC 6749, DOI 10.17487/RFC6749, October 2012, <<https://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, <<https://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, <<https://www.rfc-editor.org/info/rfc5246>>.
- [5] Jones, M., "JSON Web Key (JWK)", RFC 7517, DOI 10.17487/RFC7517, May 2015, <<https://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, <<https://www.rfc-editor.org/info/rfc7515>>.
- [7] Jones, M., "JSON Web Algorithms (JWA)", RFC 7518, DOI 10.17487/RFC7518, May 2015, <<https://www.rfc-editor.org/info/rfc7518>>.
- [8] Jones, M. and J. Hildebrand, "JSON Web Encryption (JWE)", RFC 7516, DOI 10.17487/RFC7516, May 2015, <<https://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, <<https://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, <<https://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, <<https://www.rfc-editor.org/info/rfc7638>>.
- [12] Jones, M., Seitz, L., Selander, G., Erdtman, S., and H. Tschofenig, "Proof-of-Possession Key Semantics for CBOR Web Tokens (CWTs)", draft-ietf-ace-cwt-proof-of-possession-06 (work in progress), February 2019.
- [13] Jones, M., Wahlstroem, E., Erdtman, S., and H. Tschofenig, "CBOR Web Token (CWT)", RFC 8392, DOI 10.17487/RFC8392, May 2018, <<https://www.rfc-editor.org/info/rfc8392>>.
- [14] Schaad, J., "CBOR Object Signing and Encryption (COSE)", RFC 8152, DOI 10.17487/RFC8152, July 2017, <<https://www.rfc-editor.org/info/rfc8152>>.
- [15] Jones, M., Nadalin, A., Campbell, B., Bradley, J., and C. Mortimore, "OAuth 2.0 Token Exchange", draft-ietf-oauth-token-exchange-16 (work in progress), October 2018.
- [16] Campbell, B., Bradley, J., and H. Tschofenig, "Resource Indicators for OAuth 2.0", draft-ietf-oauth-resource-indicators-02 (work in progress), January 2019.

8.2. Informative References

- [17] 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>>.
- [18] Crocker, D., Ed. and P. Overell, "Augmented BNF for Syntax Specifications: ABNF", STD 68, RFC 5234, DOI 10.17487/RFC5234, January 2008, <<https://www.rfc-editor.org/info/rfc5234>>.
- [19] 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>>.
- [20] 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>>.

- [21] 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>>.
- [22] 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.
- [23] Richer, J., Ed., "OAuth 2.0 Token Introspection", RFC 7662, DOI 10.17487/RFC7662, October 2015, <<https://www.rfc-editor.org/info/rfc7662>>.
- [24] Seitz, L., Selander, G., Wahlstroem, E., Erdtman, S., and H. Tschofenig, "Authentication and Authorization for Constrained Environments (ACE) using the OAuth 2.0 Framework (ACE-OAuth)", draft-ietf-ace-oauth-authz-24 (work in progress), March 2019.
- [25] IANA, "OAuth Parameters", October 2018.
- [26] IANA, "JSON Web Token Claims", June 2018.

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.independentid.com>

Michael B. Jones
Microsoft

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

Hannes Tschofenig
Arm Ltd.
Absam 6067
Austria

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

Mihaly Meszaros
GITDA
Debrecen 4033
Hungary

Email: bakfitty@gmail.com
URI: <https://github.com/misi>

OAuth Working Group
Internet-Draft
Intended status: Standards Track
Expires: June 20, 2016

M. Jones
Microsoft
J. Bradley
Ping Identity
H. Tschofenig
ARM Limited
December 18, 2015

Proof-of-Possession Key Semantics for JSON Web Tokens (JWTs)
draft-ietf-oauth-proof-of-possession-11

Abstract

This specification defines how to declare in a JSON Web Token (JWT) that the presenter of the JWT possesses a particular proof-of-possession key and that the recipient can cryptographically confirm proof-of-possession of the key by the presenter. Being able to prove possession of a key is also sometimes described as the presenter being a holder-of-key.

