OAuth Working Group	N. Sakimura, Ed.
Internet-Draft	Nomura Research Institute
Intended status: Standards Track	J. Bradley
Expires: July 22, 2016	Ping Identity
	January 19, 2016

OAuth 2.0 JWT Authorization Request

draft-ietf-oauth-jwsreq-07

Abstract

The authorization request in OAuth 2.0 [RFC6749] utilizes query parameter serialization, which means that parameters are encoded in the URI of the request. This document introduces the ability to send request parameters in form of a JSON Web Token (JWT) instead, which allows the request to be signed and encrypted. using JWT serialization. The request is sent by value or by reference.

Status of This Memo

Copyright Notice

Table of Contents

1. Introduction

The OAuth 2.0 specification [RFC 6749] defines the encoding of requests and responses and in case of the authorization request query parameter serialization has been chosen. 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
```

The encoding in the URI does not allow application layer security with confidentiality and integrity protection to be used. While TLS is used to offer communication security between the client and the resource server, TLS sessions are often terminated prematurely at some middlebox (such as a load balancer). The use of application layer security additionally 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.

Further, the request by reference allows to reduce the over-the-wire overhead.

There are other potential formats that could be used for this purpose instead of JWT. The JWT was chosen because of

1. its close relationship with JSON, which is used as OAuth's response format;
2. its developer friendliness due to its textaual nature;
3. its relative compactness compared to XML;
4. its development status that it is an RFC and so is its associated signing and encryption methods as [RFC7515] and [RFC7516].

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. 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:

1. The request can be signed so that an integrity check can be implemented. If a suitable algorithm is used for the signing, then it will provide verification of the client making the request.
2. The request may be encrypted so that end-to-end confidentiality may be obtained even if in the case TLS connection is terminated at a gateway or a similar device.
3. The request may be signed by a third party attesting that the authorization request is compliant to 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 statically signed by that third party. The client would then send the request along with dynamic parameters such as state. 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 are useful such as:

1. When it is desirable to reduced the size of transmitted request. Since we are using application layer security, it may substantially increase the size of the request particulary in the case of using public key cryptography.
2. Static signature: The client can make a signed Request Object and put it at a place that the Authorization Server can access. This may just be done by a client utility or other process, so that the private key does not have to reside on the client, simplifying programming. Downside of it is that the signed portion just become a token.
3. When the server wants the requests to be cacheable: The request_uri may include a SHA-256 hash of the contents of the resources referenced by the Request URI. With this, the server knows if the resource has changed without fetching it, so it does not have to re-fetch the same content, which is a win as well. This is explained in Section 4.2.

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", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119 [RFC2119].

2. Terminology

For the purposes of this specification, the following terms and definitions apply.

2.1. Request Object

JWT [RFC7519] that holds an OAuth 2.0 authorization request as JWT Claims Set

2.2. Request Object URI

Absolute URI from which the Request Object [request_object] can be obtained

3. Request Object

A Request Object [request_object] is used to provide authorization request parameters for an OAuth 2.0 authorization request. It contains OAuth 2.0 [RFC6749] authorization request parameters including extension parameters. The parameters are represented as the JWT claims. Parameter names and string values MUST be included as JSON strings. Numerical values MUST be included as JSON numbers. It MAY include any extension parameters. This JSON [RFC7159] constitutes the JWT Claims Set [RFC7519]. The JWS Claims Set is then signed, encrypted, 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.

To encrypt, JWE [RFC7516] is used. Note that JWE is always integrity protected, so if only integrity protection is desired, JWS signature is not needed.

It can also be signed then encrypted. This is sometimes desired to reduced the repudiation risk from the point of view of the receiver. In this case, it MUST be signed then encrypted, with the result being a Nested JWT, as defined in JWT [RFC7519].

The Authorization Request Object may be sent by value as described in Section 4.1 or by reference as described in Section 4.2.

REQUIRED OAuth 2.0 Authorization Request parameters that are not included in the Request Object MUST be sent as query parameters. If a required parameter is missing from both the query parameters and the Request Object, the request is malformed.

request and request_uri parameters MUST NOT be included in Request Objects.

If the parameter exists in both the query string and the Authorization Request Object, the values in the Request Object take precedence. This means that if it intends to use a cached request object, it cannot include parameters such as state that are expected to differ in every request. It is fine to include them in the request object if it is going to be prepared afresh every time.

The following is a non-normative example of the Claims in a Request Object before base64url encoding and signing. Note that it includes extension variables such as "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.ew0KICJpc3MiOiAiczZCaGRSa3
F0MyIsDQogImF1ZCI6ICJodHRwczovL3NlcnZlci5leGFtcGxlLmNvbSIsDQogInJl
c3BvbnNlX3R5cGUiOiAiY29kZSBpZF90b2tlbiIsDQogImNsaWVudF9pZCI6ICJzNk
JoZFJrcXQzIiwNCiAicmVkaXJlY3RfdXJpIjogImh0dHBzOi8vY2xpZW50LmV4YW1w
bGUub3JnL2NiIiwNCiAic2NvcGUiOiAib3BlbmlkIiwNCiAic3RhdGUiOiAiYWYwaW
Zqc2xka2oiLA0KICJub25jZSI6ICJuLTBTNl9XekEyTWoiLA0KICJtYXhfYWdlIjog
ODY0MDAsDQogImNsYWltcyI6IA0KICB7DQogICAidXNlcmluZm8iOiANCiAgICB7DQ
ogICAgICJnaXZlbl9uYW1lIjogeyJlc3NlbnRpYWwiOiB0cnVlfSwNCiAgICAgIm5p
Y2tuYW1lIjogbnVsbCwNCiAgICAgImVtYWlsIjogeyJlc3NlbnRpYWwiOiB0cnVlfS
wNCiAgICAgImVtYWlsX3ZlcmlmaWVkIjogeyJlc3NlbnRpYWwiOiB0cnVlfSwNCiAg
ICAgInBpY3R1cmUiOiBudWxsDQogICAgfSwNCiAgICJpZF90b2tlbiI6IA0KICAgIH
sNCiAgICAgImdlbmRlciI6IG51bGwsDQogICAgICJiaXJ0aGRhdGUiOiB7ImVzc2Vu
dGlhbCI6IHRydWV9LA0KICAgICAiYWNyIjogeyJ2YWx1ZXMiOiBbInVybjptYWNlOm
luY29tbW9uOmlhcDpzaWx2ZXIiXX0NCiAgICB9DQogIH0NCn0.nwwnNsk1-Zkbmnvs
F6zTHm8CHERFMGQPhos-EJcaH4Hh-sMgk8ePrGhw_trPYs8KQxsn6R9Emo_wHwajyF
KzuMXZFSZ3p6Mb8dkxtVyjoy2GIzvuJT_u7PkY2t8QU9hjBcHs68PkgjDVTrG1uRTx
0GxFbuPbj96tVuj11pTnmFCUR6IEOXKYr7iGOCRB3btfJhM0_AKQUfqKnRlrRscc8K
ol-cSLWoYE9l5QqholImzjT_cMnNIznW9E7CDyWXTsO70xnB4SkG6pXfLSjLLlxmPG
iyon_-Te111V8uE83IlzCYIb_NMXvtTIVc1jpspnTSD7xMbpL-2QgwUsAlMGzw
```

