Draft	N. Sakimura, Ed.
	NRI
	D. Recordon
	Facebook
	J. Bradeley
	Protiviti Government Services
	B. de Madeiros
	Google
	M. Jones
	Microsoft
	E. Jay, Ed.
	MGI1
	May 1, 2011

TOC

OpenID Connect Core 1.0 - draft 05

Abstract

OpenID Connect is an identity framework that provides authentication, authorization, and attribute transmition capability. It allows third party attested claims from distributed sources. The specification suite builts on OAuth 2.0 and consists of Building Blocks (Core, JSON Web Token, JSON Web Signatures, JSON WEB Encryption, JSON Web Keys, Simple Web Discovery), Protocol Bindings (e.g, Artifact Binding, Web App Binding, User Agent Binding) and Extensions. This specification is the "Core" of the suite that defines the messages used and abstract flow which will be further constrained or extended in the companion specifications in the suite.

Table of Contents

1. Requirements Notation and Conventions

TOC

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 **[RFC2119]** .

Throughout this document, values are quoted to indicate that they are to be taken literally. When using these values in protocol messages, the quotes MUST NOT be used as part of the value.

2. Terminology

TOC

In addition to "Access Token", "Refresh Token", "Authorization Code", "Authorization Grant", "Authorization Server", "Authorization Endpoint", "Client", "Client Identifier", "Client Secret", "Protected Resource", "Resource Owner", "Resource Server", "Token Endpoint" defined in **OAuth 2.0** [RFC5849], this specification defines the following terms:

Assertion
: A set of Claims about the End-User which is attested by the OP and Resource Servers.

Authentication
: An act of verifying End-User's identity through the verification of the credential.

Base64url
: Base 64 Encoding **[RFC3548]** with URL and Filename Safe Alphabet without padding.

Claims
: A piece of information about an Entity that a Claims Provider asserts about that Entity.

Entity
: Something that has separate and distinct existence and that can be identified in context.

End-user
: A human resource owner.

OpenID Provider (OP)
: Authorization Servers that are able to support OpenID Connect Messages.

OP Endpoints
: End-User Authentication, Authorization, and Token Endpoint.

OpenID Request Object
: A JSON object that holds the request variables. It holds OpenID request variables. It MAY also contain other OAuth parameters for the request signing purpose, in which case the parameter values MUST match with the OAuth request parameters.

Relying Party (RP)
: Client and Resource Servers.

RP Endpoints
: The endpoint to which the OP responses are returned through redirect.

UserInfo Endpoint
: A protected resource that when presented with a token by the client returns authorized information about the current user.

3. Overview

TOC

OpenID Connect protocol in abstract follows the following steps.

1. The Client sends a request to the Server's End-User Authorization Endpoint.
2. The Server authenticates the user and obtains appropriate authorization.
3. The Server responds with access_token and a few other variables.
4. The Client sends a request with the access_token to the Userinfo Endpoint.
5. Userinfo Endpoint returns the additional user information supported by the Server.

Each message may be signed and encrypted. When the message is encrypted, it MUST be signed first then encrypted. This specification only defines the abstract messsage flow and message formats. The actual use MUST be based on one of the companion protocol bindings specifications such as **OpenID Connect Artifact Binding 1.0** [OpenID.AB] or **OpenID Connect Authorization Code Binding 1.0** [OpenID.AC].

4. Messages

TOC

In OpenID Connect protocols in abstract, the process proceeds by the Client interacting with Endpoints. There are number of Endpoints involved.

1. Authorization Endpoint: The Client sends a request to the Server at the Authorization endpoint. The Server then authenticate the End-User to find out if he is eligible to make the authorization. Then, upon the authorization action of the End-User, the Server returns an Authorization Response that includes Authorization Code, `code`. For some Clients, Implicit Grant may be used to obtain `access_token` without using `code`. In this case, `response_type` MUST be set to `token`.
2. Token Endpoint: The Client sends the Access Token Request to the Token Endpoint to obtain Access Token Response which includes `access_token` and `openid` token.
3. UserInfo Endpoint: The `access_token` MAY be sent to the UserInfo Endpoint to obtain user information/assertion/claims about the user by sending a request to the userinfo endpoint.
4. Session Management Endpoints: The `openid` token MAY be sent to these endpoints to manage the session.

Both Request and Response may either be serialized into **Query String serialization** or **JSON** [RFC4627].

TOC

4.1. Authorization Endpoint

Client sends Autorization Request to the Authorization Endpoint to obtain Authorization Response.

TOC

4.1.1. Authorization Request

Section 4.1.1 and 4.2.1 of **OAuth 2.0** [RFC5849] defines OAuth Authorization Request parameters. In this specification, the values to the parameters are defined as follows.

response_type
: The value MUST be set to `code` for requesting an Authorization Code, `token` for requesting an Access Token.

scope
: It MUST include `openid` as one of the string.

In addition, this specification defines following extension parameter.

req
: OPTIONAL. A **JWT** [jwt] encoded OpenID Request Object.

request_uri
: OPTIONAL. A URL that points to the OpenID Request Object.

max_auth_age
: OPTIONAL. Maximum number of seconds from the last authentication that is permissible from the Client.