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 June 20, 2016.

Copyright Notice

Copyright (c) 2015 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	5
2. Terminology	5
3. Representations for Proof-of-Possession Keys	6
3.1. Confirmation Claim	6
3.2. Representation of an Asymmetric Proof-of-Possession Key	7
3.3. Representation of an Encrypted Symmetric Proof-of-Possession Key	8
3.4. Representation of a Key ID for a Proof-of-Possession Key	9
3.5. Representation of a URL for a Proof-of-Possession Key	9
3.6. Specifics Intentionally Not Specified	10
4. Security Considerations	10
5. Privacy Considerations	11
6. IANA Considerations	11
6.1. JSON Web Token Claims Registration	12
6.1.1. Registry Contents	12
6.2. JWT Confirmation Methods Registry	12
6.2.1. Registration Template	12
6.2.2. Initial Registry Contents	13
7. References	13
7.1. Normative References	13
7.2. Informative References	14
Appendix A. Acknowledgements	15
Appendix B. Document History	15
Authors' Addresses	17

1. Introduction

This specification defines how a JSON Web Token [JWT] can declare that the presenter of the JWT possesses a particular proof-of-possession (PoP) key and that the recipient can cryptographically confirm proof-of-possession of the key by the presenter. Proof-of-possession of a key is also sometimes described as the presenter being a holder-of-key. The [I-D.ietf-oauth-pop-architecture] specification describes key confirmation, among other confirmation mechanisms. This specification defines how to communicate key confirmation key information in JWTs.

Envision the following two use cases. The first use case employs a symmetric proof-of-possession key and the second use case employs an asymmetric proof-of-possession key.

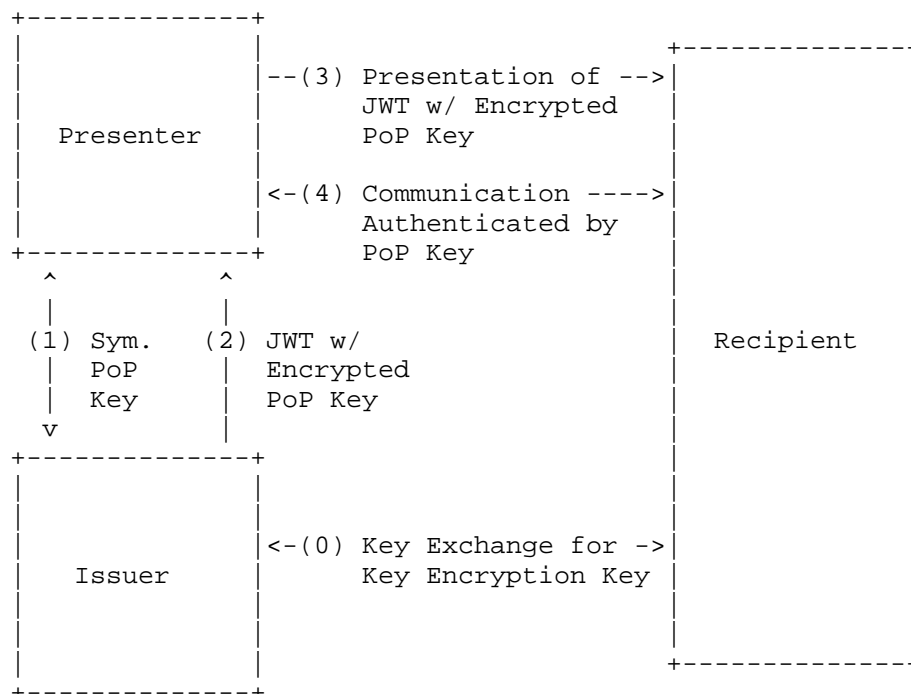


Figure 1: Proof-of-Possession with a Symmetric Key

In the case illustrated in Figure 1, either the presenter generates a symmetric key and privately sends it to the issuer (1) or the issuer generates a symmetric key and privately sends it to the presenter (1). The issuer generates a JWT with an encrypted copy of this symmetric key in the confirmation claim. This symmetric key is