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":"y9Lqv4fCp6Ei-u2-ZCKq83YvbFEk6JMs_pSj76eMkddWRuWX2aBKGHAtKlE5P
     7_vn__PCKZWePt3vGkB6ePgzAFu08NmKemwE5bQI0e6kIChtt_6KzT5OaaXDF
     I6qCLJmk51Cc4VYFaxgqevMncYrzaW_50mZ1yGSFIQzLYP8bijAHGVjdEFgZa
     ZEN9lsn_GdWLaJpHrB3ROlS50E45wxrlg9xMncVb8qDPuXZarvghLL0HzOuYR
     adBJVoWZowDNTpKpk2RklZ7QaBO7XDv3uR7s_sf2g-bAjSYxYUGsqkNA9b3xV
     W53am_UZZ3tZbFTIh557JICWKHlWj5uzeJXaw",
 "e":"AQAB"
}
```

4. Authorization Request

The client constructs the authorization request URI by adding one of the following parameters but not both to the query component of the authorization endpoint URI using the application/x-www-form-urlencoded format:

request
: The Request Object [aro] that holds authorization request parameters stated in the section 4 of OAuth 2.0 [RFC6749].

request_uri
: The absolute URL that points to the Request Object [aro] that holds authorization request parameters stated in the section 4 of OAuth 2.0 [RFC6749].

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?request=eyJhbG..AlMGzw HTTP/1.1
Host: server.example.com
```

The value for the request parameter is abbreviated for brevity.

The authorization request object MAY be signed AND/OR encrypted.

When the Request Object is used, the OAuth 2.0 request parameter values contained in the JWT supersede those passed using the OAuth 2.0 request syntax. However, parameters MAY also be passed using the OAuth 2.0 request syntax even when a Request Object is used; this would typically be done to enable a cached, pre-signed (and possibly pre-encrypted) Request Object value to be used containing the fixed request parameters, while parameters that can vary with each request, such as state and nonce, are passed as OAuth 2.0 parameters.