Following is a non-normative example when they are sent in the query parameters serialization.

```
GET /authorize?scope=openid&response_type=code
&client_id=s6BhdRkqt3
&redirect_uri=https%3A%2F%2Fclient%2Eexample%2Ecom%2Fcb
&scope=openid
&openid=HeADeR.pAyl0rd.cRypT0 HTTP/1.1
Host: server.example.com
```

TOC

4.1.2. Authorization Response

When the `response_type` in the request was `code`, the Authorization Response MUST return the parameters defined in section 4.1.2 of **OAuth 2.0** [RFC5849]. Note that if the `response_type` in the request was `token`, the **Access Token Response** defined later MUST be returned instead.

For example, the Authorization Server redirects the End-User's user-agent by sending the following HTTP response:

```
HTTP/1.1 302 Found
Location: https://client.example.com/cb?code=i1WsRn1uB1&state=1f8skd
```

TOC

4.1.3. Authorization Error Response

If the end-user denies the access request or if the request fails, the authorization server informs the client by returning parameters defined in section 4.1.2.1 of **OAuth 2.0** [RFC5849] .

TOC

4.1.3.1. Error Codes

In addition to the error codes defined in section 4.1.2.1 of **OAuth 2.0** [RFC5849], this specification defines the following additional error codes:

invalid_client
: The client identifier provided is invalid, the client failed to authenticate, the client did not include its credentials, provided multiple client credentials, or used unsupported credentials type.

unauthorized_client
: The authenticated client is not authorized to use the access grant type provided.

invalid_grant
: The provided access grant is invalid, expired, or revoked (e.g. invalid assertion, expired authorization token, bad end-user password credentials, or mismatching authorization code and redirection URI).

unsupported_grant_type
: The access grant included - its type or another attribute - is not supported by the authorization server.

invalid_scope
: The requested scope is invalid, unknown, malformed, or exceeds the previously granted scope.

invalid_request_response_type
: The requested response type is unsupported or is missing.

invalid_request_type
: The request type is unsupported.

invalid_request_openid_type
: The openid type in the the request is not supported.

invalid_request_redirect_uri
: The redirect_uri in the request is missing or malformed.

invalid_request_signature
: The request has an invalid signature.

invalid_request_realm
: The openid request realm is missing or malformed.

invalid_request_atype
: The request contains an unsupported response assertion type.

invalid_request_recipient
: The recipient of the request is invalid or does not match.

The error codes can be extended by the string prefixed by `x_`. If custome error code are used, `error_uri` MUST be provided.

TOC

4.1.4. OpenID Request Object

The OpenID Request object is used to provide OpenID request parameters that differ from the default ones. Implementing support for the OpenID Request object is OPTIONAL. Supporting it is necessary for implementations that need to request or provide sets of claims other than the default UserInfo claim set.

If present, the OpenID Request object is passed as the value of a "req=" OAuth 2.0 parameter and is represented as a JWT. Parameters that affect the information returned from the UserInfo Endpoint are in the "inf" member. Parameters that affect the information returned in the OpenID Token are in the "oit" member.

An example an OpenID request object is as follows:

```
{
 "inf":
   {
     "clm":
       {
         "name": null,
```

```
        "displayName": {"opt": true},
        "emails": null,
        "photos": {"opt": true},
      },
    "fmt": "sig"
  }
 "oit":
  {
    "clm":
      {
       "aat": null
      }
    "mxa": 86400,
    "eaa": 2
  }
}
```

The OpenID Request object is a **JWT** [jwt] that MAY contain a set of members defined by this specification and MAY contain other members that are not defined by this specification. The JWT MAY be signed or unsigned. When it is unsigned, it will be indicated by the JWT `"sig":"none"` convention in the JWT header.

The members defined by this specification are:

inf
: OPTIONAL. (UserInfo Endpoint request): Requests affecting the values to be returned from the UserInfo Endpoint. If not present, the UserInfo Endpoint behaves in the default manner.

oit
: OPTIONAL. (OpenID Token request): Requests affecting the values to be included in the OpenID Token. If not present, the default OpenID Token contents are used. If present, these parameters are used to request deltas to the default contents of the OpenID Token.

If signed, the OpenID Request object SHOULD contain the standard JWT "iss" and "aud" claims.

The structure of the "inf" (UserInfo Endpoint request) member is a JSON object that MAY contain the following members:

clm
: OPTIONAL. (Requested Claims): Set of requested claims from the UserInfo Endpoint. If not present, the default UserInfo claims held by the IdP are returned.

fmt
: OPTIONAL. (Format): The requested format for the UserInfo Endpoint information. If not present, the format is an unsigned JSON object.

The "clm" member is a JSON object with a member for each requested claim. The member names are the requested claim names. The member values may be either:

null
: This indicates that this claim is being requested in the default manner. In particular, this is a required claim. OR

A JSON Object
: This is used to provide additional information about the claim being requested.

All members of the "clm" object are OPTIONAL.

The members of the JSON object value following a claim name defined by this specification are:

opt
: If this is an optional claim, this member's value MUST be `true`, else, if present, its value MUST be `false`, which indicates that it is a required claim. If it is not present, it is a required claim.