encrypted with a key known only to the issuer and the recipient, which was previously established in step (0). The entire JWT is integrity protected by the issuer. The JWT is then (2) sent to the presenter. Now, the presenter is in possession of the symmetric key as well as the JWT (which includes the confirmation claim). When the presenter (3) presents the JWT to the recipient, it also needs to demonstrate possession of the symmetric key; the presenter, for example, (4) uses the symmetric key in a challenge/response protocol with the recipient. The recipient is then able to verify that it is interacting with the genuine presenter by decrypting the key in the confirmation claim of the JWT. By doing this, the recipient obtains the symmetric key, which it then uses to verify cryptographically protected messages exchanged with the presenter (4). This symmetric key mechanism described above is conceptually similar to the use of Kerberos tickets.

Note that for simplicity, the diagram above and associated text describe the direct use of symmetric keys without the use of derived keys. A more secure practice is to derive the symmetric keys actually used from secrets exchanged, such as the key exchanged in step (0), using a Key Derivation Function (KDF) and use the derived keys, rather than directly using the secrets exchanged.

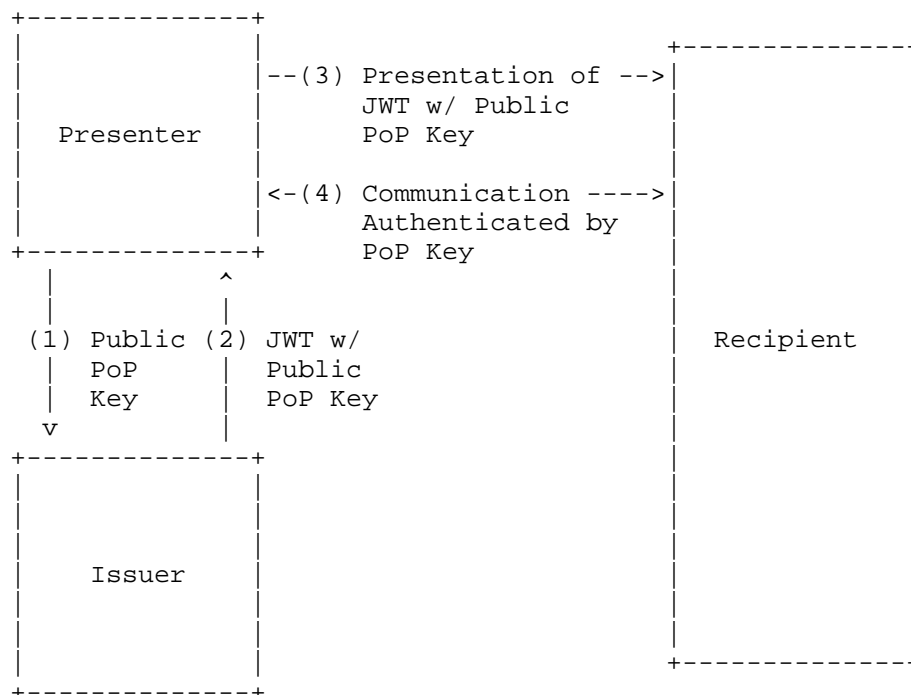


Figure 2: Proof-of-Possession with an Asymmetric Key

In the case illustrated in Figure 2, the presenter generates a public/private key pair and (1) sends the public key to the issuer, which creates a JWT that contains the public key (or an identifier for it) in the confirmation claim. The entire JWT is integrity protected using a digital signature to protect it against modifications. The JWT is then (2) sent to the presenter. When the presenter (3) presents the JWT to the recipient, it also needs to demonstrate possession of the private key. The presenter, for example, (4) uses the private key in a TLS exchange with the recipient or (4) signs a nonce with the private key. The recipient is able to verify that it is interacting with the genuine presenter by extracting the public key from the confirmation claim of the JWT (after verifying the digital signature of the JWT) and utilizing it with the private key in the TLS exchange or by checking the nonce signature.

