Internet Engineering Task Force
Internet-Draft
Intended status: Standards Track
Expires: January 5, 2015

N. Sakimura, Ed.
Nomura Research Institute
J. Bradley
Ping Identity
July 4, 2014

# Request by JWS ver.1.0 for OAuth 2.0
# draft-sakimura-oauth-requrl-05

Abstract

The authorization request in OAuth 2.0 utilizes query parameter serizalization. This specification defines the authorization request using JWT serialization. The request is sent thorugh "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 5, 2015.

Table of Contents



## 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) [JWT] whose body holds the JSON encoded OAuth 2.0 authorization request parameters. 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:

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 is useful such as:

1. When it is detected that the User Agent does not suport long URLs: It is entirely possible that 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 on the client. 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 cacheable: 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.

## 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 [JWT] that holds OAuth 2.0 authorization requests as JSON object in its body

## 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) [JWS] signed JWT [JWT] . The parameters are included as the top level members of JSON [RFC4627]. 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 [RFC4627] constitues the body of the [JWT].

The Request Object MAY be signed or unsigned (plaintext). When it is plaintext, this is indicated by use of the "none" algorithm [JWA] 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 [JWT] specification.

The Request Object MAY also be encrypted using JWE [JWE] after signing, with nesting performed in the same manner as specified for JWTs [JWT]. 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 or the Request Object, it forms a malformed request.

If the parameter exists both in the query string and the Authorization Request Object, they MUST exactly match.

Following is the example of the JSON which consitutes the body of the [JWT].

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

The following is a non-normative example of a [JWT] 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.eyJyZXNwb25zZV90eXBlIjoiY29kZ
    SBpZF90b2tlbiIsImNsaWVudF9pZCI6InM2QmhkUmtxdDMiLCJyZWRpcmVjdF91cmkiO
    iJodHRwczpcL1wvY2xpZW50LmV4YW1wbGUuY29tXC9jYiIsInNjb3BlIjoib3BlbmlkI
    HByb2ZpbGUiLCJzdGF0ZSI6ImFmMGlmanNsZGtqIiwidXNlcmluZm8iOnsiY2xhaW1zI
    jp7Im5hbWUiOm51bGwsIm5pY2tuYW1lIjp7Im9wdGlvbmFsIjp0cnVlfSwiZW1haWwiO
    m51bGwsInZlcmlmaWVkIjpudWxsLCJwaWN0dXJlIjp7Im9wdGlvbmFsIjp0cnVlfX0sI
    mZvcm1hdCI6InNpZ25lZCJ9LCJpZF90b2tlbiI6eyJtYXhfYWdlIjo4NjQwMCwiaXNvM
    jkxMTUiOiIyIn19.2OiqRgrbrHkA1FZ5p_7bc_RSdTbH-wo_Agk-ZRpD3wY
```

## 4. Request Object URI

Instead of sending the Request Object in a 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 [RFC5246] AND/OR [JWS] 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 autorization 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[JWS]. 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 [JWE] 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 (MGI1), (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. References

### 10.1. 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)

[JWA] Jones, M., "JSON Web Algorithms (JWA)", March 2011.

[JWE] Jones, M., "JSON Web Encryption (JWE)", March 2011.

[JWS] Jones, M., Balfanz, D., Bradley, J., Goland, Y., Panzer, J., Sakimura, N., and P. Tarjan, "JSON Web Signature (JWS)", April 2011.

[JWT] Jones, M., Balfanz, D., Bradley, J., Goland, Y., Panzer, J., Sakimura, N., and P. Tarjan, "JSON Web Token", July 2011.

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

[RFC4627] Crockford, D., "The application/json Media Type for JavaScript Object Notation (JSON)", RFC 4627, July 2006.

[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.

### 10.2. Informative References

[openid_ab] openid-specs-ab@openid.net, , "OpenID Connect Core 1.0", November 2013.

## 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

John Bradley
Ping Identity

Email: ve7jtb@ve7jtb.com