|  |
| --- |
| [TOC](#toc) |

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  | | --- | --- | | Draft | N. Sakimura | |  | NRI | |  | J. Bradley | |  | Ping Identity | |  | M. Jones | |  | Microsoft | |  | B. de Medeiros | |  | Google | |  | C. Mortimore | |  | Salesforce | |  | E. Jay | |  | Illumila | |  | May 30, 2013 | |

# OpenID Connect Messages 1.0 - draft 19

### Abstract

OpenID Connect 1.0 is a simple identity layer on top of the OAuth 2.0 protocol. It allows Clients to verify the identity of the End-User based on the authentication performed by an Authorization Server, as well as to obtain basic profile information about the End-User in an interoperable and REST-like manner.

This specification only defines the endpoints and the associated message formats. The actual use MUST be based on one of the companion protocol bindings specifications such as OpenID Connect Standard 1.0.

### Table of Contents

[**1.**](#Introduction)  Introduction  
    [**1.1.**](#rnc)  Requirements Notation and Conventions  
    [**1.2.**](#terminology)  Terminology  
    [**1.3.**](#Overview)  Overview  
[**2.**](#Messages)  Messages  
    [**2.1.**](#AuthorizationEndpoint)  Authorization Endpoint  
        [**2.1.1.**](#AuthorizationRequest)  Authorization Request  
            [**2.1.1.1.**](#RequestParameters)  Request Parameters  
        [**2.1.2.**](#AuthzResponse)  Authorization Response  
            [**2.1.2.1.**](#id_token)  ID Token  
        [**2.1.3.**](#AuthError)  Authorization Error Response  
    [**2.2.**](#token_ep)  Token Endpoint  
        [**2.2.1.**](#client_authentication)  Client Authentication  
        [**2.2.2.**](#access_token_request)  Access Token Request  
        [**2.2.3.**](#access_token_response)  Access Token Response  
        [**2.2.4.**](#TokenErrorResponse)  Access Token Error Response  
    [**2.3.**](#userinfo)  UserInfo Endpoint  
        [**2.3.1.**](#UserInfoRequest)  UserInfo Request  
        [**2.3.2.**](#UserInfoResponse)  UserInfo Response  
        [**2.3.3.**](#UserInfoError)  UserInfo Error Response  
    [**2.4.**](#scopes)  Scope Values  
    [**2.5.**](#StandardClaims)  Standard Claims  
        [**2.5.1.**](#address_claim)  Address Claim  
        [**2.5.2.**](#ClaimsLanguagesAndScripts)  Claims Languages and Scripts  
        [**2.5.3.**](#claim.stability)  Claim Stability and Uniqueness  
        [**2.5.4.**](#AdditionalClaims)  Additional Claims  
    [**2.6.**](#ClaimsRequest)  Claims Request  
        [**2.6.1.**](#IndividualClaimsRequests)  Individual Claims Requests  
        [**2.6.2.**](#IndividualClaimsLanguages)  Languages and Scripts for Individual Claims  
    [**2.7.**](#ClaimTypes)  Claim Types  
        [**2.7.1.**](#NormalClaims)  Normal Claims  
        [**2.7.2.**](#AggregatedDistributedClaims)  Aggregated and Distributed Claims  
            [**2.7.2.1.**](#AggregatedExample)  Example of Aggregated Claims  
            [**2.7.2.2.**](#DistributedExample)  Example of Distributed Claims  
    [**2.8.**](#idtype)  Subject Identifier Types  
        [**2.8.1.**](#idtype.pairwise.alg)  Pairwise Identifier Algorithm  
    [**2.9.**](#RequestObject)  Request Object  
    [**2.10.**](#RequestUriParameter)  Using the "request\_uri" Parameter  
        [**2.10.1.**](#RequestUriRationale)  "request\_uri" Rationale  
[**3.**](#sigenc)  Signatures and Encryption  
    [**3.1.**](#sigenc.alg)  Supported Algorithms  
    [**3.2.**](#sigenc.key)  Keys  
    [**3.3.**](#sigs)  Signing  
        [**3.3.1.**](#rotate.sig.keys)  Rotation of Asymmetric Signing Keys  
    [**3.4.**](#enc)  Encryption  
        [**3.4.1.**](#rotate.enc.keys)  Rotation of Asymmetric Encryption Keys  
[**4.**](#Validation)  Validation  
    [**4.1.**](#AuthorizationRequestValidation)  Authorization Request Validation  
        [**4.1.1.**](#EncryptedRequestObject)  Encrypted Request Object  
        [**4.1.2.**](#signed.req.obj.var)  Signed Request Object  
        [**4.1.3.**](#req.obj.veri)  Parameter Validation  
    [**4.2.**](#id.token.validation)  ID Token Validation  
    [**4.3.**](#UserInfoResponseValidation)  UserInfo Response Validation  
    [**4.4.**](#access.token.validation)  Access Token Validation  
    [**4.5.**](#code.validation)  Code Validation  
[**5.**](#OfflineAccess)  Offline Access  
[**6.**](#self_issued)  Self-Issued OpenID Provider  
    [**6.1.**](#self_issued.discovery)  Self-Issued OpenID Provider Discovery  
    [**6.2.**](#self_issued.registration)  Self-Issued OpenID Provider Registration  
        [**6.2.1.**](#SelfIssuedRegistrationRequest)  Providing Additional Registration Information  
    [**6.3.**](#self_issued.request)  Self-Issued OpenID Provider Request  
    [**6.4.**](#self_issued.response)  Self-Issued OpenID Provider Response  
    [**6.5.**](#self_issued.validation)  Self-Issued ID Token Validation  
[**7.**](#stringops)  String Operations  
[**8.**](#ImplementationConsiderations)  Implementation Considerations  
    [**8.1.**](#ServerMTI)  Mandatory to Implement Features for All OpenID Providers  
    [**8.2.**](#DynamicMTI)  Mandatory to Implement Features for Dynamic OpenID Providers  
    [**8.3.**](#related)  Related Specifications  
[**9.**](#security_considerations)  Security Considerations  
    [**9.1.**](#request_disclosure)  Request Disclosure  
    [**9.2.**](#server_masquerading)  Server Masquerading  
    [**9.3.**](#token_manufacture)  Token Manufacture/Modification  
    [**9.4.**](#response_disclosure)  Server Response Disclosure  
    [**9.5.**](#server_response_repudiation)  Server Response Repudiation  
    [**9.6.**](#request_repudation)  Request Repudiation  
    [**9.7.**](#access_token_redirect)  Access Token Redirect  
    [**9.8.**](#token_reuse)  Token Reuse  
    [**9.9.**](#auth_code_capture)  Eavesdropping or Leaking Authorization Codes (Secondary Authenticator Capture)  
    [**9.10.**](#token_substitution)  Token Substitution  
    [**9.11.**](#TimingAttack)  Timing Attack  
    [**9.12.**](#OtherCryptoAttacks)  Other Crypto Related Attacks  
    [**9.13.**](#signing_order)  Signing and Encryption Order  
    [**9.14.**](#issuer_identifier)  Issuer Identifier  
    [**9.15.**](#TLS_requirements)  TLS Requirements  
    [**9.16.**](#token_lifetime)  Lifetimes of Access Tokens and Refresh Tokens  
    [**9.17.**](#SymmetricKeyEntropy)  Symmetric Key Entropy  
    [**9.18.**](#NeedForSignedRequests)  Need for Signed Requests  
    [**9.19.**](#NeedForEncryptedRequests)  Need for Encrypted Requests  
[**10.**](#privacy_considerations)  Privacy Considerations  
[**11.**](#IANA)  IANA Considerations  
    [**11.1.**](#ClaimsRegistry)  JSON Web Token Claims Registry  
        [**11.1.1.**](#ClaimsContents)  Registry Contents  
    [**11.2.**](#OAuthParametersRegistry)  OAuth Parameters Registry  
        [**11.2.1.**](#ParametersContents)  Registry Contents  
    [**11.3.**](#OAuthErrorRegistry)  OAuth Extensions Error Registry  
        [**11.3.1.**](#ErrorContents)  Registry Contents  
[**12.**](#rfc.references1)  References  
    [**12.1.**](#rfc.references1)  Normative References  
    [**12.2.**](#rfc.references2)  Informative References  
[**Appendix A.**](#Acknowledgements)  Acknowledgements  
[**Appendix B.**](#Notices)  Notices  
[**Appendix C.**](#History)  Document History  
[**§**](#rfc.authors)  Authors' Addresses