In both cases, the JWT may contain other claims that are needed by the application.

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. Terminology

This specification uses terms defined in the JSON Web Token [JWT], JSON Web Key [JWK], and JSON Web Encryption [JWE] specifications.

These terms are defined by this specification:

Issuer

Party that creates the JWT and binds the proof-of-possession key to it.

Presenter

Party that proves possession of a private key (for asymmetric key cryptography) or secret key (for symmetric key cryptography) to a recipient.

Recipient

Party that receives the JWT containing the proof-of-possession key information from the presenter.

3. Representations for Proof-of-Possession Keys

By including a "cnf" (confirmation) claim in a JWT, the issuer of the JWT declares that the presenter possesses a particular key, and that the recipient can cryptographically confirm that the presenter has possession of that key. The value of the "cnf" claim is a JSON object and the members of that object identify the proof-of-possession key.

The presenter can be identified in one of several ways by the JWT, depending upon the application requirements. If the JWT contains a "sub" (subject) claim [JWT], the presenter is normally the subject identified by the JWT. (In some applications, the subject identifier will be relative to the issuer identified by the "iss" (issuer) claim [JWT].) If the JWT contains no "sub" (subject) claim, the presenter is normally the issuer identified by the JWT using the "iss" (issuer) claim. The case in which the presenter is the subject of the JWT is analogous to SAML 2.0 [OASIS.saml-core-2.0-os] SubjectConfirmation usage. At least one of the "sub" and "iss" claims MUST be present in the JWT. Some use cases may require that both be present.

Another means used by some applications to identify the presenter is an explicit claim, such as the "azp" (authorized party) claim defined by OpenID Connect [OpenID.Core]. Ultimately, the means of identifying the presenter is application-specific, as is the means of confirming possession of the key that is communicated.

3.1. Confirmation Claim

The "cnf" (confirmation) claim is used in the JWT to contain members used to identify the proof-of-possession key. Other members of the "cnf" object may be defined because a proof-of-possession key may not be the only means of confirming the authenticity of the token. This is analogous to the SAML 2.0 [OASIS.saml-core-2.0-os] SubjectConfirmation element, in which a number of different subject confirmation methods can be included, including proof-of-possession key information.

The set of confirmation members that a JWT must contain to be considered valid is context dependent and is outside the scope of this specification. Specific applications of JWTs will require implementations to understand and process some confirmation members in particular ways. However, in the absence of such requirements,

all confirmation members that are not understood by implementations MUST be ignored.

This specification establishes the IANA "JWT Confirmation Methods" registry for these members in Section 6.2 and registers the members defined by this specification. Other specifications can register other members used for confirmation, including other members for conveying proof-of-possession keys, possibly using different key representations.

The "cnf" claim value MUST represent only a single proof-of-possession key; thus, at most one of the "jwk", "jwe", and "jku" confirmation values defined below may be present. Note that if an application needs to represent multiple proof-of-possession keys in the same JWT, one way for it to achieve this is to use other claim names, in addition to "cnf", to hold the additional proof-of-possession key information. These claims could use the same syntax and semantics as the "cnf" claim. Those claims would be defined by applications or other specifications and could be registered in the IANA "JSON Web Token Claims" registry [IANA.JWT.Claims].

3.2. Representation of an Asymmetric Proof-of-Possession Key

When the key held by the presenter is an asymmetric private key, the "jwk" member is a JSON Web Key [JWK] representing the corresponding asymmetric public key. The following example demonstrates such a declaration in the JWT Claims Set of a JWT:

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

The JWK MUST contain the required key members for a JWK of that key type and MAY contain other JWK members, including the "kid" (key ID) member.

The "jwk" member MAY also be used for a JWK representing a symmetric

key, provided that the JWT is encrypted so that the key is not revealed to unintended parties. If the JWT is not encrypted, the symmetric key MUST be encrypted as described below.

