OAuth Working Group	N. Sakimura
Internet-Draft	NAT.Consulting
Intended status: Standards Track	J. Bradley
Expires: March 13, 2021	Yubico
	M. Jones
	Microsoft
	September 9, 2020

The OAuth 2.0 Authorization Framework: JWT Secured Authorization Request (JAR)

draft-ietf-oauth-jwsreq-29

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, and (b) the source of the communication is not authenticated. 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

Copyright Notice

Table of Contents

1. Introduction

The Authorization Request in OAuth 2.0 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 and Mix-up attack [FETT], 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 third party so that a client application cannot request more permissions than previously agreed. This offers an additional degree of privacy protection.

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

The JWT 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 flows. The request parameter is a JSON Web Token (JWT) 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 can be passed to the authorization endpoint by reference, in which case the parameter request_uri is used instead of the request.

Using JWT 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 third party attesting that the authorization request is compliant with a certain policy. For example, a request can be pre-examined by a 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 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, JSON Web Signature, and JSON Web Encryption apply.

2.1. Request Object

JSON Web Token (JWT) 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, 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 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 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 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 object constitutes the JWT Claims Set defined in JWT. The JWT Claims Set is then signed or signed and encrypted.

To sign, JSON Web Signature (JWS) is used. The result is a JWS signed JWT. 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 specification. The value of aud should be the value of the Authorization Server (AS) issuer as defined in RFC8414.

To encrypt, JWE 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 has the mime-type 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 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 results in this Request Object value (with line wraps within values for display purposes only):

```
eyJhbGciOiJSUzI1NiIsImtpZCI6ImsyYmRjIn0.ewogICAgImlzcyI6ICJzNkJoZF
JrcXQzIiwKICAgICJhdWQiOiAiaHR0cHM6Ly9zZXJ2ZXIuZXhhbXBsZS5jb20iLAog
ICAgInJlc3BvbnNlX3R5cGUiOiAiY29kZSBpZF90b2tlbiIsCiAgICAiY2xpZW50X2
lkIjogInM2QmhkUmtxdDMiLAogICAgInJlZGlyZWN0X3VyaSI6ICJodHRwczovL2Ns
aWVudC5leGFtcGxlLm9yZy9jYiIsCiAgICAic2NvcGUiOiAib3BlbmlkIiwKICAgIC
JzdGF0ZSI6ICJhZjBpZmpzbGRraiIsCiAgICAibm9uY2UiOiAibi0wUzZfV3pBMk1q
IiwKICAgICJtYXhfYWdlIjogODY0MDAKfQ.Nsxa_18VUElVaPjqW_ToI1yrEJ67BgK
b5xsuZRVqzGkfKrOIX7BCx0biSxYGmjK9KJPctH1OC0iQJwXu5YVY-vnW0_PLJb1C2
HG-ztVzcnKZC2gE4i0vgQcpkUOCpW3SEYXnyWnKzuKzqSb1wAZALo5f89B_p6QA6j6
JwBSRvdVsDPdulW8lKxGTbH82czCaQ50rLAg3EYLYaCb4ik4I1zGXE4fvim9FIMs8O
CMmzwIB5S-ujFfzwFjoyuPEV4hJnoVUmXR_W9typPf846lGwA8h9G9oNTIuX8Ft2jf
pnZdFmLg3_wr3Wa5q3a-lfbgF3S9H_8nN3j1i7tLR_5Nz-g
```

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_cdHmfmG2ltd2yabEO9XadD8
      pJNZubINPpmgHh3J1aD9WRwS05ucmFq3CfFsluLt13_7oX5yDRSKX7poXmT_5
      ko8k4NJZPMAO8fPToDTH7kHYbONSE2FYa5GZ60CUsFhSonI-dcMDJ0Ary9lxI
      w5k2z4TAdARVWcS7sD07VhlMMshrwsPHBQgTatlkxyIHXbYdtak8fqvNAwr7O
      lVEvM_Ipf5OfmdB8Sd-wjzaBsyP4VhJKoi_qdgSzpC694XZeYPq45Sw-q51iF
      UlcOlTCI7z6jltUtnR6ySn6XDGFnzH5Fe5ypw",
 "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 that holds authorization request parameters stated in section 4 of OAuth 2.0. 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 that is the Request Object URI referencing the authorization request parameters stated in section 4 of OAuth 2.0. If this parameter is present in the authorization request, request MUST NOT be present.