4.1. 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 a non-normative example of an Authorization Request using the request parameter (with line wraps within values for display purposes only):

```
https://server.example.com/authorize?
  response_type=code%20id_token
  &client_id=s6BhdRkqt3
  &redirect_uri=https%3A%2F%2Fclient.example.org%2Fcb
  &scope=openid
  &state=af0ifjsldkj
  &nonce=n-0S6_WzA2Mj
  &request=eyJhbGciOiJSUzI1NiIsImtpZCI6ImsyYmRjIn0.ew0KICJpc3MiOiA
  iczZCaGRSa3F0MyIsDQogImF1ZCI6ICJodHRwczovL3NlcnZlci5leGFtcGxlLmN
  vbSIsDQogInJlc3BvbnNlX3R5cGUiOiAiY29kZSBpZF90b2tlbiIsDQogImNsaWV
  udF9pZCI6ICJzNkJoZFJrcXQzIiwNCiAicmVkaXJlY3RfdXJpIjogImh0dHBzOi8
  vY2xpZW50LmV4YW1wbGUub3JnL2NiIiwNCiAic2NvcGUiOiAib3BlbmlkIiwNCiA
  ic3RhdGUiOiAiYWYwaWZqc2xka2oiLA0KICJub25jZSI6ICJuLTBTNl9XekEyTWo
  iLA0KICJtYXhfYWdlIjogODY0MDAsDQogImNsYWltcyI6IA0KICB7DQogICAidXN
  lcmluZm8iOiANCiAgICB7DQogICAgICJnaXZlbl9uYW1lIjogeyJlc3NlbnRpYWw
  iOiB0cnVlfSwNCiAgICAgIm5pY2tuYW1lIjogbnVsbCwNCiAgICAgImVtYWlsIjo
  geyJlc3NlbnRpYWwiOiB0cnVlfSwNCiAgICAgImVtYWlsX3ZlcmlmaWVkIjogeyJ
  lc3NlbnRpYWwiOiB0cnVlfSwNCiAgICAgInBpY3R1cmUiOiBudWxsDQogICAgfSw
  NCiAgICJpZF90b2tlbiI6IA0KICAgIHsNCiAgICAgImdlbmRlciI6IG51bGwsDQo
  gICAgICJiaXJ0aGRhdGUiOiB7ImVzc2VudGlhbCI6IHRydWV9LA0KICAgICAiYWN
  yIjogeyJ2YWx1ZXMiOiBbInVybjptYWNlOmluY29tbW9uOmlhcDpzaWx2ZXIiXX0
  NCiAgICB9DQogIH0NCn0.nwwnNsk1-ZkbmnvsF6zTHm8CHERFMGQPhos-EJcaH4H
  h-sMgk8ePrGhw_trPYs8KQxsn6R9Emo_wHwajyFKzuMXZFSZ3p6Mb8dkxtVyjoy2
  GIzvuJT_u7PkY2t8QU9hjBcHs68PkgjDVTrG1uRTx0GxFbuPbj96tVuj11pTnmFC
  UR6IEOXKYr7iGOCRB3btfJhM0_AKQUfqKnRlrRscc8Kol-cSLWoYE9l5QqholImz
  jT_cMnNIznW9E7CDyWXTsO70xnB4SkG6pXfLSjLLlxmPGiyon_-Te111V8uE83Il
  zCYIb_NMXvtTIVc1jpspnTSD7xMbpL-2QgwUsAlMGzw
```

4.2. 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 at the specified URL, rather than passed by value.

When the request_uri parameter is used, the OAuth 2.0 authorization request parameter values contained in the referenced JWT supersede those passed using the OAuth 2.0 request syntax. However, parameters MAY also be passed using the OAuth 2.0 request syntax even when a request_uri is used; this would typically be done to enable a cached, pre-signed (and possibly pre-encrypted) Request Object value to be used containing the fixed request parameters, while parameters that can vary with each request, such as state and nonce, are passed as OAuth 2.0 parameters.

Servers MAY cache the contents of the resources referenced by Request URIs. If the contents of the referenced resource could ever change, the URI SHOULD include the base64url encoded SHA-256 hash as defined in FIPS180-2 [FIPS180-2] of the referenced resource contents as the fragment component of the URI. If the fragment value used for a URI changes, that signals the server that any cached value for that URI with the old fragment value is no longer valid.

The entire Request URI MUST NOT exceed 512 ASCII characters. There are three reasons for this restriction.