3.3. Representation of an Encrypted Symmetric Proof-of-Possession Key

When the key held by the presenter is a symmetric key, the "jwe" member is an encrypted JSON Web Key [JWK] encrypted to a key known to the recipient using the JWE Compact Serialization containing the symmetric key. The rules for encrypting a JWK are found in Section 7 of the JSON Web Key [JWK] specification.

The following example illustrates a symmetric key that could subsequently be encrypted for use in the "jwe" member:

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

The UTF-8 [RFC3629] encoding of this JWK is used as the JWE Plaintext when encrypting the key.

The following example is a JWE Header that could be used when encrypting this key:

```
{
  "alg": "RSA-OAEP",
  "enc": "A128CBC-HS256"
}
```

The following example JWT Claims Set of a JWT illustrates the use of an encrypted symmetric key as the "jwe" member value:

```
{
  "iss": "https://server.example.com",
  "sub": "24400320",
  "aud": "s6BhdRkqt3",
  "nonce": "n-0S6_WzA2Mj",
  "exp": 1311281970,
  "iat": 1311280970,
  "cnf": {
    "jwe":
      "eyJhbGciOiJSU0EtT0FFUCIsImVuYyI6IkJExMjhdQkMtSFMyNTYifQ.
      (remainder of JWE omitted for brevity)"
  }
}
```

3.4. Representation of a Key ID for a Proof-of-Possession Key

The proof-of-possession key can also be identified by the use of a Key ID instead of communicating the actual key, provided the recipient is able to obtain the identified key using the Key ID. In this case, the issuer of a JWT declares that the presenter possesses a particular key and that the recipient can cryptographically confirm proof-of-possession of the key by the presenter by including a "cnf" (confirmation) claim in the JWT whose value is a JSON object, with the JSON object containing a "kid" (key ID) member identifying the key.

The following example demonstrates such a declaration in the JWT Claims Set of a JWT:

```
{
  "iss": "https://server.example.com",
  "aud": "https://client.example.org",
  "exp": 1361398824,
  "cnf": {
    "kid": "dfdlaa97-6d8d-4575-a0fe-34b96de2bfad"
  }
}
```

The content of the "kid" value is application specific. For instance, some applications may choose to use a JWK Thumbprint [JWK.Thumbprint] value as the "kid" value.

3.5. Representation of a URL for a Proof-of-Possession Key

The proof-of-possession key can be passed by reference instead of being passed by value. This is done using the "jku" (JWK Set URL) member. Its value is a URI [RFC3986] that refers to a resource for a set of JSON-encoded public keys represented as a JWK Set [JWK], one of which is the proof-of-possession key. If there are multiple keys in the referenced JWK Set document, a "kid" member MUST also be included, with the referenced key's JWK also containing the same "kid" value.

The protocol used to acquire the resource MUST provide integrity protection. An HTTP GET request to retrieve the JWK Set MUST use Transport Layer Security (TLS) [RFC5246] and the identity of the server MUST be validated, as per Section 6 of RFC 6125 [RFC6125].

The following example demonstrates such a declaration in the JWT Claims Set of a JWT:

```
{
  "iss": "https://server.example.com",
  "sub": "17760704",
  "aud": "https://client.example.org",
  "exp": 1440804813,
  "cnf": {
    "jku": "https://keys.example.net/pop-keys.json",
    "kid": "2015-08-28"
  }
}
```

3.6. Specifics Intentionally Not Specified

Proof-of-possession is typically demonstrated by having the presenter sign a value determined by the recipient using the key possessed by the presenter. This value is sometimes called a "nonce" or a "challenge".

The means of communicating the nonce and the nature of its contents are intentionally not described in this specification, as different protocols will communicate this information in different ways. Likewise, the means of communicating the signed nonce is also not specified, as this is also protocol-specific.

Note that another means of proving possession of the key when it is a symmetric key is to encrypt the key to the recipient. The means of obtaining a key for the recipient is likewise protocol-specific.

For examples using the mechanisms defined in this specification, see [I-D.ietf-oauth-pop-architecture].