Other members MAY be defined to provide additional information about the requested claim. If the "clm" member is present in the "info" object, the claims requested within it override the default claim set that would otherwise be returned from the UserInfo Endpoint.

The "fmt" member of the "inf" object is used to specify the requested format of the UserInfo Endpoint contents. Values defined are:

nor
: (normal) - in which case the contents are an unsigned JSON object

sig
: (signed) - in which case the contents are a signed JWT

enc
: (encrypted) - in which case the contents are an signed and encrypted JWT

All members of the "inf" object are OPTIONAL. Other members MAY be present and if so, SHOULD understood by both parties.

The structure and function of the "oit" (OpenID Token request) member of the OpenID Request object is similar to that of the "inf" member. It also contains an optional "clm" member, with the same structure as that for the "inf" member. If the "clm" member is present in the "oit" object, the claims requested within it modify the default claim set that would otherwise be returned in the OpenID Token. Unlike for the "inf" member, typically these claims will augment, rather than override the default set.

Following claim MAY be requested in the OpenID Token by specifying it in the "clm" member:

aat
: OPTIONAL. (authenticated at): Requests that the "aat" claim be present. The claim value is the number of seconds from 1970-01-01T0:0:0Z as measured in UTC until the date/time that the user authentication occurred. (The "aat" claim semantically corresponds to the openid.pape.auth_time response parameter.)

In addition to the "clm" member, this additional member is defined within the "oit" member of the OpenID Request object:

mxa
: OPTIONAL. (max authentication age): Specifies that the user must be actively authenticated if any present authentication is older than the specified number of seconds. (The "mxa" request parameter corresponds to the OpenID 2.0 openid.pape.max_auth_age request parameter.)

eaa
: OPTIONAL. (entity authentication assurance): Specifies the X.eaa/ISO29115 entity authentication assurance level that is requested by the client.

It is anticipated that additional "oit" parameters MAY be defined to request that additional properties hold for the authentication - for instance, that certain authentication policies be applied (in the same spirit of the OpenID 2.0 openid.pape.auth_policies values), or that the authentication conform to the policies defined by a specified trust framework. These parameters MAY be defined by extension specifications.

All members of the "oit" object are OPTIONAL. Other members MAY be present and if so, SHOULD understood by both parties.

All members of the OpenID Request object are OPTIONAL. Other members MAY be present and if so, SHOULD be understood by both parties.

TOC

4.2. Token Endpoint

Access Token Request / Response interacts with a Token Endpoint. Upon the successful request, it returns two tokens, Access Token and OpenID Token.

TOC

4.2.1. Access Token Request

The client obtains an access token by authenticating with the authorization server and presenting its access grant (in the form of an authorization code, resource owner credentials, an assertion, or a refresh token).

The request parameters of of the Access Token Request is defined in section 4.1.3 of **OAuth 2.0** [RFC5849] .

TOC

4.2.2. Access Token Response

After receiving and verifying a valid and authorized Access Token Request from the client, the Authorization Server returns a Positive Assertion that includes an Access Token. The parameters in the successful response is defined in Section 5.1 of **OAuth 2.0** [RFC5849] .

In addition, this specification defines the following parameter:

openid
: REQUIRED if it was requested in the request. An OpenID Token. It is a **JWS** [jws] signed claim described below.

Following is a non-normative example.

```
{
    "access_token": "SlAV32hkKG",
```

```
    "token_type": "jwt",
    "refresh_token": "8xLOxBtZp8",
    "user_id": "http://op.example.com/alice#1234",
    "domain": "op.example.com",
    "expires_in": 3600,
    "openid":"jwtheader.jwtpayload.jwtcrypto"
}
```

As in the **OAuth 2.0** [RFC5849], Clients SHOULD ignore unrecognized response parameters.

TOC

4.2.2.1. OpenID Token

The OpenID Token is a JWS signed claim that attests the following:

server_id
: REQUIRED. The unique identifier of the authorization server such that when paired with the user_id creates a globally unique and never reassigned identifier.

user_id
: REQUIRED. A locally unique and never reassigned identifier for the user, which is intended to be consumed by the Client. e.g. "24400320" or "AItOawmwtWwcT0k51BayewNvutrJUqsvl6qs7A4". It MUST NOT exceed 255 ASCII characters in length.

aud
: REQUIRED. The **JWT** [jwt]aud (audience) claim.

exp
: REQUIRED. The **JWT** [jwt] exp (expiration time) claim.

pape
: OPTIONAL. (TBD) If we want this token to be short, we probably want to define a shorter equivalent of PAPE.

TOC

4.2.3. Token Error Response

If the assertion request is invalid or unauthorized, the authorization server constructs the error response. The parameters of the Token Error Response is defined as in Section 5.2 of **OAuth 2.0** [RFC5849].

TOC

4.2.3.1. Error Codes

In addition to the error codes defined in Section 5.2 of **OAuth 2.0** [RFC5849], this specification defines the following error codes.

invalid_client_secret
: The client secret does not matched pre-shared secret code, is of a different type, or has an invalid signature.

invalid_secret_type
: The specified secret type is unsupported.

invalid_request_signature
: The request has an invalid signature.

invalid_request_code
: The authorization code is missing, malformed, or invalid.

invalid_request_openid_type
: The openid type in the the request is not supported.

