

OAuth Working Group  
Internet-Draft  
Intended status: Standards Track  
Expires: January 6, 2016

N. Sakimura, Ed.  
Nomura Research Institute  
J. Bradley  
Ping Identity  
July 05, 2015

**Request by JWS ver.1.0 for OAuth 2.0**  
**draft-ietf-oauth-jwsreq-03**

Abstract

The authorization request in OAuth 2.0 utilizes query parameter serialization. This specification defines the authorization request using JWT serialization. The request is sent through "request" parameter or by reference through "request\_uri" parameter that points to the JWT, allowing the request to be optionally signed and encrypted.

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 6, 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

<a href="#">1.</a>	<a href="#">Introduction</a>	<a href="#">2</a>
<a href="#">1.1.</a>	<a href="#">Requirements Language</a>	<a href="#">3</a>
<a href="#">2.</a>	<a href="#">Terminology</a>	<a href="#">3</a>
<a href="#">2.1.</a>	<a href="#">Request Object</a>	<a href="#">3</a>
<a href="#">2.2.</a>	<a href="#">Request Object URI</a>	<a href="#">3</a>
<a href="#">3.</a>	<a href="#">Request Object</a>	<a href="#">4</a>
<a href="#">4.</a>	<a href="#">Request Object URI</a>	<a href="#">5</a>
<a href="#">5.</a>	<a href="#">Authorization Request</a>	<a href="#">6</a>
<a href="#">6.</a>	<a href="#">Authorization Server Response</a>	<a href="#">7</a>
<a href="#">7.</a>	<a href="#">IANA Considerations</a>	<a href="#">7</a>
<a href="#">8.</a>	<a href="#">Security Considerations</a>	<a href="#">7</a>
<a href="#">9.</a>	<a href="#">Acknowledgements</a>	<a href="#">8</a>
<a href="#">10.</a>	<a href="#">Revision History</a>	<a href="#">8</a>
<a href="#">11.</a>	<a href="#">References</a>	<a href="#">8</a>
<a href="#">11.1.</a>	<a href="#">Normative References</a>	<a href="#">8</a>
<a href="#">11.2.</a>	<a href="#">Informative References</a>	<a href="#">9</a>
	<a href="#">Authors' Addresses</a>	<a href="#">9</a>

## [1.](#) Introduction

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 [\[RFC7519\]](#) can be passed to the authorization endpoint by reference, in which case the parameter "request\_uri" is used instead of the "request".

Using [\[RFC7519\]](#) as the request encoding instead of query parameters has several advantages:

1. The request may be signed so that integrity check may be implemented. If a suitable algorithm is used for the signing, then non-repudiation property may be obtained in addition.
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.

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



1. When it is detected that the User Agent does not support long URLs: Some extensions may extend the URL. For example, the client might want to send a public key with the request.
2. Static signature: The client may make a signed Request Object and put it at the place where 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.
3. When the server wants the requests to be cache-able: The request\_uri may include a sha256 hash of the file, as defined in FIPS180-2 [[FIPS180-2](#)], the server knows if the file has changed without fetching it, so it does not have to re-fetch a same file, which is a win as well.
4. When the client wants to simplify the implementation without compromising the security. If the request parameters go through the Browser, they may be tampered in the browser even if TLS was used. This implies we need to have signature on the request as well. However, if HTTPS "request\_uri" was used, it is not going to be tampered, thus we now do not have to sign the request. This simplifies the implementation.

This capability is in use by OpenID Connect [[openid\\_ab](#)].

### **[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 OAuth 2.0 authorization requests as JWT Claims Set

### **[2.2.](#) Request Object URI**

Absolute URI from which the Request Object ([Section 2.1](#)) can be obtained



### 3. Request Object

A Request Object ([Section 2.1](#)) is used to provide authorization request parameters for OAuth 2.0 authorization request. It contains OAuth 2.0 [\[RFC6749\]](#) authorization request parameters including extension parameters. It is a JSON Web Signature (JWS) [\[RFC7515\]](#) signed JWT [\[RFC7519\]](#). The parameters are included as the top-level members of JSON [\[RFC7159\]](#). 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 [\[RFC7519\]](#) Claims Set.