4. Security Considerations

All of the security considerations that are discussed in [JWT] also apply here. In addition, proof-of-possession introduces its own unique security issues. Possessing a key is only valuable if it is kept secret. Appropriate means must be used to ensure that unintended parties do not learn private key or symmetric key values.

Applications utilizing proof-of-possession should also utilize audience restriction, as described in Section 4.1.3 of [JWT], as it provides different protections. Proof-of-possession can be used by recipients to reject messages from unauthorized senders. Audience restriction can be used by recipients to reject messages intended for different recipients.

A recipient might not understand the "cnf" claim. Applications that

require the proof-of-possession keys communicated with it to be understood and processed must ensure that the parts of this specification that they use are implemented.

Proof-of-possession via encrypted symmetric secrets is subject to replay attacks. This attack can be avoided when a signed nonce or challenge is used, since the recipient can use a distinct nonce or challenge for each interaction. Replay can also be avoided if a sub-key is derived from a shared secret that is specific to the instance of the PoP demonstration.

Similarly to other information included in a JWT, it is necessary to apply data origin authentication and integrity protection (via a keyed message digest or a digital signature). Data origin authentication ensures that the recipient of the JWT learns about the entity that created the JWT, since this will be important for any policy decisions. Integrity protection prevents an adversary from changing any elements conveyed within the JWT payload. Special care has to be applied when carrying symmetric keys inside the JWT, since those not only require integrity protection, but also confidentiality protection.

5. Privacy Considerations

A proof-of-possession key can be used as a correlation handle if the same key is used with multiple parties. Thus, for privacy reasons, it is recommended that different proof-of-possession keys be used when interacting with different parties.

6. IANA Considerations

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

Values are registered on a Specification Required [RFC5226] basis after a three-week review period on the `oauth-pop-reg-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. [[Note to the RFC Editor: The name of the mailing list should be determined in consultation with the IESG and IANA. Suggested name: `oauth-pop-reg-review@ietf.org`.]]

Registration requests sent to the mailing list for review should use an appropriate subject (e.g., "Request to register JWT Confirmation

Method: example"). 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 include 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, evaluating the security properties of the item being registered, and whether the registration makes sense.

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.

6.1. JSON Web Token Claims Registration

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

6.1.1. Registry Contents

- o Claim Name: "cnf"
- o Claim Description: Confirmation
- o Change Controller: IESG
- o Specification Document(s): Section 3.1 of [[this document]]

6.2. JWT Confirmation Methods Registry

This specification establishes the IANA "JWT Confirmation Methods" registry for JWT "cnf" member values. The registry records the confirmation method member and a reference to the specification that defines it.

6.2.1. Registration Template

Confirmation Method Value:

The name requested (e.g., "kid"). Because a core goal of this specification is for the resulting representations to be compact, it is RECOMMENDED that the name be short -- not to exceed 8 characters without a compelling reason to do so. This name is case-sensitive. Names may not match other registered names in a case-insensitive manner unless the Designated Experts state that there is a compelling reason to allow an exception.

Confirmation Method Description:

Brief description of the confirmation method (e.g., "Key Identifier").

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.

6.2.2. Initial Registry Contents

- o Confirmation Method Value: "jwk"
- o Confirmation Method Description: JSON Web Key Representing Public Key
- o Change Controller: IESG
- o Specification Document(s): Section 3.2 of [[this document]]

- o Confirmation Method Value: "jwe"
- o Confirmation Method Description: Encrypted JSON Web Key
- o Change Controller: IESG
- o Specification Document(s): Section 3.3 of [[this document]]

- o Confirmation Method Value: "kid"
- o Confirmation Method Description: Key Identifier
- o Change Controller: IESG
- o Specification Document(s): Section 3.4 of [[this document]]

- o Confirmation Method Value: "jku"
- o Confirmation Method Description: JWK Set URL
- o Change Controller: IESG
- o Specification Document(s): Section 3.5 of [[this document]]

7. References**7.1. Normative References**

[IANA.JWT.Claims]

IANA, "JSON Web Token Claims",
<<http://www.iana.org/assignments/jwt>>.