|  |
| --- |
| [TOC](#toc) |

### 1.  Introduction

|  |
| --- |
| [TOC](#toc) |

### 1.1.  Requirements Notation and Conventions

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 (**Bradner, S., “Key words for use in RFCs to Indicate Requirement Levels,” March 1997.**)](#RFC2119) [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.

All uses of [JSON Web Signature (JWS) (**Jones, M., Bradley, J., and N. Sakimura, “JSON Web Signature (JWS),” May 2013.**)](#JWS) [JWS] and [JSON Web Encryption (JWE) (**Jones, M., Rescorla, E., and J. Hildebrand, “JSON Web Encryption (JWE),” May 2013.**)](#JWE) [JWE] data structures in this specification utilize the JWS Compact Serialization or the JWE Compact Serialization; the JWS JSON Serialization and the JWE JSON Serialization are not used.

|  |
| --- |
| [TOC](#toc) |

### 1.2.  Terminology

This specification uses the terms "Access Token", "Refresh Token", "Authorization Code", "Authorization Grant", "Authorization Server", "Authorization Endpoint", "Client", "Client Identifier", "Client Secret", "Protected Resource", "Resource Owner", "Resource Server", and "Token Endpoint" defined by [OAuth 2.0 (**Hardt, D., “The OAuth 2.0 Authorization Framework,” October 2012.**)](#RFC6749) [RFC6749], the terms "Claim Names" and "Claim Values" defined by [JSON Web Token (JWT) (**Jones, M., Bradley, J., and N. Sakimura, “JSON Web Token (JWT),” May 2013.**)](#JWT) [JWT].

This specification also defines the following terms:

Authentication

Process of verifying the accuracy of the identity. Typically it involves the verification of the current or past possession of particular credentials including what the entity knows, possesses, has physical feature of, behaves, and combination of these utilizing heuristics..

Authentication Context

Information that the Relying Party can require before it makes an entitlement decision with respect to an authentication response. Such context can include, but is not limited to, the actual authentication method used or level of assurance such as [**ISO/IEC 29115 (International Organization for Standardization, “ISO/IEC FDIS 29115 -- Information technology - Security techniques - Entity authentication assurance framework,” December 2012.)**](#ISO29115) [ISO29115] entity authentication assurance level.

Authentication Context Class

A set of authentication methods or procedures that are considered to be equivalent to each other in a particular context.

Authentication Context Class Reference

Identifier for an Authentication Context Class.

Claim

Piece of information about an Entity that a Claims Provider asserts about that Entity.

Claim Type

Syntax used for representing a Claim Value. This specification defines Normal, Aggregated, and Distributed Claim Types.

Claims Provider

Server that can return Claims about an Entity.

End-User

Human Resource Owner.

Entity

Something that has a separate and distinct existence and that can be identified in context. An End-User is one example of an Entity.

Essential Claim

Claim specified by the Client as being necessary to ensure a smooth authorization experience for the specific task requested by the End-User.

Identity

Set of attributes related to an entity

Identifier

one or more attributes that uniquely characterize an entity in a specific context

ID Token

Token that contains Claims about the authentication event. It MAY contain other Claims.

Issuer

Entity that issues a set of Claims.

Issuer Identifier

Verifiable identifier for an Issuer. An Issuer Identifier is a URL using the https scheme that contains scheme, host, and OPTIONALLY, port number and path components. (No query or fragment components MAY be present.)

Message

Request or a response between an OpenID Relying Party and an OpenID Provider.

OpenID Provider (OP)

OAuth 2.0 Authorization Server that is capable of returning Claims to a Relying Party about the authentication event and the End-User in an ID Token and/or a UserInfo Endpoint response.