The Request Object MAY be signed or unsigned (plaintext). When it is plaintext, this is indicated by use of the "none" algorithm JWA [\[RFC7518\]](#) in the JWS header. 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 Request Object MAY also be encrypted using JWE [\[RFC7516\]](#) after signing, with nesting performed in the same manner as specified for JWT [\[RFC7519\]](#). The Authorization Request Object MAY alternatively be sent by reference using "request\_uri" parameter.

REQUIRED OAuth 2.0 Authorization Request parameters that are not included in the Request Object MUST be sent as a query parameter. If a required parameter is not present in neither the query parameter nor the Request Object, it forms a malformed request.

If the parameter exists in both the query string and the Authorization Request Object, the values in the Request Object takes precedence. This means that if it intends to use a cached request object, it cannot include such parameters like "state" that is 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.

Following is the example of the JSON that constitutes the [\[RFC7519\]](#) Claims Set.

```
{
  "redirect_url":"https://example.com/rp/endpoint_url",
  "cliend_id":"http://example.com/rp/"
}
```

The following is a non-normative example of a [\[RFC7519\]](#) encoded authorization request object. It includes extension variables such as "nonce", "userinfo", and "id\_token". Note that the line wraps within the values are for display purpose only:



JWT algorithm = HS256

HMAC HASH Key = 'aaa'

JSON Encoded Header = '{"alg":"HS256","typ":"JWT"}'

JSON Encoded Payload = '{"response\_type":"code id\_token",  
 "client\_id":"s6BhdRkqt3",  
 "redirect\_uri":"https://client.example.com/cb",  
 "scope":"openid profile",  
 "state":"af0ifjsldkj",  
 "nonce":"n-0S6\_WzA2Mj",  
 "userinfo":{"claims":{"name":null,"nickname":{"optional":true},  
 "email":null,"verified":null,  
 "picture":{"optional":true}},"format":"signed"},  
 "id\_token":{"max\_age":86400,"iso29115":"2"}}'

JWT = eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJyZXNwb25zZV90eXBliJoiY29kZSBpZF90b2t1biIsImNsaWVudF9pZCI6InM2QmhkUmtxdDMiLCJyZWRpcmVjdF91cmkiOiJodHRwczpcL1wvY2xpZW50LmV4YW1wbGUuY29tXC9jYiIsInNjb3BlIjoib3BlbmlkIHB5b2ZpbGUiLCJzdGF0ZSI6ImFmMGImanNsZGtqIiwidXNlcmZm8iOnsiY2xhaw1zIjpw7Im5hbWUiOm51bGwsIm5pY2tuYW1lIjpw7Im9wdGlvbmFsIjpwcnVlfSwiZW1haWwiOm51bGwsInZlcm1maWVkiJpudWxsLCJwaWN0dXJlIjpw7Im9wdGlvbmFsIjpwcnVlfx0sImZvcmlhdCI6InNpZ25lZCJ9LCJpZF90b2t1biI6eyJtYXhfYWdlIjo4NjQwMCwiaXNVMjxkMTUiOiIyIn19.20iqRgrbrHKA1FZ5p\_7bc\_RSdTbH-wo\_Agk-ZRpD3wY

#### 4. Request Object URI

Instead of sending the Request Object in an OAuth 2.0 authorization request directly, this specification allows it to be obtained from the Request Object URI. Using this method has an advantage of reducing the request size, enabling the caching of the Request Object, and generally not requiring integrity protection through a cryptographic operation on the Request Object if the channel itself is protected.

The Request Object URI is sent as a part of the OAuth Authorization Request as the value for the parameter called "request\_uri". How the Request Object is registered at Request Object URI is out of scope of this specification, but it MUST be done in a protected channel.