client_id

REQUIRED. OAuth 2.0 client_id. The value MUST match the request or request_uri Request Object's 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=eyJhbG..AlMGzw 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=eyJhbGciOiJSUzI1NiIsImtpZCI6ImsyYmRjIn0.ewogICAgImlzcyI6
  ICJzNkJoZFJrcXQzIiwKICAgICJhdWQiOiAiaHR0cHM6Ly9zZXJ2ZXIuZXhhbXBs
  ZS5jb20iLAogICAgInJlc3BvbnNlX3R5cGUiOiAiY29kZSBpZF90b2tlbiIsCiAg
  ICAiY2xpZW50X2lkIjogInM2QmhkUmtxdDMiLAogICAgInJlZGlyZWN0X3VyaSI6
  ICJodHRwczovL2NsaWVudC5leGFtcGxlLm9yZy9jYiIsCiAgICAic2NvcGUiOiAi
```

b3BlbmlkIiwKICAgICJzdGF0ZSI6ICJhZjBpZmpzbGRraiIsCiAgICAibm9uY2Ui
OiAibi0wUzZfV3pBMk1qIiwKICAgICJtYXhfYWdlIjogODY0MDAKfQ.Nsxa_18VU
ElVaPjqW_ToI1yrEJ67BgKb5xsuZRVqzGkfKrOIX7BCx0biSxYGmjK9KJPctH1OC
0iQJwXu5YVY-vnW0_PLJb1C2HG-ztVzcnKZC2gE4i0vgQcpkUOCpW3SEYXnyWnKz
uKzqSb1wAZALo5f89B_p6QA6j6JwBSRvdVsDPdulW8lKxGTbH82czCaQ50rLAg3E
YLYaCb4ik4I1zGXE4fvim9FIMs8OCMmzwIB5S-ujFfzwFjoyuPEV4hJnoVUmXR_W
9typPf846lGwA8h9G9oNTIuX8Ft2jfpnZdFmLg3_wr3Wa5q3a-lfbgF3S9H_8nN3
j1i7tLR_5Nz-g

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 MUST NOT exceed 512 ASCII characters. There are three 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. In the second case, it MUST be a URN, as specified in 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):

eyJhbGciOiJSUzI1NiIsImtpZCI6ImsyYmRjIn0.ewogICAgImlzcyI6ICJzNkJoZF
JrcXQzIiwKICAgICJhdWQiOiAiaHR0cHM6Ly9zZXJ2ZXIuZXhhbXBsZS5jb20iLAog
ICAgInJlc3BvbnNlX3R5cGUiOiAiY29kZSBpZF90b2tlbiIsCiAgICAiY2xpZW50X2
lkIjogInM2QmhkUmtxdDMiLAogICAgInJlZGlyZWN0X3VyaSI6ICJodHRwczovL2Ns
aWVudC5leGFtcGxlLm9yZy9jYiIsCiAgICAic2NvcGUiOiAib3BlbmlkIiwKICAgIC
JzdGF0ZSI6ICJhZjBpZmpzbGRraiIsCiAgICAibm9uY2UiOiAibi0wUzZfV3pBMk1q
IiwKICAgICJtYXhfYWdlIjogODY0MDAKfQ.Nsxa_18VUElVaPjqW_ToI1yrEJ67BgK
b5xsuZRVqzGkfKrOIX7BCx0biSxYGmjK9KJPctH1OC0iQJwXu5YVY-vnW0_PLJb1C2
HG-ztVzcnKZC2gE4i0vgQcpkUOCpW3SEYXnyWnKzuKzqSb1wAZALo5f89B_p6QA6j6
JwBSRvdVsDPdulW8lKxGTbH82czCaQ50rLAg3EYLYaCb4ik4I1zGXE4fvim9FIMs8O
CMmzwIB5S-ujFfzwFjoyuPEV4hJnoVUmXR_W9typPf846lGwA8h9G9oNTIuX8Ft2jf
pnZdFmLg3_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 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. 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 third-party Trust Framework Provider (TFP) 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 third-party Trust Framework Provider (TFP) 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

eyJhbGciOiJSUzI1NiIsImtpZCI6ImsyYmRjIn0.ewogICAgImlzcyI6ICJzNkJoZF
JrcXQzIiwKICAgICJhdWQiOiAiaHR0cHM6Ly9zZXJ2ZXIuZXhhbXBsZS5jb20iLAog
ICAgInJlc3BvbnNlX3R5cGUiOiAiY29kZSBpZF90b2tlbiIsCiAgICAiY2xpZW50X2
lkIjogInM2QmhkUmtxdDMiLAogICAgInJlZGlyZWN0X3VyaSI6ICJodHRwczovL2Ns
aWVudC5leGFtcGxlLm9yZy9jYiIsCiAgICAic2NvcGUiOiAib3BlbmlkIiwKICAgIC
JzdGF0ZSI6ICJhZjBpZmpzbGRraiIsCiAgICAibm9uY2UiOiAibi0wUzZfV3pBMk1q
IiwKICAgICJtYXhfYWdlIjogODY0MDAKfQ.Nsxa_18VUElVaPjqW_ToI1yrEJ67BgK
b5xsuZRVqzGkfKrOIX7BCx0biSxYGmjK9KJPctH1OC0iQJwXu5YVY-vnW0_PLJb1C2
HG-ztVzcnKZC2gE4i0vgQcpkUOCpW3SEYXnyWnKzuKzqSb1wAZALo5f89B_p6QA6j6
JwBSRvdVsDPdulW8lKxGTbH82czCaQ50rLAg3EYLYaCb4ik4I1zGXE4fvim9FIMs8O
CMmzwIB5S-ujFfzwFjoyuPEV4hJnoVUmXR_W9typPf846lGwA8h9G9oNTIuX8Ft2jf
```

```
pnZdFmLg3_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 specification.

The result is a signed request object.

If decryption fails, the Authorization Server MUST return an invalid_request_object error.

6.2. JWS Signed Request Object

The Authorization Server MUST validate the signature of the JSON Web Signature signed Request Object. The signature MUST be validated using the key for that client_id and the algorithm specified in the alg Header Parameter.

If signature validation fails, the Authorization Server MUST return an invalid_request_object error.

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.