OP Endpoints

Authorization Endpoint, Token Endpoint, and UserInfo Endpoint.

Request Object

JWT that contains a set of request parameters as its Claims.

Personally Identifiable Information (PII)

Any information that (a) can be used to identify the natural person to whom such information relates, or (b) is or might be directly or indirectly linked to a natural person to whom such information relates.

Pairwise Pseudonymous Identifier (PPID)

Identifier that identifies the Entity to a Relying Party. An Entity's PPID at one Relying Party cannot be correlated with the Entity's PPID at another Relying Party.

Relying Party (RP)

Application requiring Claims from an OpenID Provider. It is an extended OAuth 2.0 Client.

Self-Issued OpenID Provider

Personal OpenID Provider that issues self-signed ID Tokens.

UserInfo Endpoint

Protected Resource that, when presented with an Access Token by the Client, returns Claims about the End-User represented by that Access Token.

**Validation**

Process intended to establish the soundness or correctness of a construct.

**Verification**

Process intended to test or prove the truth or accuracy of a fact or value.

Voluntary Claim

Claim specified by the Client as being useful but not Essential for the specific task requested by the End-User.

For more background on some of the terminology used, see [ITU-T X.1252 (**International Telecommunication Union, “ITU-T Recommendation X.1252 -- Cyberspace security -- Identity management -- Baseline identity management terms and definitions,” November 2010.**)](#X.1252) [X.1252], ISO/IEC 29115 Entity Authentication Assurance [ISO29115], and [RFC4949].

|  |
| --- |
| [TOC](#toc) |

### 1.3.  Overview

The OpenID Connect protocol, in abstract, follows the following steps.

1. The RP (Client) sends a request to the OP's (Authorization Server's) End-User Authorization Endpoint.
2. The OP authenticates the End-User and obtains appropriate authorization.
3. The OP responds with an Access Token, an Id Token, and a few other variables.
4. The RP sends a request with the Access Token to the UserInfo Endpoint, per [**Section 2.3 (UserInfo Endpoint)**](#userinfo).
5. The UserInfo Endpoint returns the additional End-User information supported by the Resource Server.

This specification only defines the abstract message flow and message formats. The actual use MUST be based on one of the companion protocol bindings specifications such as [OpenID Connect Standard 1.0 (**Sakimura, N., Bradley, J., Jones, M., de Medeiros, B., Mortimore, C., and E. Jay, “OpenID Connect Standard 1.0,” May 2013.**)](#OpenID.Standard) [OpenID.Standard], [OpenID Connect Basic Client Profile 1.0 (**Sakimura, N., Bradley, J., Jones, M., de Medeiros, B., and C. Mortimore, “OpenID Connect Basic Client Profile 1.0,” May 2013.**)](#OpenID.Basic) [OpenID.Basic], or [OpenID Connect Implicit Client Profile 1.0 (**Sakimura, N., Bradley, J., Jones, M., de Medeiros, B., Mortimore, C., and E. Jay, “OpenID Connect Implicit Client Profile 1.0,” May 2013.**)](#OpenID.Implicit) [OpenID.Implicit].

|  |
| --- |
| [TOC](#toc) |

### 2.  Messages

In OpenID Connect, the RP interacts with these endpoints at the OP:

1. Authorization Endpoint: The RP sends a request to the OP at the Authorization Endpoint. The OP then authenticates the End-User to determine whether he is eligible to be authenticated. Then, after potential authorization actions by the End-User, the Authorization Server returns an Authorization Response, which can include an Authorization Code value. For some Clients, the Implicit Grant will be used to obtain an ID Token and Access Token without using an Authorization Code; the response\_type parameter value id\_token token does this. For some Clients, the Implicit Grant will be used to obtain only an ID Token; the response\_type parameter value id\_token does this.
2. Token Endpoint: Clients using an Authorization Code send an Access Token Request containing the Authorization Code to the Token Endpoint to obtain an Access Token Response that includes an Access Token and an ID Token.
3. UserInfo Endpoint: Clients using an Access Token send the Access Token to the UserInfo Endpoint to obtain Claims about the End-User.

|  |
| --- |
| [TOC](#toc) |

### 2.1.  Authorization Endpoint

The RP sends an Authorization Request to the Authorization Endpoint of the OP to obtain an Authorization Response, which MAY contain an ID Token and Access Token, depending on the response\_type value used.

|  |
| --- |
| [TOC](#toc) |

### 2.1.1.  Authorization Request

An Authorization Request is a message sent from an RP to the OP's Authorization Endpoint. It is an extended [OAuth 2.0 (**Hardt, D., “The OAuth 2.0 Authorization Framework,” October 2012.**)](#RFC6749) [RFC6749] Authorization Request. Section 4.1.1 and 4.2.1 of [OAuth 2.0 (**Hardt, D., “The OAuth 2.0 Authorization Framework,” October 2012.**)](#RFC6749) [RFC6749] define the OAuth 2.0 Authorization Request parameters.

|  |
| --- |
| [TOC](#toc) |

### 2.1.1.1.  Request Parameters

OpenID Connect uses the following OAuth 2.0 request parameters:

response\_type

REQUIRED. Value that controls the parameters returned in the response from the Authorization Endpoint.

The OAuth 2.0 specification defines two response types:

code

When supplied as the value for the response\_type parameter, a successful response MUST include an Authorization Code as defined in the OAuth 2.0 specification. Both successful and error responses MUST be added as parameters to the query component of the response. All tokens are returned from the Token Endpoint. When used by OpenID Connect, an ID Token is also returned from the Token Endpoint. OpenID Providers that are not Self-Issued OPs MUST support this response\_type.

token

When supplied as the value for the response\_type parameter, a successful response MUST include an Access Token as defined in the OAuth 2.0 specification. Both successful and error responses MUST be fragment-encoded. No ID Token is provided to the Client; therefore, this response\_type is not used by OpenID Connect.