The error codes can be extended by the string prefixed by `x_`. If custome error code are used, `error_uri` MUST be provided.

TOC

4.3. UserInfo Endpoint

UserInfo Request/Response interacts with UserInfo Endpoint.

TOC

4.3.1. UserInfo Request

Client MAY send request with following parameters to the UserInfo Endpoint to obtain further information about the user.

access_token
: REQUIRED. The access_token obtained above.

user_id
: REQUIRED. A locally unique and never reassigned identifier for the user. e.g. "24400320" or "AItOawmwtWwcT0k51BayewNvutrJUqsvl6qs7A4". It MUST NOT exceed 255 ASCII characters in length. It could be a pairwise private identifier of the enduser between the Client and the Server.

client_id
: REQUIRED. The client identifier recognized by the authorization server.

TOC

4.3.2. UserInfo Response

The response is a JSON object which contains some (or all) of the following reserved keys:

[ToDo: Replace with scim based definition -- Pam doing the table.]

user_id
: REQUIRED. A locally unique and never reassigned identifier for the user. e.g. "24400320" or "AItOawmwtWwcT0k51BayewNvutrJUqsvl6qs7A4". It MUST NOT exceed 255 ASCII characters in length. It MUST NOT be reassigned to another user.

server_id
: REQUIRED. The unique identifier of the authorization server such that when paired with the user_id creates a globally unique and never reassigned identifier.

client_id
: REQUIRED. The client identifier recognized by the authorization server.

asserted_user
: REQUIRED. One of "true" if the access was issued for this user or "false" if it is for a different user.

profile_urls
: OPTIONAL. An array of URLs that belong to the user across any number of domains.

display_name
: OPTIONAL. The display name of the user. e.g., "David Recordon".

given_name
: OPTIONAL. The first name of the user. e.g., "David".

family_name
: OPTIONAL. The family name of the user. e.g., "Recordon".

email
: OPTIONAL. The verified email address of the user. e.g., "recordond@gmail.com".

language
: OPTIONAL. End User's preferred language as specified by ISO639.

picture
: OPTIONAL. The URL of End User's Picture. e.g., "http://graph.facebook.com/davidrecordon/picture".

openid
: REQUIRED if OpenID variables were specified in the Authorization Request. It is a JSON Object that includes the claim responses.

TOC

4.3.3. UserInfo Error Response

The Authorization Server includes one of the following error codes with the error response:

invalid_request
: The request is missing a required parameter, includes an unsupported parameter or parameter value, repeats a parameter, includes multiple credentials, utilizes more than one mechanism for authenticating the client, or is otherwise malformed.

invalid_client
: The client identifier provided is invalid, the client failed to authenticate, the client did not include its credentials, provided multiple client credentials, or used unsupported credentials type.

unauthorized_client
: The authenticated client is not authorized to use the access grant type provided.

invalid_grant
: The provided access grant is invalid, expired, or revoked (e.g. invalid assertion, expired authorization token, bad end-user password credentials, or mismatching authorization code and redirection URI).

unsupported_grant_type
: The access grant included - its type or another attribute - is not supported by the authorization server.

invalid_scope
: The requested scope is invalid, unknown, malformed, or exceeds the previously granted scope.

invalid_access_token

invalid_access_token
: The access token is not valid for the requested resource, malformed, is in an incorrect format, outside the valid scope, or expired.

invalid_refresh_token
: The refresh token is not valid, malformed, is in an incorrect format, outside the valid scope, or expired.

invalid_request_signature
: The request has an invalid signature.

invalid_request_type
: The request type is unsupported.

invalid_request_atype
: The request contains an unsupported response assertion type.

invalid_request_recipient
: The recipient of the request is invalid or does not match.

TOC

4.4. Session Management Endpoints

To manage a session, the client sends a request to the session management endpoints at the authorization server. The session management endpoints at the authorization server are:

Session Refresh
: Refreshes an expired ID Token

Check Session
: Get a plain text JSON structure from a ID Token

End Session
: Ends a session

TOC

4.4.1. Session Refresh

To refresh a ID Token session that has expired, the client sends a request to the Refresh Session endpoint with an ID Token. A new ID Token is returned in JWS signed format.

Request Parameters:

openid
: REQUIRED. A previously issued ID Token from a session request

state
: REQUIRED. An opaque value used by the Client to maintain state between the request and callback. If provided, the Authorization Server MUST include this value when redirecting the user-agent back to the Client. Clients are strongly advised to use this variable to relate the request and response.

redirect_uri
: REQUIRED. An absolute URI to which the authorization server will redirect the user-agent to with the new ID Token.