1. Many WAP / feature phones do not accept large payloads. The restriction are typically either 512 or 1024 ASCII characters.
2. The maximum URL length supported by older versions of Internet Explorer is 2083 ASCII characters.
3. On a slow connection such as 2G mobile connection, a large URL would cause the slow response and using such is not advisable from the user experience point of view.

The contents of the resource referenced by the URL MUST be a Request Object. The scheme used in the request_uri value MUST be https, unless the target Request Object is signed in a way that is verifiable by the Authorization Server. The request_uri value MUST be reachable by the Authorization Server, and SHOULD be reachable by the Client.

The following is a non-normative 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.ew0KICJpc3MiOiAiczZCaGRSa3
F0MyIsDQogImF1ZCI6ICJodHRwczovL3NlcnZlci5leGFtcGxlLmNvbSIsDQogInJl
c3BvbnNlX3R5cGUiOiAiY29kZSBpZF90b2tlbiIsDQogImNsaWVudF9pZCI6ICJzNk
JoZFJrcXQzIiwNCiAicmVkaXJlY3RfdXJpIjogImh0dHBzOi8vY2xpZW50LmV4YW1w
bGUub3JnL2NiIiwNCiAic2NvcGUiOiAib3BlbmlkIiwNCiAic3RhdGUiOiAiYWYwaW
Zqc2xka2oiLA0KICJub25jZSI6ICJuLTBTNl9XekEyTWoiLA0KICJtYXhfYWdlIjog
ODY0MDAsDQogImNsYWltcyI6IA0KICB7DQogICAidXNlcmluZm8iOiANCiAgICB7DQ
ogICAgICJnaXZlbl9uYW1lIjogeyJlc3NlbnRpYWwiOiB0cnVlfSwNCiAgICAgIm5p
Y2tuYW1lIjogbnVsbCwNCiAgICAgImVtYWlsIjogeyJlc3NlbnRpYWwiOiB0cnVlfS
wNCiAgICAgImVtYWlsX3ZlcmlmaWVkIjogeyJlc3NlbnRpYWwiOiB0cnVlfSwNCiAg
ICAgInBpY3R1cmUiOiBudWxsDQogICAgfSwNCiAgICJpZF90b2tlbiI6IA0KICAgIH
sNCiAgICAgImdlbmRlciI6IG51bGwsDQogICAgICJiaXJ0aGRhdGUiOiB7ImVzc2Vu
dGlhbCI6IHRydWV9LA0KICAgICAiYWNyIjogeyJ2YWx1ZXMiOiBbInVybjptYWNlOm
luY29tbW9uOmlhcDpzaWx2ZXIiXX0NCiAgICB9DQogIH0NCn0.nwwnNsk1-Zkbmnvs
F6zTHm8CHERFMGQPhos-EJcaH4Hh-sMgk8ePrGhw_trPYs8KQxsn6R9Emo_wHwajyF
KzuMXZFSZ3p6Mb8dkxtVyjoy2GIzvuJT_u7PkY2t8QU9hjBcHs68PkgjDVTrG1uRTx
0GxFbuPbj96tVuj11pTnmFCUR6IEOXKYr7iGOCRB3btfJhM0_AKQUfqKnRlrRscc8K
ol-cSLWoYE9l5QqholImzjT_cMnNIznW9E7CDyWXTsO70xnB4SkG6pXfLSjLLlxmPG
iyon_-Te111V8uE83IlzCYIb_NMXvtTIVc1jpspnTSD7xMbpL-2QgwUsAlMGzw
```

4.2.1. URL Referencing the Request Object

The Client stores the Request Object resource either locally or remotely at a URL the Authorization Server can access. This URL is the Request URI, request_uri.

It is possible for the Request Object to include values that is to be revealed only to the Authorization Server. As such, the request_uri MUST have appropriate entropy for its lifetime. It is RECOMMENDED that it be removed if it is known that it will not be used again or after a reasonable timeout unless access control measures are taken.

The following is a non-normative example of a Request URI value (with line wraps within values for display purposes only):

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

4.2.2. Request using the "request_uri" Request Parameter

The Client sends the Authorization Request to the Authorization Endpoint.

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

```
https://server.example.com/authorize?
  response_type=code%20id_token
  &client_id=s6BhdRkqt3
  &request_uri=https%3A%2F%2Fclient.example.org%2Frequest.jwt
  %23GkurKxf5T0Y-mnPFCHqWOMiZi4VS138cQO_V7PZHAdM
  &state=af0ifjsldkj
```

4.2.3. 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 already cached, and parse it to recreate the Authorization Request parameters.

Note that the client SHOULD use a unique URI for each request utilizing distinct parameters, or otherwise prevent the Authorization Server from caching the request_uri.

The following is a non-normative example of this fetch process:

```
GET /request.jwt HTTP/1.1
Host: client.example.org
```

5. Validating JWT-Based Requests

5.1. Encrypted Request Object

The Authorization Server MUST decrypt the JWT in accordance with the JSON Web Encryption [RFC7516] specification. If the result is a signed request object, signature validation MUST be performed as defined in Section 5.2 as well.

The Authorization Server MUST return an error if decryption fails.

5.2. Signed Request Object

To perform Signature Validation, the alg Header Parameter in the JOSE Header MUST match the value of the pre-registered algorithm. The signature MUST be validated against the appropriate key for that client_id and algorithm.

The Authorization Server MUST return an error if signature validation fails.

5.3. Request Parameter Assembly and Validation

The Authorization Server MUST assemble the set of Authorization Request parameters to be used from the Request Object value and the OAuth 2.0 Authorization Request parameters (minus the request or request_uri parameters). If the same parameter exists both in the Request Object and the OAuth Authorization Request parameters, the parameter in the Request Object is used. Using the assembled set of Authorization Request parameters, the Authorization Server then validates the request as specified in OAuth 2.0 [RFC6749].

6. 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 use of the request_uri parameter.

7. IANA Considerations

This specification requests no actions by IANA.

8. Security Considerations

In addition to the all the security considerations discussed in OAuth 2.0 [RFC6819], the following security considerations SHOULD be taken into account.

When sending the authorization request object through request parameter, it SHOULD be signed with then considered appropriate algorithm using [RFC7515]. The alg=none SHOULD NOT be used in such a case.

If the request object contains personally identifiable or sensitive information, the "request_uri" MUST be of one-time use and MUST have large enough entropy deemed necessary with applicable security policy. For higher security requirement, using [RFC7516] is strongly recommended.

9. Acknowledgements

Follwoing people contributed to the creation of this document in OAuth WG. (Affiliations at the time of the contribution is used.)

Sergey Beryozkin, Brian Campbell (Ping Identity), Michael B. Jones (Microsoft), Jim Manico, Axel Nenker(DT), (add yourself).

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

Brian Campbell (Ping Identity), George Fletcher (AOL), Ryo Itou (Yahoo! Japan), Edmund Jay (Illumila), Michael B. Jones (Microsoft), Breno de Medeiros (Google), Hideki Nara (TACT), Justin Richer (MITRE), (add yourself).

In addition following people contributed to this and previous versions through The OAuth Working Group.

Dirk Balfanz (Google), James H. Manger (Telstra), John Panzer (Google), David Recordon (Facebook), Marius Scurtescu (Google), Luke Shepard (Facebook), (add yourself).

10. Revision History

-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 takes 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 contributers 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.

11. References

11.1. Normative References

[FIPS180-2] U.S. Department of Commerce and National Institute of Standards and Technology, "Secure Hash Signature Standard", FIPS 180-2, August 2002.

Defines Secure Hash Algorithm 256 (SHA256)

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997.

[RFC5246] Dierks, T. and E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.2", RFC 5246, DOI 10.17487/RFC5246, August 2008.

[RFC6749] Hardt, D., "The OAuth 2.0 Authorization Framework", RFC 6749, DOI 10.17487/RFC6749, October 2012.

[RFC6819] Lodderstedt, T., McGloin, M. and P. Hunt, "OAuth 2.0 Threat Model and Security Considerations", RFC 6819, DOI 10.17487/RFC6819, January 2013.

[RFC7159] Bray, T., "The JavaScript Object Notation (JSON) Data Interchange Format", RFC 7159, DOI 10.17487/RFC7159, March 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.

11.2. Informative References

[OpenID.Core] Sakimura, N., Bradley, J., Jones, M., de Medeiros, B. and C. Mortimore, "OpenID Connect Core 1.0", February 2014.

Authors' Addresses

Nat Sakimura (editor)
Nomura Research Institute
1-6-5 Marunouchi, Marunouchi Kitaguchi Bldg.
Chiyoda-ku, Tokyo 100-0005
Japan
Phone: +81-3-5533-2111
EMail: n-sakimura@nri.co.jp
URI: http://nat.sakimura.org/

John Bradley
Ping Identity
Casilla 177, Sucursal Talagante
Talagante, RM
Chile
Phone: +44 20 8133 3718
EMail: ve7jtb@ve7jtb.com
URI: http://www.thread-safe.com/