If the validation fails, then the Authorization Server MUST return an error as specified in OAuth 2.0.

7. Authorization Server Response

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

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).

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:

- 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.
- DNS names in server certificates MAY contain the wildcard character "*".
- 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].
- 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].

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

- Name: sub
- Parameter Usage Location: authorization request
- Change Controller: IETF
- Specification Document(s): Section 4.1.2 of [RFC7519] and this document.

- Name: aud
- Parameter Usage Location: authorization request
- Change Controller: IETF
- Specification Document(s): Section 4.1.3 of [RFC7519] and this document.

- Name: exp
- Parameter Usage Location: authorization request
- Change Controller: IETF
- Specification Document(s): Section 4.1.4 of [RFC7519] and this document.

- Name: nbf
- Parameter Usage Location: authorization request
- Change Controller: IETF
- Specification Document(s): Section 4.1.5 of [RFC7519] and this document.

- Name: iat
- Parameter Usage Location: authorization request
- Change Controller: IETF
- Specification Document(s): Section 4.1.6 of [RFC7519] and this document.

- Name: jti
- Parameter Usage Location: authorization request

- Change Controller: IETF
- Specification Document(s): Section 4.1.7 of [RFC7519] and this document.

9.2. OAuth Authorization Server Metadata Registry

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

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

9.3. OAuth Dynamic Client Registration Metadata Registry

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

- Metadata Name: require_signed_request_object
- Metadata Description: Indicates where authorization request needs to be protected as Request Object and provided through either request or request_uri parameter.
- Change Controller: IETF
- 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.

- Type name: application
- Subtype name: oauth.authz.req+jwt
- Required parameters: n/a
- Optional parameters: n/a
- 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.
- Security considerations: See Section 10 of [[this specification]]
- Interoperability considerations: n/a
- Published specification: Section 4 of [[this specification]]
- Applications that use this media type: Applications that use Request Objects to make an OAuth 2.0 Authorization Request
- Fragment identifier considerations: n/a
- Additional information:

  Magic number(s): n/a

  File extension(s): n/a

  Macintosh file type code(s): n/a

- Person & email address to contact for further information:
  Nat Sakimura, nat@nat.consulting

- Intended usage: COMMON
- Restrictions on usage: none
- Author: Nat Sakimura, nat@nat.consulting
- Change controller: IETF
- Provisional registration? No

10. Security Considerations

In addition to the all the security considerations discussed in OAuth 2.0, 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 or signed then encrypted using JWS and JWE 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.

(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 third party, such as a Trust Framework Provider(TFP), implements a service that 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 a member of the trust framework and MUST know out-of-band that the client also uses the trust framework.

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.

The lack of linkage among those endpoints is cited as the cause of the Cross-Phase Attacks described in [FETT]. An extension specification could be created providing this linkage as a means to address the risks.