[JWE] Jones, M. and J. Hildebrand, "JSON Web Encryption (JWE)",

- RFC 7516, DOI 10.17487/RFC7156, May 2015,
<<http://www.rfc-editor.org/info/rfc7516>>.
- [JWK] Jones, M., "JSON Web Key (JWK)", RFC 7517, DOI 10.17487/
RFC7157, May 2015,
<<http://www.rfc-editor.org/info/rfc7517>>.
- [JWT] Jones, M., Bradley, J., and N. Sakimura, "JSON Web Token
(JWT)", RFC 7519, DOI 10.17487/RFC7159, May 2015,
<<http://www.rfc-editor.org/info/rfc7519>>.
- [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>>.
- [RFC3629] Yergeau, F., "UTF-8, a transformation format of ISO
10646", STD 63, RFC 3629, DOI 10.17487/RFC3629,
November 2003, <<http://www.rfc-editor.org/info/rfc3629>>.
- [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,
<<http://www.rfc-editor.org/info/rfc3986>>.
- [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>>.
- [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>>.
- [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, <<http://www.rfc-editor.org/info/rfc6125>>.

7.2. Informative References

- [I-D.ietf-oauth-pop-architecture]
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-05 (work

in progress), October 2015.

[JWK.Thumbprint]

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>>.

[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>.

Appendix A. Acknowledgements

The authors wish to thank Brian Campbell, Stephen Farrell, Barry Leiba, Kepeng Li, Chris Lonvick, James Manger, Kathleen Moriarty, Justin Richer, and Nat Sakimura for their reviews of the specification.

Appendix B. Document History

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

-11

- o Addressed Sec-Dir review comments by Chris Lonvick and ballot comments by Stephen Farrell.

-10

- o Addressed ballot comments by Barry Leiba.

-09

- o Removed erroneous quotation marks around numeric "exp" claim values in examples.

-08

- o Added security consideration about also utilizing audience restriction.

-07

- o Addressed review comments by Hannes Tschofenig, Kathleen Moriarty, and Justin Richer. Changes were:
- o Clarified that symmetric proof-of-possession keys can be generated by either the presenter or the issuer.
- o Clarified that confirmation members that are not understood must be ignored unless otherwise specified by the application.

-06

- o Added diagrams to the introduction.

-05

- o Addressed review comments by Kepeng Li.

-04

- o Allowed the use of "jwk" for symmetric keys when the JWT is encrypted.
- o Added the "jku" (JWK Set URL) member.
- o Added privacy considerations.
- o Reordered sections so that the "cnf" (confirmation) claim is defined before it is used.
- o Noted that applications can define new claim names, in addition to "cnf", to represent additional proof-of-possession keys, using the same representation as "cnf".
- o Applied wording clarifications suggested by Nat Sakimura.

-03

- o Separated the "jwk" and "jwe" confirmation members; the former represents a public key as a JWK and the latter represents a symmetric key as a JWE encrypted JWK.
- o Changed the title to indicate that a proof-of-possession key is being communicated.

- o Updated language that formerly assumed that the issuer was an OAuth 2.0 authorization server.
- o Described ways that applications can choose to identify the presenter, including use of the "iss", "sub", and "azp" claims.
- o Harmonized the registry language with that used in JWT [RFC 7519].
- o Addressed other issues identified during working group last call.
- o Referenced the JWT and JOSE RFCs.

-02

- o Defined the terms Issuer, Presenter, and Recipient and updated their usage within the document.
- o Added a description of a use case using an asymmetric proof-of-possession key to the introduction.
- o Added the "kid" (key ID) confirmation method.
- o These changes address the open issues identified in the previous draft.

-01

- o Updated references.

-00

- o Created the initial working group draft from draft-jones-oauth-proof-of-possession-02.