Response:

The response is a new ID Token. In a typical HTTP binding, an HTTP 302 redirect to the specified redirect_uri in the request with a new ID Token.

openid
: A new ID Token

The following is a non-normative session refresh request:

```
Request:

GET /op/refresh_token?openid=eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiIsImtpZCI6
ImNsaWVudC5leGFtcGxlLmNvbSJ9.eyJpc3N1ZXIiOiJodHRwOlwvXC9zZXJ2ZXIuZXhhbXBs
ZS5jb20iLCJjbGllbnRfaWQiOiJjbGllbnQuZXhhbXBsZS5jb20iLCJhdWRpZW5jZSI6ImNsa
WVudC5leGFtcGxlLmNvbSIsImlkIjoidXNlcl8yMzQyMzQiLCJleHAiOjEzMDM4NTI4ODB9.a
JwagC6501Da-zK-X8Az9B-Y625aSEfxVuBpFEDjOxQ
&state=bar&redirect_uri=https%3A%2F%2Fclient%2Eexample%2Ecom%2Fidtoken_cb
Host: server.example.com

Response:

HTTP/1.1 302 OK
Location: http://client.example.com/idtoken_cb?openid=eyJ0eXAiOiJKV1QiLCJh
bGciOiJIUzI1NiIsImtpZCI6ImNsaWVudC5leGFtcGxlLmNvbSJ9.eyJpc3N1ZXIiOiJodHRwO
lwvXC9zZXJ2ZXIuZXhhbXBsZS5jb20iLCJjbGllbnRfaWQiOiJjbGllbnQuZXhhbXBsZS5jb20
iLCJhdWRpZW5jZSI6ImNsaWVudC5leGFtcGxlLmNvbSIsImlkIjoidXNlcl8yMzQyMzQiLCJle
HAiOjEzMDM4NTI4ODB9.aJwagC6501Da-zK-X8Az9B-Y625aSEfxVuBpFEDjOxQ&state=bar&
expires_in=3600
```

TOC

4.4.2. Check Session

For clients that are not capable of dealing with JWS signed ID Tokens, they can send the ID Token to the Check Session endpoint. It will validate the ID Token and return a plain text JSON structure of the ID Token.

Request Parameters:

openid
: REQUIRED. A previously issued ID Token from a session request

Response:

The response body is a plain text JSON structure of the base64url decoded payload of the signed ID Token. In a typical HTTP binding, the response is a HTTP 200 response code with the content-type header set to "application/json".

The following is a non-normative example of a check session request:

```
Request:
POST /op/check_openid?openid=eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiIsImtpZCI6
ImNsaWVudC5leGFtcGxlLmNvbSJ9.eyJpc3N1ZXIiOiJodHRwOlwvXC9zZXJ2ZXIuZXhhbXBs
ZS5jb20iLCJjbGllbnRfaWQiOiJjbGllbnQuZXhhbXBsZS5jb20iLCJhdWRpZW5jZSI6ImNsa
WVudC5leGFtcGxlLmNvbSIsImlkIjoidXNlcl8yMzQyMzQiLCJleHAiOjEzMDM4NTI4ODB9.a
JwagC6501Da-zK-X8Az9B-Y625aSEfxVuBpFEDjOxQ

Response:
HTTP/1.1 200 OK
Content-Type: application/json

{
  "iss":"http://server.example.com",
  "client_id","http://client.example.com",
  "audience", "http://client.example.com",
  "user_id":"user_328723",
  "exp":1303852880
}
```

TOC

4.4.3. End Session

To end the session, the client sends a ID Token to the End Session endpoint. Upon receiving the request, the authorization server performs the logout flow for the user and then redirects the user-agent to the specified redirect_uri.

Request Parameters:

openid
: REQUIRED. A previously issued ID Token from a session request

state
: REQUIRED. An opaque value used by the Client to maintain state between the request and callback. If provided, the Authorization Server MUST include this value when redirecting the user-agent back to the Client. Clients are strongly advised to use this variable to relate the request and response.

redirect_uri
: REQUIRED. An absolute URI to which the authorization server will redirect the user-agent to with the new ID Token.

Response:

The response is dependant on the particular binding. In HTTP binding, the response is a HTTP 302 redirect response to the redirect_uri specified in the request.

The following is a non-normative session refresh request:

```
Request:

GET /op/end_session?openid=eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiIsImtpZCI6
ImNsaWVudC5leGFtcGxlLmNvbSJ9.eyJpc3N1ZXIiOiJodHRwOlwvXC9zZXJ2ZXIuZXhhbX
BsZS5jb20iLCJjbGllbnRfaWQiOiJjbGllbnQuZXhhbXBsZS5jb20iLCJhdWRpZW5jZSI6I
mNsaWVudC5leGFtcGxlLmNvbSIsImlkIjoidXNlcl8yMzQyMzQiLCJleHAiOjEzMDM4NTI4
```

```
ODB9.aJwagC6501Da-zK-X8Az9B-Y625aSEfxVuBpFEDjOxQ
&state=bar
&redirect_uri=https%3A%2F%2Fclient%2Eexample%2Ecom%2Fendtoken_cb
Host: server.example.com

...
   Authorizion server performs logout flow
...

Response:

HTTP/1.1 302 OK
Location: http://client.example.com/endtoken_cb?state=bar
```

TOC

5. serializations

Each message can be serialized either in query parameter serialization or JSON serialization unless it was explicitly stated in the previous sections.