OpenID Connect supports these [**additional response types (de Medeiros, B., Scurtescu, M., and P. Tarjan, “OAuth 2.0 Multiple Response Type Encoding Practices,” June 2013.)**](#OAuth.Responses) [OAuth.Responses], which have been registered with IANA:

id\_token

When supplied as the value for the response\_type parameter, a successful response MUST include an ID Token. Both successful and error responses SHOULD be fragment-encoded. OpenID Providers MUST support this response\_type.

id\_token token

When supplied as the value for the response\_type parameter, a successful response MUST include both an Access Token and an ID Token. Both successful and error responses SHOULD be fragment-encoded. OpenID Providers MUST support this response\_type.

code token

When supplied as the value for the response\_type parameter, a successful response MUST include both an Access Token and an Authorization Code as defined in the OAuth 2.0 specification. When used by OpenID Connect, an ID Token is also returned from the Token Endpoint. Both successful and error responses SHOULD be fragment-encoded.

code id\_token

When supplied as the value for the response\_type parameter, a successful response MUST include both an Authorization Code and an ID Token. Both successful and error responses SHOULD be fragment-encoded.

code id\_token token

When supplied as the value for the response\_type parameter, a successful response MUST include an Authorization Code, an ID Token, and an Access Token. Both successful and error responses SHOULD be fragment-encoded.

All OpenID Providers MUST support the id\_token and token id\_token response types and all OpenID Providers that are not Self-Issued OPs MUST also support the code response type.

The Client can use any OAuth 2.0 registered response type supported by the OpenID Provider other than token (which provides no ID Token).

client\_id

REQUIRED. OAuth 2.0 Client Identifier.

scope

REQUIRED. Space delimited, case sensitive list of ASCII OAuth 2.0 scope values. OpenID Connect requests MUST contain the openid scope value. OPTIONAL scope values of profile, email, address, phone, and offline\_access are also defined. See [**Section 2.4 (Scope Values)**](#scopes) for more about the scope values defined by this specification.

redirect\_uri

REQUIRED. Redirection URI to which the response will be sent. This MUST be pre-registered with the OpenID Provider. If the Client uses the OAuth implicit grant type, the redirection URI MUST NOT use the http scheme unless the Client is a native application, in which case it MAY use the http: scheme with localhost as the hostname. If the Client only uses the OAuth authorization\_code grant type, the redirection URI MAY use the http scheme, provided that the Client Type is confidential, as defined in Section 2.1 of OAuth 2.0.

state

RECOMMENDED. Opaque value used to maintain state between the request and the callback; it can serve as a protection against XSRF attacks, among other uses.

This specification also defines the following request parameters:

nonce

REQUIRED or OPTIONAL. String value used to associate a Client session with an ID Token, and to mitigate replay attacks. The value is passed through unmodified from the Authorization Request to the ID Token. Sufficient entropy MUST be present in the nonce values used to prevent attackers from guessing values. Use of the nonce is REQUIRED for all requests where an ID Token is returned directly from the Authorization Endpoint. It is OPTIONAL when the ID Token is returned from the Token Endpoint.

display

OPTIONAL. ASCII string value that specifies how the Authorization Server displays the authentication and consent user interface pages to the End-User. The defined values are:

page

The Authorization Server SHOULD display authentication and consent UI consistent with a full User-Agent page view. If the display parameter is not specified this is the default display mode.

popup

The Authorization Server SHOULD display authentication and consent UI consistent with a popup User-Agent window. The popup User-Agent window SHOULD be 450 pixels wide and 500 pixels tall.

touch

The Authorization Server SHOULD display authentication and consent UI consistent with a device that leverages a touch interface. The Authorization Server MAY attempt to detect the touch device and further customize the interface.

wap

The Authorization Server SHOULD display authentication and consent UI consistent with a "feature phone" type display.

prompt

OPTIONAL. Space delimited, case sensitive list of ASCII string values that specifies whether the Authorization Server prompts the End-User for reauthentication and consent. The defined values are:

none

The Authorization Server MUST NOT display any authentication or consent user interface pages. An error is returned if the End-User is not already authenticated or the Client does not have pre-configured consent for the requested Claims. This can be used as a method to check for existing authentication and/or consent.

login

The Authorization Server MUST prompt the End-User for reauthentication.

consent

The Authorization Server MUST prompt the End-User for consent before returning information to the Client.

select\_account

The Authorization Server MUST prompt the End-User to select a user account. This allows an End-User who has multiple accounts at the Authorization Server to select amongst the multiple accounts that they might have current sessions for.

The prompt parameter can be used by the Client to make sure that the End-User is still present for the current session or to bring attention to the request. If this parameter contains none with any other value, an error is returned.

max\_age

OPTIONAL. Maximum Authentication Age. Specifies that the End-User MUST be actively authenticated if the End-User was authenticated longer ago than the specified number of seconds. (The max\_age request parameter corresponds to the OpenID 2.0 [**PAPE (Recordon, D., Jones, M., Bufu, J., Ed., Daugherty, J., Ed., and N. Sakimura, “OpenID Provider Authentication Policy Extension 1.0,” December 2008.)**](#OpenID.PAPE) [OpenID.PAPE] max\_auth\_age request parameter.) When max\_age is used, the ID Token returned MUST include an auth\_time Claim Value.

ui\_locales

OPTIONAL. End-User's preferred languages and scripts for the user interface, represented as a space-separated list of [**BCP47 (Phillips, A. and M. Davis, “Tags for Identifying Languages,” September 2009.)**](#RFC5646) [RFC5646] language tag values, ordered by preference. For instance, the value "fr-CA fr en" represents a preference for French as spoken in Canada, then French (without a region designation), followed by English (without a region designation). An error SHOULD NOT result if some or all of the requested locales are not supported by the OpenID Provider.

claims\_locales

OPTIONAL. End-User's preferred languages and scripts for Claims being returned, represented as a space-separated list of [**BCP47 (Phillips, A. and M. Davis, “Tags for Identifying Languages,” September 2009.)**](#RFC5646) [RFC5646] language tag values, ordered by preference. An error SHOULD NOT result if some or all of the requested locales are not supported by the OpenID Provider.

id\_token\_hint

OPTIONAL. Previously issued ID Token passed to the Authorization Server as a hint about the End-User's current or past authenticated session with the Client. This SHOULD be present when prompt=none is used. If the End-User identified by the ID Token is logged in or is logged in by the request, then the Authorization Server returns a positive response; otherwise, it SHOULD return a negative response. The Authorization Server need not be listed as an audience of the ID Token when it is used as an id\_token\_hint value.