NOTE: the Request Object MAY be registered at the Authorization Server at the client registration time.

When the Authorization Server obtains the Request Object from Request Object URI, it MUST do so over a protected channel. If it is obtained from a remote server, it SHOULD use either HTTP over TLS 1.2 as defined in [RFC5246] AND/OR [RFC7515] with the algorithm considered appropriate at the time.





When sending the request by "request\_uri", the client MAY provide the sha256 hash as defined in FIPS180-2 [[FIPS180-2](#)] of the Request Object as the fragment to it to assist the cache utilization decision of the Authorization Server.

## 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 3](#)) that holds authorization request parameters stated in the [section 4](#) of OAuth 2.0 [[RFC6749](#)].

request\_uri REQUIRED unless "request" is specified. The absolute URL that points to the Request Object ([Section 3](#)) that holds authorization request parameters stated in the [section 4](#) of OAuth 2.0 [[RFC6749](#)].

state RECOMMENDED. OAuth 2.0 [[RFC6749](#)] state.

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 (line breaks are for display purposes only):

```
GET /authorize?request_uri=https%3A%2F%2Fclient%2Eexample%2Ecom%2Fcb HTTP/1.1
Host: server.example.com
```

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

Upon receipt of "request\_uri" in the request, the authorization server MUST send a GET request to the "request\_uri" to retrieve the authorization request object unless it is already cached at the Authorization Server.

If the response was signed AND/OR encrypted, it has to be decoded accordingly before being processed.

Then, the Authorization Server MUST reconstruct the complete client request from the original HTTP request and the content of the request object. Then, the process continues as described in [Section 3](#) of 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 defines additional 'error' values as follows:

`invalid_request_uri` The provided `request_uri` was not available.

`invalid_request_format` The Request Object format was invalid.

`invalid_request_params` The parameter set provided in the Request Object was invalid.

## **7. IANA Considerations**

This document registers following error strings to the OAuth Error Registry.

`invalid_request_uri` The provided `request_uri` was not available.

`invalid_request_format` The Request Object format was invalid.

`invalid_request_params` The parameter set provided in the Request Object was invalid.

## **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**

Following people contributed to creating this document through the OpenID Connect 1.0 [[openid\\_ab](#)].

Breno de Medeiros (Google), Hideki Nara (TACT), John Bradley ( Ping Identity) <author>, Nat Sakimura (NRI) <author/editor>, Ryo Itou (Yahoo! Japan), George Fletcher (AOL), Justin Richer (MITRE), Edmund Jay (Illumila), (add yourself).

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

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

## **10. Revision History**

-03

- o Fixed the non-normative description about the advantage of static signature.
- o Changed the requement for the parameter values in the request iteself 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.

## **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](#), March 1997.
- [RFC5246] Dierks, T. and E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.2", [RFC 5246](#), August 2008.
- [RFC6749] Hardt, D., "The OAuth 2.0 Authorization Framework", [RFC 6749](#), October 2012.
- [RFC6819] Lodderstedt, T., McGloin, M., and P. Hunt, "OAuth 2.0 Threat Model and Security Considerations", [RFC 6819](#), January 2013.
- [RFC7159] Bray, T., "The JavaScript Object Notation (JSON) Data Interchange Format", [RFC 7159](#), March 2014.
- [RFC7515] Jones, M., Bradley, J., and N. Sakimura, "JSON Web Signature (JWS)", [RFC 7515](#), May 2015.
- [RFC7516] Jones, M. and J. Hildebrand, "JSON Web Encryption (JWE)", [RFC 7516](#), May 2015.
- [RFC7518] Jones, M., "JSON Web Algorithms (JWA)", [RFC 7518](#), May 2015.
- [RFC7519] Jones, M., Bradley, J., and N. Sakimura, "JSON Web Token (JWT)", [RFC 7519](#), May 2015.

### **11.2. Informative References**

- [openid\_ab]  
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](mailto:ve7jtb@ve7jtb.com)

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