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 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 content 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 content 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". 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". If omitted, the default value is false.

10.6. TLS Security Considerations

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

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.

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 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 and The OAuth 2.0 Authorization Framework: Bearer Token Usage 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 Trust Framework Provider 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 Trust Framework Provider to obtain the Request Object URI. This process is two steps:

(1)
: (Certification Process) The TFP 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 TFP to get the request_uri.

(2)
: (Translation Process) The client uses the client credential that it got to push the request object to the TFP to get the request_uri. The TFP 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 Trust Framework Provider. 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 Trust Framework Operator for the 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 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 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), Sergey Beryozkin, Ben Campbell (as AD), Brian Campbell (Ping Identity), Roman Danyliw (as AD), Vladimir Dzhuvinov (Connect2id), Joel Halpern (as GENART), Benjamin Kaduk (as AD), Stephen Kent (as SECDIR), Murray Kucherawy (as AD), Warren Kumari (as OPSDIR), Torsten Lodderstedt (yes.com), Jim Manico, Axel Nennker (Deutsche Telecom), Hannes Tschofenig (ARM), Dirk Balfanz (Google), James H. Manger (Telstra), Kathleen Moriarty (as AD), John Panzer (Google), David Recordon (Facebook), Marius Scurtescu (Google), Luke Shepard (Facebook), Filip Skokan (Auth0), Éric Vyncke (as AD), and Robert Wilton (as AD).

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

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.

-28

- Removed unused references, as suggested by Roman Danyliw.

-27

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

-20

- BK comments
- Section 3 Removed WAP
- Section 4. Clarified authorization request object parameters, removed extension parameters from examples
- Section 4. Specifies application/oauth.authz.req+jwt as mime-type fore request objects
- Section 5.2.1 Added reference to Capability URLs
- Section 5.2.3. Added entropy fragment to example request
- Section 8. Replaced "subjectAltName dnsName" with "DNS-ID"
- Section 9. Registers authorization request parameters in JWT Claims Registry.
- Section 9. Registers application/oauth.authz.req in IANA mime-types registry
- Section 10.1. Clarified encrypted request objects are "signed then encrypted" to maintain consistency
- Section 10.2. Clarifies trust between AS and TFP
- Section 10.3. Clarified endpoints subject to the practice
- Section 10.4 Replaced "redirect_uri" to "request_uri"
- Section 10.4. Added reference to RFC 3986 for risks
- Section 10.4.1.d Deleted "do" to maintain grammar flow

- Section 10.4.1, 10.4.2 Replaced "application/jose" to "application/jwt"
- Section 12.1. Extended description for submitting authorization request to TFP to obtain request object
- Section 12.2.2. Replaced per-user Request Object URI with static per-user Request URIs
- Section 13. Combined OAuth WG contributors together
- Section Whole doc Replaced application/jwt with application/oauth.authz.req+jwt

-19

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

-17

- #78 Typos in content-type

-16

- Treated remaining Ben Campbell comments.

-15