If the ID Token received by the RP is encrypted, the Client MUST decrypt the signed ID Token contained within the encrypted ID Token. The Client MAY re-encrypt the signed ID token to the Authentication Server using a key that enables the server to decrypt the ID Token.

For a Self-Issued ID Token, the sub (subject) of the signed ID Token MUST be sent as the kid (Key ID) of the JWE.

login\_hint

OPTIONAL. Hint to the Authorization Server about the login identifier the End-User might use to log in (if necessary). This hint can be used by an RP if it first asks the End-User for their e-mail address (or other identifier) and then wants to pass that value as a hint to the discovered authorization service. It is RECOMMENDED that the hint value match the value used for discovery. This value MAY also be a phone number in the format specified for the phone\_number Claim. The use of this parameter is left to the OP's discretion.

acr\_values

OPTIONAL. Requested Authentication Context Class Reference values. Space-separated string that specifies the acr values that the Authorization Server MUST use for processing requests from this Client. The Authentication Context Class satisfied by the authentication performed is returned as the acr Claim Value, as specified in [**Section 2.1.2.1 (ID Token)**](#id_token).

claims

OPTIONAL. This parameter is used to request that specific Claims be returned. The value is a JSON object, as specified in [**Section 2.6 (Claims Request)**](#ClaimsRequest).

registration

OPTIONAL. This parameter is used by the Client to provide information about itself to a Self-Issued OP that would normally be provided to an OP during Dynamic Client Registration, as specified in [**Section 6.2.1 (Providing Additional Registration Information)**](#SelfIssuedRegistrationRequest). The registration parameter SHOULD NOT be used when the OP is not a Self-Issued OP.

request

OPTIONAL. This parameter enables OpenID Connect requests to be passed in a single, self-contained parameter and to be signed and optionally encrypted. The parameter value is a Request Object value, as specified in [**Section 2.9 (Request Object)**](#RequestObject). It represents the request as a JWT whose Claims are the request parameters above.

When the request parameter is used, the OpenID Connect request parameter values contained in the JWT supersede those passed using the OAuth 2.0 request syntax. Even if a scope parameter is present in the Request Object value, a scope parameter MUST always be passed using the OAuth 2.0 request syntax containing the openid scope value to indicate to the underlying OAuth 2.0 logic that this is an OpenID Connect request.

request\_uri

OPTIONAL. This parameter enables OpenID Connect requests to be passed by reference, rather than by value. The request\_uri value is a URL using the https scheme referencing a resource containing a Request Object value, which is a JWT containing the request parameters. This parameter is used identically to the request parameter, other than that the Request Object value is retrieved from the specified URL, rather than passed by value. See [**Section 2.10 (Using the "request\_uri" Parameter)**](#RequestUriParameter) for more information on using the request\_uri parameter.

|  |
| --- |
| [TOC](#toc) |

### 2.1.2.  Authorization Response

An Authorization Response is a message returned from the OP's Authorization Endpoint in response to the Authorization Request by the RP.

This specification only describes [OAuth 2.0 Bearer Token Usage (**Jones, M. and D. Hardt, “The OAuth 2.0 Authorization Framework: Bearer Token Usage,” October 2012.**)](#RFC6750) [RFC6750]. The OAuth 2.0 response parameter token\_type MUST be set to Bearer unless another Token Type has been negotiated with the Client.

When the response\_type in the request is code, the Authorization Response MUST return the parameters defined in Section 4.1.2 of [OAuth 2.0 (**Hardt, D., “The OAuth 2.0 Authorization Framework,” October 2012.**)](#RFC6749) [RFC6749].

When the response\_type includes other values, they MUST be returned as defined by their registration. The id\_token and token id\_token response types are defined in [OAuth 2.0 Multiple Response Type Encoding Practices (**de Medeiros, B., Scurtescu, M., and P. Tarjan, “OAuth 2.0 Multiple Response Type Encoding Practices,” June 2013.**)](#OAuth.Responses) [OAuth.Responses].

|  |
| --- |
| [TOC](#toc) |

### 2.1.2.1.  ID Token

The ID Token is a security token that contains Claims about the authentication event and other requested Claims. The ID Token is represented as a [JSON Web Token (JWT) (**Jones, M., Bradley, J., and N. Sakimura, “JSON Web Token (JWT),” May 2013.**)](#JWT) [JWT].

The ID Token is used to manage the authentication event and user identifier and is scoped to a particular Client via the aud (audience) and nonce Claims.

The following Claims are used within the ID Token:

iss

REQUIRED. Issuer Identifier for the Issuer of the response.

sub

REQUIRED. Subject identifier. A locally unique and never reassigned identifier within the Issuer for the End-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. Audience(s) that this ID Token is intended for. It MUST contain the OAuth 2.0 client\_id of the Relying Party as an audience value. It MAY also contain identifiers for other audiences.

exp

REQUIRED. Expiration time on or after which the ID Token MUST NOT be accepted for processing. The processing of this parameter requires that the current date/time MUST be before the expiration date/time listed in the value. Implementers MAY provide for some small leeway, usually no more than a few minutes, to account for clock skew. The time is represented as the number of seconds from 1970-01-01T0:0:0Z as measured in UTC until the date/time. See [**RFC 3339 (Klyne, G., Ed. and C. Newman, “Date and Time on the Internet: Timestamps,” July 2002.)**](#RFC3339) [RFC3339] for details regarding date/times in general and UTC in particular.

iat

REQUIRED. Time at which the JWT was issued. The time is represented as the number of seconds from 1970-01-01T0:0:0Z as measured in UTC until the date/time.