TOC

5.1. Query String serialization

In order to serialize the parameters into Query String Serialization, the client constructs the string by adding the following parameters to the end-user authorization endpoint URI query component using the `application/x-www-form-urlencoded` format as defined by **HTML 4.01 Specification** [html401]:

Following is a non-normative example of such Serialization.

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

TOC

5.2. JSON Serialization

The parameters are serialized into a JSON structure by adding each parameter at the highest structure level. Parameter names and string values are included as JSON strings. Numerical values are included as JSON numbers. Each parameter may have JSON Structure as its value.

Following is a non-normative example of such Serialization.

```
{
  "access_token":"SlAV32hkKG",
  "expires_in":3600,
  "refresh_token":"8xLOxBtZp8",
  "openid": {
    "type": "http://openid.net/specs/ab/1.0#id_res",
    "mode": "id_res",
    "op_endpoint": "https://op.example.com/op_endpoint",
    "client_id": "http://rp.example.com/",
    "server_id": "http://op.example.com/",
    "claimed_id": "https://example.com/alice#1234",
    "identity": "alice",
    "issued_at": 1274889460
  }
}
```

TOC

6. Signatures

Depending on the transport through wich the messages are transported, the integrity of the message may not be guaranteed, nor the originator of the message is not authenticated. To mitigate these risks, OpenID Connect supports **JSON Web Signatures(JWS)** [jws].

Following is the parameters for JWT.

typ
: OPTIONAL. One of `"JWT"`, `"openid2json+sig"`.

alg
: REQUIRED. One of the algorithm specified in Table 4 of **JWT** [jwt]

Compact Serialization SHOULD BE used when passing it through query parameters, while JSON serialization SHOULD BE used when returning it in HTTP Body.

Following is a non-normative example of such signature in Compact serialization, where HS256 algorithm was used (with line breaks for display purposes only):

```
eyJ0eXAiOiJKV1QiLA0KICJhbGciOiJIUzI1NiJ9
.
eyJpc3MiOiJqb2UiLA0KICJleHAiOjEzMDA4MTkzODAsDQogImh0dHA6Ly9leGFtcGxlLmNvbS9pc19yb290Ijp0cnVlfQ
.
dBjftJeZ4CVP-mB92K27uhbUJU1p1r_wW1gFWFOEjXk
```

7. Encryption

TOC

To achieve message confidentiality and audience restriction, OpenID Connect uses **JSON Web Encryption (JWE)** [jwe].

8. Requests and Responses

TOC

Requests and Responses can either be plain, signed or encrypted. Signature should be applied to the entire request or response. Signed request and responses in the query are sent in the parameter "signed" together with other parameters. If the request and responses are in the JSON Serialization, the JWS signed version MUST use the JSON serialization.

If the request and responses are to be encrypted with **JSON Web Encryption (JWE)** [jwe], non-encrypted payload MUST NOT be sent. The parameter name for the encrypted payload MUST be 'jwe'.

9. Verification

TOC

9.1. Authorization Request Verification

TOC

If the request was signed, the Server MUST verify the signature as in JWT.

9.2. Authorization Response Verification

TOC

To verify the validity of the Authorization Response, the client MUST to the following:

1. If the response was signed, the Client SHOULD verify the signature as in JWT as the first step.
2. Check that OP that it connected was really the intended OP through TLS/SSL server certificate check if the endpoint is TLS/SSL endpoint.
3. Check that current time is within the validity period.

If the client does not verify the signature, it MUST make a User Info API request.

9.3. UserInfo Request Verification

TOC

If the request was signed, the Server MUST verify the signature as in **JWT** [jwt].

9.4. UserInfo Response Verification

TOC

To verify the validity of the UserInfo Response, the client MUST to the following:

1. If the response was signed, the Client SHOULD verify the signature as in JWT as the first step.
2. Check that OP that it connected was really the intended OP through TLS/SSL server certificate check if the endpoint is TLS/SSL endpoint.
3. Check if the current time is within the validity period.

10. Extensions

TOC

OpenID Connect supports the extension parameters in OpenID Request Object, OpenID Token, and UserInfo Response.

11. Security Considerations

TOC

Followings are the list of attack vectors and remedies that were considered for this specification.

For details of the attack vector, see **[SP800-63]**.

11.1. Assertion manufacture/modification

TOC

To mitigate this attack, there are two ways to mitigate it.

1. The assertion may be digitally signed by the OP. The Relying Party SHOULD check the digital signature to verify that it was issued by a legitimate OP.
2. The assertion may be sent over a protected channel such as TLS/SSL. In order to protect the integrity of assertions from malicious attack, the OP MUST be authenticated. In this specification, the assertion is always sent over TLS/SSL protected channel.

11.2. Assertion disclosure

TOC

The Assertion disclosure can be mitigated in the following two ways.

1. Assertion is sent over TLS/SSL protected channel, where RP is authenticated by "client_id" and "client_secret".
2. Signed Assertion is encrypted by the RP's public key.

11.3. Assertion repudiation

TOC

To mitigate this threat, the assertion may be digitally signed by the OP using a key that supports non-repudiation. The RP SHOULD check the digital signature to verify that it was issued by a legitimate OP.