Authors' Addresses

Michael B. Jones
Microsoft

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

John Bradley
Ping Identity

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

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: 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.eyJhdWQiOiJodHRwczovL2JhY2t1bmcuZm9udXhhdXBsZS5jb20iLCJpc3MiOiJodHRwczovL2FzLmV4YW1wbGUuY29tIiwiaXNjb3BlIjoiaXBpIn0.40y3ZgQedw6rx
f59WlwHDD9jryF0r0_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.eyJhdWQiOiJodHRwczovL2JhY2t1bmQuZXhhbXBsZS5jb20iLCJpc3MiOiJodHRwczovL2FzLmV4YW1wbGUuY29tIiwiaXNjaWJoxNDQxOTE3NTkzLCJpYXQiOiJlbnV4bm90eS5MTc1MzMsInN1YiI6ImUuY29tIiwiaXNjaWJ3BlIjoiYXBpIn0.40y3ZgQe
dw6rx5f59WlhDD9jryFOr0_Wh3CGozQBihNBhnXEQgU85AI9x3KmsPottVMLPIW
vmDCMy5-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.eyJhdWQiOiJodHRwczovL2FzLmV4YW1wbGUuY29tIiwiaXNzIjoiaHR0cHM6Ly9vcmlnaW5hbC1pc3N1ZXIuZXhhbXBsZS5uZXQlLCJleHAiOjE0NDE5MTA2MDAsIm5iZiI6MTQ0MTkwOTAwMCwic3ViIjoiYmRjQGV4YW1wbGUubmV0Iiwic2NvcGUiOiJvcmlcnMgcHJvZmlsZSBoaXN0b3J5In0.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": "eyJhbGciOiJIJFZlIiwiaW50ZG90IiwiaXNzIjoiaHR0cHM6Ly9hcy51eGFtcGxlLnV4cCI6MTQ0MTkxMzYxMCwic3ViIjoieYmRjQGV4YW1wbGUubmV0Iiwic2NvcGUiOiJvcmlldmVudDZ2Z2WkGLMqR9ztj6w2OXraQ1kjmGjyicq24kcB7AI2VqVx13wSWnVKh85A",
  "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.eyJhdWQiOiJodHRwczovL2FzLmV4YW1wbGUuY29tIiwiaXNzIjoiaHR0cHM6Ly9vcmlnaW5hbC1pc3N1ZXIuZlZlZmV4YW1wbGUuY29tIiwibWF5X2FjZDI6eyJzdWIiOiJhZG1lbnBkbleGFtcGx1Lm5ldCJ9fQ.4rPRSWihQbpMIgAmAoqaJojAxj-p2X8_fAtAGTXrvMxU-eEZhnXqY0_AOZgLdxw5DyLzua8H_I10MCcckF-Q_g
&subject_token_type=urn%3Aietf%3Aparams%3Aoauth%3Atoken-type%3Ajwt
&actor_token=eyJhbGciOiJIUzI1NiIsImtpZCI6IjE2In0.eyJhdWQiOiJodHRwczovL2FzLmV4YW1wbGUuY29tIiwiaXNzIjoiaHR0cHM6Ly9vcmlnaW5hbC1pc3N1ZXIuZlZlZmV4YW1wbGUuY29tIiwibWF5X2FjZDI6eyJzdWIiOiJhZG1lbnBkbleGFtcGx1Lm5ldCJ9fQ.4rPRSWihQbpMIgAmAoqaJojAxj-p2X8_fAtAGTXrvMxU-eEZhnXqY0_AOZgLdxw5DyLzua8H_I10MCcckF-Q_g
&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": "eyJhbGciOiJIUzI1NiIsImtpZCI6IjcyIn0.eyJhdWQiOiJ1cm46ZXhhbXBzZTpjb29wZXJhdGlvbiljb250ZXh0IiwiaXNzIjoiaHR0cHM6Ly9hcy5leGFtcGxlLmNvbSIsImV4cCI6MTQ0MTkxMzYxMCwic2NvcGUOiJzdGF0dXMgZmVlZCIsInN1YiI6InVzZXJAZXhhbXBzZS5uZXQlLCJhY3QiOiOnsic3ViIjojYWWRtaW5AZXhhbXBzZS5uZXQifX0.3paKl9UySKYB5ng6_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 (deseccration -> 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