auth\_time

OPTIONAL or REQUIRED. Time when the End-User authentication occurred. The time is represented as the number of seconds from 1970-01-01T0:0:0Z as measured in UTC until the date/time. When a max\_age request is made or when auth\_time is requested as an Essential Claim, then this Claim is REQUIRED. (The auth\_time Claim semantically corresponds to the OpenID 2.0 [**PAPE (Recordon, D., Jones, M., Bufu, J., Ed., Daugherty, J., Ed., and N. Sakimura, “OpenID Provider Authentication Policy Extension 1.0,” December 2008.)**](#OpenID.PAPE) [OpenID.PAPE] auth\_time response parameter.)

nonce

OPTIONAL or REQUIRED. String value used to associate a Client session with an ID Token, and to mitigate replay attacks. The value is passed through unmodified from the Authorization Request to the ID Token. If present in the ID Token, Clients MUST verify that the nonce Claim Value is equal to the value of the nonce parameter sent in the Authorization Request. If present in the Authorization Request, Authorization Servers MUST include a nonce Claim in the ID Token with the Claim Value being the nonce value sent in the Authorization Request. Authorization Servers SHOULD perform no other processing on nonce values used. Use of the nonce is REQUIRED for all requests where an ID Token is returned directly from the Authorization Endpoint. It is OPTIONAL when the ID Token is returned from the Token Endpoint.

at\_hash

OPTIONAL or REQUIRED. Access Token hash value. If the ID Token is issued from the Authorization Endpoint with an access\_token, this is REQUIRED. This is OPTIONAL when the ID Token is issued from the Token Endpoint. Its value is the base64url encoding of the left-most half of the hash of the octets of the ASCII representation of the access\_token value, where the hash algorithm used is the hash algorithm used in the alg parameter of the ID Token's [**JWS (Jones, M., Bradley, J., and N. Sakimura, “JSON Web Signature (JWS),” May 2013.)**](#JWS) [JWS] header. For instance, if the alg is RS256, hash the access\_token value with SHA-256, then take the left-most 128 bits and base64url encode them.

c\_hash

Sometimes REQUIRED. Code hash value. If the ID Token is issued from the Authorization Endpoint with a code, this is REQUIRED. Its value is the base64url encoding of the left-most half of the hash of the octets of the ASCII representation of the code value, where the hash algorithm used is the hash algorithm used in the alg parameter of the ID Token's [**JWS (Jones, M., Bradley, J., and N. Sakimura, “JSON Web Signature (JWS),” May 2013.)**](#JWS) [JWS] header. For instance, if the alg is HS512, hash the code value with SHA-512, then take the left-most 256 bits and base64url encode them.

acr

OPTIONAL. Authentication Context Class Reference. String specifying an Authentication Context Class Reference value that identifies the Authentication Context Class that the authentication performed satisfied. The value "0" indicates the End-User authentication did not meet the requirements of [**ISO/IEC 29115 (International Organization for Standardization, “ISO/IEC FDIS 29115 -- Information technology - Security techniques - Entity authentication assurance framework,” December 2012.)**](#ISO29115) [ISO29115] level 1. Authentication using a long-lived browser cookie, for instance, is one example where the use of "level 0" is appropriate. Authentications with level 0 SHOULD never be used to authorize access to any resource of any monetary value. (This corresponds to the OpenID 2.0 [**PAPE (Recordon, D., Jones, M., Bufu, J., Ed., Daugherty, J., Ed., and N. Sakimura, “OpenID Provider Authentication Policy Extension 1.0,” December 2008.)**](#OpenID.PAPE) [OpenID.PAPE] nist\_auth\_level 0.) An absolute URI or a [**registered name (Johansson, L., “An IANA Registry for Level of Assurance (LoA) Profiles,” August 2012.)**](#RFC6711) [RFC6711] MAY be used as an acr value. The definition of particular values to be used in the acr Claim is beyond the scope of this specification. Parties using this claim will need to agree upon the meanings of the values used, which may be context-specific.

amr

OPTIONAL. Authentication Methods References. JSON array of strings that are identifiers for authentication methods used in the authentication. For instance, values might indicate that both password and OTP authentication methods were used. The definition of particular values to be used in the amr Claim is beyond the scope of this specification. Parties using this claim will need to agree upon the meanings of the values used, which may be context-specific.

azp

OPTIONAL or REQUIRED. Authorized Party - the party to which the ID Token was issued. If present, it MUST contain the OAuth 2.0 client\_id of the Relying Party. This Claim is only REQUIRED when the party requesting the ID Token is not the same as the sole audience of the ID Token. It MAY be included even when the Authorized Party is the same as the sole audience. The azp value is a case sensitive string containing a StringOrURI value.

sub\_jwk

NOT RECOMMENDED or REQUIRED. Public key value used to check the signature of an ID Token issued by a Self-Issued OpenID Provider, as specified in [**Section 6 (Self-Issued OpenID Provider)**](#self_issued). The key is a bare key in JWK format (not an X.509 certificate value). Use of the sub\_jwk Claim is REQUIRED when the OP is a Self-Issued OP and is NOT RECOMMENDED when the OP is not Self-Issued.

The JWT MAY contain other Claims. Any Claims used that are not understood MUST be ignored.

ID Tokens MUST be signed using [JWS (**Jones, M., Bradley, J., and N. Sakimura, “JSON Web Signature (JWS),” May 2013.**)](#JWS) [JWS] and OPTIONALLY both signed and then encrypted using [JWS (**Jones, M., Bradley, J., and N. Sakimura, “JSON Web Signature (JWS),” May 2013.**)](#JWS) [JWS] and [JWE (**Jones, M., Rescorla, E., and J. Hildebrand, “JSON Web Encryption (JWE),” May 2013.**)](#JWE) [JWE] respectively, thereby providing authentication, integrity, non-repudiation, and optionally, confidentiality, per [Section 9.13 (**Signing and Encryption Order**)](#signing_order).

ID Tokens SHOULD NOT use the JWS or JWE x5u, x5c, jku, or jwk header parameter fields. Instead, key values and key references used for ID Tokens are communicated in advance using Discovery and Registration parameters.