11.4. Assertion redirect

TOC

To mitigate this threat, the assertion includes the identity of the RP for whom it was generated as "client_id". The RP verifies that incoming assertions include its identity as the recipient of the assertion.

11.5. Assertion reuse

TOC

The assertion includes a timestamp and a short lifetime of validity. The Relying Party checks the timestamp and lifetime values to ensure that the assertion is currently valid.

11.6. Secondary authenticator manufacture

TOC

Due to the large entropy requirement of the Artifact ("code") and short life nature of its validity, the success probability of this attack is extremely low.

11.7. Secondary authenticator capture

TOC

Secondary authenticator (="code") is transmitted only through HTTPS, thus it is protected between the OP and the User-Agent, and User-Agent and the RP.

Only the place it can be captured is the User-Agent where the TLS session is terminated, and is possible if the User-Agent is infested by malwares. However, it renders no usefulness as long as the profile in use either RP authentication or assertion encryption.

11.8. Assertion substitution

TOC

Responses to assertion requests is bound to the corresponding requests by message order in HTTP, as both assertions and requests are protected by TLS that can detect and disallow malicious reordering of packets.

11.9. Authentication Request Disclosure

TOC

If the authentication request is POSTed directly through a protected channel, it is not possible to disclose the authentication request.

If the Request File is encrypted by the OP's public key, the authentication request will not be disclosed unless OP's private key gets compromised or the encryption algorithm becomes vulnerable.

11.10. Timing Attack

TOC

Timing attack can be used to reduce the effctive key length of the signature if the time required to return the response in case of signature error and correct signature exists. Care should be taken in the implementation to avoid this attack.

11.11. Authentication Process Threats

TOC

In the category of Authentication Process Threats, following threats exists.

- Online guessing
- Phishing
- Pharming
- Eavesdropping
- Replay
- Session hijack
- Man-in-the-middle

Authentication process per se as described in NIST SP800-63-rev1 is out of scope for this protocol, but care SHOULD be taken to achieve appropriate protection.

12. IANA Considerations

TOC

12.1. OAuth Parameters Registry

TOC

12.1.1. Scope Parameters

TOC

The following is the parameter registration request for the "scope" parameter as defined in this specification:

- Parameter name: openid
- Parameter usage location: The End-User Authorization Endpoint request, the End-User Authorization Endpoint response, the Token Endpoint request, the Token Endpoint response, and the `WWW-Authenticate` header field.

- Change controller: IETF
- Specification document(s): [[this document]]
- Related information: None

12.1.2. Authorization Request Parameters

TOC

The follwoing is the parameter registration request for the Authorization Reqeust in this specification:

- Parameter name: openid
- Parameter usage location: Authorization Request
- Change controller: IETF
- Specification document(s): [[this document]]
- Related information: None

12.1.3. Access Token Response Parameters

TOC

The following is the parameter registration request for the Access Token Response in this specification:

- Parameter name: openid
- Parameter usage location: Access Token Response
- Change controller: IETF
- Specification document(s): [[this document]]
- Related information: None

---

13. Open Issues and Things To Be Done (TBD)

TOC

[[To be removed from the final specification.]]

Following items remains to be done in this draft.

1. Clean Up and add references.
2. Update JWT/JWS/JWE related things with the most current version of them.
3. Finish the security consideration section.
4. Properly capitalize the Defined Words.
5. Better to split the Authentication and Authorization server? (Model-wise, yes, but it gets complicated. Current model is implicitly assuming that the Authentication and Authorization server are operated by the same entity and the protocol between them are proprietary.)

Appendix A. Acknowledgements

TOC

As a successor version of **OpenID Authentication 2.0** [OpenID.authentication-2.0], this specification heavily relies on **OpenID Authentication 2.0** [OpenID.authentication-2.0]. Please refer to Appendix C of **OpenID Authentication 2.0** [OpenID.authentication-2.0] for the full list of the contributors for **OpenID Authentication 2.0** [OpenID.authentication-2.0].

This specification is largely compliant with OAuth 2.0 draft 15. As the draft is not yet referenceable, relevant text has been incorporated into this draft. Please refer to the OAuth 2.0 specification for the list of contributors.

In addition, the OpenID Community would like to thank the following people for the work they've done in the drafting and editing of this specification.

Anthony Nadalin (tonynad@microsoft.com), Microsoft.

Breno de Medeiros (breno@gmail.com), Google.

Chuck Mortimore (cmortimore@salesforce.com), Salesforce.com.

David Recordon (dr@fb.com)<author>, Facebook.

George Fletcher (george.fletcher@corp.aol.com), AOL.

Hideki Nara (hideki.nara@gmail.com), Takt Communications.

John Bradley (jbradely@mac.com) <author>, Protiviti Government Service.

Mike Jones (Michael.Jones@microsoft.com), Microsoft.

Nat Sakimura (n-sakimura@nri.co.jp) <author/editor>, Nomura Research Institute, Ltd.

Paul Tarjan (pt@fb.com), Facebook.

Ryo Itou (ritou@yahoo-corp.jp), Yahoo! Japan.