- Removed further duplication

-14

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

-13

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

-12

- fixes #62 - Alexey Melnikov Discuss
- fixes #48 - OPSDIR Review : General - delete semicolons after list items
- fixes #58 - DP Comments for the Last Call
- fixes #57 - GENART - Remove "non-normative ... " from examples.
- fixes #45 - OPSDIR Review : Introduction - are attacks discovered or already opened
- fixes #49 - OPSDIR Review : Introduction - Inconsistent colons after initial sentence of list items.
- fixes #53 - OPSDIR Review : 6.2 JWS Signed Request Object - Clarify JOSE Header
- fixes #42 - OPSDIR Review : Introduction - readability of 'and' is confusing
- fixes #50 - OPSDIR Review : Section 4 Request Object - Clarify 'signed, encrypted, or signed and encrypted'
- fixes #39 - OPSDIR Review : Abstract - Explain/Clarify JWS and JWE

- fixed #50 - OPSDIR Review : Section 4 Request Object - Clarify 'signed, encrypted, or signed and encrypted'
- fixes #43 - OPSDIR Review : Introduction - 'properties' sounds awkward and are not exactly 'properties'
- fixes #56 - OPSDIR Review : 12 Acknowledgements - 'contribution is' => 'contribution are'
- fixes #55 - OPSDIR Review : 11.2.2 Privacy Considerations - ' It is in a way' => 'In a way, it is'
- fixes #54 - OPSDIR Review : 11 Privacy Considerations - 'and not specific' => 'and are not specific'
- fixes #51 - OPSDIR Review : Section 4 Request Object - 'It is fine' => 'It is recommended'
- fixes #47 - OPSDIR Review : Introduction - 'over- the- wire' => 'over-the-wire'
- fixes #46 - OPSDIR Review : Introduction - 'It allows' => 'The use of application security' for
- fixes #44 - OPSDIR Review : Introduction - 'has' => 'have'
- fixes #41 - OPSDIR Review : Introduction - missing 'is' before 'typically sent'
- fixes #38 - OPSDIR Review : Section 11 - Delete 'freely accessible' regarding ISO 29100

-11

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

-10

- #20: KM1 -- some wording that is awkward in the TLS section.
- #21: KM2 - the additional attacks against OAuth 2.0 should also have a pointer
- #22: KM3 -- Nit: in the first line of 10.4:
- #23: KM4 -- Mention RFC6973 in Section 11 in addition to ISO 29100
- #24: SECDIR review: Section 4 -- Confusing requirements for sign+encrypt
- #25: SECDIR review: Section 6 -- authentication and integrity need not be provided if the requestor encrypts the token?
- #26: SECDIR Review: Section 10 -- why no reference for JWS algorithms?
- #27: SECDIR Review: Section 10.2 - how to do the agreement between client and server "a priori"?
- #28: SECDIR Review: Section 10.3 - Indication on "large entropy" and "short lifetime" should be indicated
- #29: SECDIR Review: Section 10.3 - Typo
- #30: SECDIR Review: Section 10.4 - typos and missing articles
- #31: SECDIR Review: Section 10.4 - Clearer statement on the lack of endpoint identifiers needed
- #32: SECDIR Review: Section 11 - ISO29100 needs to be moved to normative reference
- #33: SECDIR Review: Section 11 - Better English and Entropy language needed
- #34: Section 4: Typo
- #35: More Acknowledgment
- #36: DP - More precise qualification on Encryption needed.

-09

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

-08

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

-07

- Changed the abbrev to OAuth JAR from oauth-jar.
- Clarified sig and enc methods.
- Better English.
- Removed claims from one of the example.
- Re-worded the URI construction.
- Changed the example to use request instead of request_uri.
- Clarified that Request Object parameters take precedence regardless of request or request_uri parameters were used.
- Generalized the language in 4.2.1 to convey the intent more clearly.
- Changed "Server" to "Authorization Server" as a clarification.