Clients MUST directly validate the ID Token per [Section 4.2 (**ID Token Validation**)](#id.token.validation).

The following is a non-normative example of a base64url decoded ID Token (with line wraps for display purposes only):

{ "iss": "https://server.example.com", "sub": "24400320", "aud": "s6BhdRkqt3", "nonce": "n-0S6\_WzA2Mj", "exp": 1311281970, "iat": 1311280970, "auth\_time": 1311280969, "acr": "urn:mace:incommon:iap:silver", "at\_hash": "MTIzNDU2Nzg5MDEyMzQ1Ng" }

|  |
| --- |
| [TOC](#toc) |

### 2.1.3.  Authorization Error Response

If the End-User denies the access request or if the request fails, the OP (Authorization Server) informs the RP (Client) by using the Error Response parameters defined in Sections 4.1.2.1 or 4.2.2.1 of [OAuth 2.0 (**Hardt, D., “The OAuth 2.0 Authorization Framework,” October 2012.**)](#RFC6749) [RFC6749], according to the response\_type used.

In addition to the error codes defined in Sections 4.1.2.1 and 4.2.2.1 of OAuth 2.0, this specification also defines the following error codes:

interaction\_required

The Authorization Server requires End-User interaction of some form to proceed. This error MAY be returned when the prompt parameter in the Authorization Request is set to none to request that the Authorization Server SHOULD NOT display any user interfaces to the End-User, but the Authorization Request cannot be completed without displaying a user interface for End-User interaction.

login\_required

The Authorization Server requires End-User authentication. This error MAY be returned when the prompt parameter in the Authorization Request is set to none to request that the Authorization Server SHOULD NOT display any user interfaces to the End-User, but the Authorization Request cannot be completed without displaying a user interface for user authentication.

session\_selection\_required

The End-User is REQUIRED to select a session at the Authorization Server. The End-User MAY be authenticated at the Authorization Server with different associated accounts, but the End-User did not select a session. This error MAY be returned when the prompt parameter in the Authorization Request is set to none to request that the Authorization Server SHOULD NOT display any user interfaces to the End-User, but the Authorization Request cannot be completed without displaying a user interface to prompt for a session to use.

consent\_required

The Authorization Server requires End-User consent. This error MAY be returned when the prompt parameter in the Authorization Request is set to none to request that the Authorization Server SHOULD NOT display any user interfaces to the End-User, but the Authorization Request cannot be completed without displaying a user interface for End-User consent.

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.

registration\_not\_supported

The OP does not support use of the registration parameter.

request\_not\_supported

The OP does not support use of the request parameter.

request\_uri\_not\_supported

The OP does not support use of the request\_uri parameter.

|  |
| --- |
| [TOC](#toc) |

### 2.2.  Token Endpoint

The RP (Client) sends an Access Token Request to the Token Endpoint to obtain an Access Token Response, which MAY include an Access Token, a Refresh Token, an ID Token, and other results.

|  |
| --- |
| [TOC](#toc) |

### 2.2.1.  Client Authentication

During Client Registration, the RP (Client) MAY register an authentication method. If no method is registered, the default method of client\_secret\_basic MUST be used.

The Supported options are:

client\_secret\_basic

Clients that have received a client\_secret value from the Authorization Server, authenticate with the Authorization Server in accordance with Section 3.2.1 of [**OAuth 2.0 (Hardt, D., “The OAuth 2.0 Authorization Framework,” October 2012.)**](#RFC6749) [RFC6749] using HTTP Basic authentication scheme.

client\_secret\_post

Clients that have received a client\_secret value from the Authorization Server, authenticate with the Authorization Server in accordance with Section 3.2.1 of [**OAuth 2.0 (Hardt, D., “The OAuth 2.0 Authorization Framework,” October 2012.)**](#RFC6749) [RFC6749] by including the Client Credentials in the request body.

client\_secret\_jwt

Clients that have received a client\_secret value from the Authorization Server create a JWT using an HMAC SHA algorithm, such as HMAC SHA-256. The HMAC (Hash-based Message Authentication Code) is calculated using the octets of the UTF-8 representation of the client\_secret as the shared key.

The Client authenticates in accordance with Section 2.2 of [**OAuth JWT Bearer Token Profiles (Jones, M., Campbell, B., and C. Mortimore, “JSON Web Token (JWT) Bearer Token Profiles for OAuth 2.0,” March 2013.)**](#OAuth.JWT) [OAuth.JWT] and [**OAuth 2.0 Assertion Profile (Campbell, B., Mortimore, C., Jones, M., and Y. Goland, “Assertion Framework for OAuth 2.0,” March 2013.)**](#OAuth.Assertions) [OAuth.Assertions]. The JWT MUST contain the following REQUIRED Claim Values and MAY contain the following OPTIONAL Claim Values:

iss

REQUIRED. Issuer. This MUST contain the client\_id of the OAuth Client.

sub

REQUIRED. Subject. This MUST contain the client\_id of the OAuth Client.

aud

REQUIRED. Audience. The aud (audience) Claim. Value that identifies the Authorization Server as an intended audience. The Authorization Server MUST verify that it is an intended audience for the token. The Audience SHOULD be the URL of the Authorization Server's Token Endpoint.

jti

REQUIRED. JWT ID. A unique identifier for the token. The JWT ID MAY be used by implementations requiring message de-duplication for one-time use assertions.

exp

REQUIRED. Expiration time on or after which the ID Token MUST NOT be accepted for processing.

iat

OPTIONAL. Time at which the JWT was issued.

The JWT MAY contain other Claims. Any Claims used that are not understood MUST be ignored.

The authentication token MUST be sent as the value of the [**[OAuth.Assertions] (Campbell, B., Mortimore, C., Jones, M., and Y. Goland, “Assertion Framework for OAuth 2.0,” March 2013.)**](#OAuth.Assertions) client\_assertion parameter.