Appendix B. Document History

TOC

-01

First Draft that incorporates the core of both openidonnect.com proposal and OpenID Artifact Binding RC3 and abstracted.

-02

Catch up to OAuth 2.0 d15. Replaced JSS and JSE to JWS and JWE. Section grouping and reorganizations. Added more contributors.

-03

Combined with Session Management. Moved "openid" back to Token Endpoint. Renaming the sections according to the Endpoint names. Rewrote intro to the Messages section to be more approacheable.

-04

To keep the ID Token small so that it fits cookie more easily, moved OPTIONAL parameters to UserInfo endpoint response.

-05

Reference OAuth 2.0 now since it will be stable.

14. Normative References

TOC

[OpenID.AB] Sakimura, N., Ed., Bradley, J., de Madeiros, B., Ito, R., and M. Jones, "**OpenID Connect Artifact Binding 1.0**," January 2011.

[OpenID.AC] Mortimore, C., Ed., Sakimura, N., Bradley, J., de Madeiros, B., Ito, R., and M. Jones, "**OpenID Connect Authorization Code Binding 1.0**," January 2011.

[OpenID.authentication-2.0] specs@openid.net, "OpenID Authentication 2.0," 2007 (**TXT**, **HTML**).

[RFC1421] **Linn, J.**, "**Privacy Enhancement for Internet Electronic Mail: Part I: Message Encryption and Authentication Procedures**," RFC 1421, February 1993 (**TXT**).

[RFC1422] **Kent, S.**, "**Privacy Enhancement for Internet Electronic Mail: Part II: Certificate-Based Key Management**," RFC 1422, February 1993 (**TXT**).

[RFC1423] **Balenson, D.**, "**Privacy Enhancement for Internet Electronic Mail: Part III: Algorithms, Modes, and Identifiers**," RFC 1423, February 1993 (**TXT**).

[RFC1424] **Kaliski, B.**, "**Privacy Enhancement for Internet Electronic Mail: Part IV: Key Certification and Related Services**," RFC 1424, February 1993 (**TXT**).

[RFC1750] **Eastlake, D.**, **Crocker, S.**, and **J. Schiller**, "**Randomness Recommendations for Security**," RFC 1750, December 1994 (**TXT**).

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

[RFC2616] **Fielding, R.**, **Gettys, J.**, **Mogul, J.**, **Frystyk, H.**, **Masinter, L.**, **Leach, P.**, and **T. Berners-Lee**, "**Hypertext Transfer Protocol -- HTTP/1.1**," RFC 2616, June 1999 (**TXT**, **PS**, **PDF**, **HTML**, **XML**).

[RFC2617] **Franks, J.**, **Hallam-Baker, P.**, **Hostetler, J.**, **Lawrence, S.**, **Leach, P.**, Luotonen, A., and **L. Stewart**, "**HTTP Authentication: Basic and Digest Access Authentication**," RFC 2617, June 1999 (**TXT**, **HTML**, **XML**).

[RFC3339] **Klyne, G., Ed.** and **C. Newman**, "**Date and Time on the Internet: Timestamps**," RFC 3339, July 2002 (**TXT**, **HTML**, **XML**).

[RFC3548] Josefsson, S., "**The Base16, Base32, and Base64 Data Encodings**," RFC 3548, July 2003 (**TXT**).

[RFC3629] Yergeau, F., "**UTF-8, a transformation format of ISO 10646**," STD 63, RFC 3629, November 2003 (**TXT**).

[RFC3986] **Berners-Lee, T.**, **Fielding, R.**, and **L. Masinter**, "**Uniform Resource Identifier (URI): Generic Syntax**," STD 66, RFC 3986, January 2005 (**TXT**, **HTML**, **XML**).

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

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

[RFC5849] Hammer-Lahav, E., "**The OAuth 1.0 Protocol**," RFC 5849, April 2010 (**TXT**).

[SP800-63] National Institute of Standards and Technology, "**NIST SP800-63rev.1: Electronic Authentication Guideline**," NIST SP800-63.

Defines LoA

[html401] Ragget, D., "**HTML 4.01 Specification**," December 1999.

[jwe] Jones, M., Belfanz, D., Bradeley, J., Goland, Y., Panzer, J., Sakimura, N., and P. Tarjan, "**JSON Web Encryption**," March 2011.

[jws] Jones, M., Belfanz, D., Bradeley, J., Goland, Y., Panzer, J., Sakimura, N., and P. Tarjan, "**JSON Web Signatures**," March 2011.

[jwt] Jones, M., Belfanz, D., Bradeley, J., Goland, Y., Panzer, J., Sakimura, N., and P. Tarjan, "**JSON Web Token**," January 2011.

Authors' Addresses

TOC

Nat Sakimura (editor)
Nomura Research Institute, Ltd.
Email: n-sakimura@nri.co.jp

David Recordon
Facebook Inc.
Email: dr@fb.com

John Bradley
Protiviti Government Services
Email: jbradley@mac.com

Breno de Madeiros
Google Inc.
Email: breno@google.com

Mike Jones
Microsoft Corporation
Email: Michael.Jones@microsoft.com

Edmund Jay (editor)
MGI1
Email: ejay@mgi1.com