- Stopped talking about request_object_signing_alg.
- IANA considerations now reflect the current status.
- 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.
- Added "older versions of " to the reference to IE URI length limitations.
- Stopped talking about signed or unsigned JWS etc.
- 1.Introduction improved.

-06

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

-05

- More alignment with OpenID Connect.

-04

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

-03

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

-02

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

-01

- 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"

[RFC2119]	Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997.
[RFC3629]	Yergeau, F., "UTF-8, a transformation format of ISO 10646", STD 63, RFC 3629, DOI 10.17487/RFC3629, November 2003.
[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.
[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.
[RFC6749]	Hardt, D., "The OAuth 2.0 Authorization Framework", RFC 6749, DOI 10.17487/RFC6749, October 2012.
[RFC6750]	Jones, M. and D. Hardt, "The OAuth 2.0 Authorization Framework: Bearer Token Usage", RFC 6750, DOI 10.17487/RFC6750, October 2012.
[RFC7159]	Bray, T., "The JavaScript Object Notation (JSON) Data Interchange Format", RFC 7159, DOI 10.17487/RFC7159, March 2014.
[RFC7230]	Fielding, R. and J. Reschke, "Hypertext Transfer Protocol (HTTP/1.1): Message Syntax and Routing", RFC 7230, DOI 10.17487/RFC7230, June 2014.
[RFC7515]	Jones, M., Bradley, J. and N. Sakimura, "JSON Web Signature (JWS)", RFC 7515, DOI 10.17487/RFC7515, May 2015.
[RFC7516]	Jones, M. and J. Hildebrand, "JSON Web Encryption (JWE)", RFC 7516, DOI 10.17487/RFC7516, May 2015.
[RFC7518]	Jones, M., "JSON Web Algorithms (JWA)", RFC 7518, DOI 10.17487/RFC7518, May 2015.
[RFC7519]	Jones, M., Bradley, J. and N. Sakimura, "JSON Web Token (JWT)", RFC 7519, DOI 10.17487/RFC7519, May 2015.
[RFC8141]	Saint-Andre, P. and J. Klensin, "Uniform Resource Names (URNs)", RFC 8141, DOI 10.17487/RFC8141, April 2017.
[RFC8174]	Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017.
[RFC8259]	Bray, T., "The JavaScript Object Notation (JSON) Data Interchange Format", STD 90, RFC 8259, DOI 10.17487/RFC8259, December 2017.
[RFC8414]	Jones, M., Sakimura, N. and J. Bradley, "OAuth 2.0 Authorization Server Metadata", RFC 8414, DOI 10.17487/RFC8414, June 2018.

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.
[CapURLs]	Tennison, J., "Good Practices for Capability URLs", W3C Working Draft, February 2014.
[FETT]	Fett, D., Kusters, R. and G. Schmitz, "A Comprehensive Formal Security Analysis of OAuth 2.0", CCS '16 Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security Pages 1204-1215 , October 2016.
[IANA.JWT.Claims]	IANA, "JSON Web Token Claims"
[IANA.OAuth.Parameters]	IANA, "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.

[RFC2046]	Freed, N. and N. Borenstein, "Multipurpose Internet Mail Extensions (MIME) Part Two: Media Types", RFC 2046, DOI 10.17487/RFC2046, November 1996.
[RFC6819]	Lodderstedt, T., McGloin, M. and P. Hunt, "OAuth 2.0 Threat Model and Security Considerations", RFC 6819, DOI 10.17487/RFC6819, January 2013.
[RFC6838]	Freed, N., Klensin, J. and T. Hansen, "Media Type Specifications and Registration Procedures", BCP 13, RFC 6838, DOI 10.17487/RFC6838, January 2013.
[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.
[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.
[RFC7591]	Richer, J., Jones, M., Bradley, J., Machulak, M. and P. Hunt, "OAuth 2.0 Dynamic Client Registration Protocol", RFC 7591, DOI 10.17487/RFC7591, July 2015.
[RFC8725]	Sheffer, Y., Hardt, D. and M. Jones, "JSON Web Token Best Current Practices", BCP 225, RFC 8725, DOI 10.17487/RFC8725, February 2020.

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/