The value of the [**[OAuth.Assertions] (Campbell, B., Mortimore, C., Jones, M., and Y. Goland, “Assertion Framework for OAuth 2.0,” March 2013.)**](#OAuth.Assertions) client\_assertion\_type parameter MUST be "urn:ietf:params:oauth:client-assertion-type:jwt-bearer", per [**[OAuth.JWT] (Jones, M., Campbell, B., and C. Mortimore, “JSON Web Token (JWT) Bearer Token Profiles for OAuth 2.0,” March 2013.)**](#OAuth.JWT).

private\_key\_jwt

Clients that have registered a public key sign a JWT using that key. The Client authenticates in accordance with Section 2.2 of [**OAuth JWT Bearer Token Profiles (Jones, M., Campbell, B., and C. Mortimore, “JSON Web Token (JWT) Bearer Token Profiles for OAuth 2.0,” March 2013.)**](#OAuth.JWT) [OAuth.JWT] and [**OAuth 2.0 Assertion Profile (Campbell, B., Mortimore, C., Jones, M., and Y. Goland, “Assertion Framework for OAuth 2.0,” March 2013.)**](#OAuth.Assertions) [OAuth.Assertions]. The JWT MUST contain the following REQUIRED Claim Values and MAY contain the following OPTIONAL Claim Values:

iss

REQUIRED. Issuer. This MUST contain the client\_id of the OAuth Client.

sub

REQUIRED. Subject. This MUST contain the client\_id of the OAuth Client.

aud

REQUIRED. Audience. The aud (audience) Claim. Value that identifies the Authorization Server as an intended audience. The Authorization Server MUST verify that it is an intended audience for the token. The Audience SHOULD be the URL of the Authorization Server's Token Endpoint.

jti

REQUIRED. JWT ID. A unique identifier for the token. The JWT ID MAY be used by implementations requiring message de-duplication for one-time use assertions.

exp

REQUIRED. Expiration time on or after which the ID Token MUST NOT be accepted for processing.

iat

OPTIONAL. Time at which the JWT was issued.

The JWT MAY contain other Claims. Any Claims used that are not understood MUST be ignored.

The authentication token MUST be sent as the value of the [**[OAuth.Assertions] (Campbell, B., Mortimore, C., Jones, M., and Y. Goland, “Assertion Framework for OAuth 2.0,” March 2013.)**](#OAuth.Assertions) client\_assertion parameter.

The value of the [**[OAuth.Assertions] (Campbell, B., Mortimore, C., Jones, M., and Y. Goland, “Assertion Framework for OAuth 2.0,” March 2013.)**](#OAuth.Assertions) client\_assertion\_type parameter MUST be "urn:ietf:params:oauth:client-assertion-type:jwt-bearer", per [**[OAuth.JWT] (Jones, M., Campbell, B., and C. Mortimore, “JSON Web Token (JWT) Bearer Token Profiles for OAuth 2.0,” March 2013.)**](#OAuth.JWT).

For example (with line wraps for display purposes only):

POST /token HTTP/1.1 Host: server.example.com Content-Type: application/x-www-form-urlencoded grant\_type=authorization\_code& code=i1WsRn1uB1& client\_id=s6BhdRkqt3& client\_assertion\_type= urn%3Aietf%3Aparams%3Aoauth%3Aclient-assertion-type%3Ajwt-bearer& client\_assertion=PHNhbWxwOl ... ZT

|  |
| --- |
| [TOC](#toc) |

### 2.2.2.  Access Token Request

The Client obtains an Access Token by authenticating with the Authorization Server and presenting its Authorization Grant (in the form of an Authorization Code or Refresh Token).

In addition to the Client authentication parameters, if this is a Refresh Token Request, the Client MUST send the additional parameters specified in Section 6 of [OAuth 2.0 (**Hardt, D., “The OAuth 2.0 Authorization Framework,” October 2012.**)](#RFC6749) [RFC6749]. Otherwise, the Client MUST send the request parameters as specified in Section 4.1.3 of [OAuth 2.0 (**Hardt, D., “The OAuth 2.0 Authorization Framework,” October 2012.**)](#RFC6749) [RFC6749].

|  |
| --- |
| [TOC](#toc) |

### 2.2.3.  Access Token Response

After receiving and validating a valid and authorized Access Token Request from the Client, the Authorization Server returns a successful response that includes an Access Token and an ID Token. The parameters in the successful response are defined in Section 4.1.4 of [OAuth 2.0 (**Hardt, D., “The OAuth 2.0 Authorization Framework,” October 2012.**)](#RFC6749) [RFC6749].

This specification only describes [OAuth 2.0 Bearer Token Usage (**Jones, M. and D. Hardt, “The OAuth 2.0 Authorization Framework: Bearer Token Usage,” October 2012.**)](#RFC6750) [RFC6750]. The OAuth 2.0 response parameter token\_type MUST be set to Bearer unless another Token Type has been negotiated with the Client. Servers SHOULD support [OAuth 2.0 Bearer Token Usage (**Jones, M. and D. Hardt, “The OAuth 2.0 Authorization Framework: Bearer Token Usage,” October 2012.**)](#RFC6750) [RFC6750] for interoperability. For security reasons Servers MAY only allow Clients to register specific token\_type. Clients MUST support [OAuth 2.0 Bearer Token Usage (**Jones, M. and D. Hardt, “The OAuth 2.0 Authorization Framework: Bearer Token Usage,” October 2012.**)](#RFC6750) [RFC6750] and MAY support other token\_type.

In addition to the OAuth 2.0 response parameters, the following parameters MUST be included in the response if the grant\_type value is authorization\_code and the Authorization Request scope parameter contained openid:

id\_token

ID Token value associated with the authenticated session.

An id\_token MUST be returned when the grant\_type value is authorization\_code and MAY be returned when other grant types are used.

Following is a non-normative example:

{ "access\_token": "SlAV32hkKG", "token\_type": "Bearer", "refresh\_token": "8xLOxBtZp8", "expires\_in": 3600, "id\_token": "eyJ0 ... NiJ9.eyJ1c ... I6IjIifX0.DeWt4Qu ... ZXso" }

As in the [OAuth 2.0 (**Hardt, D., “The OAuth 2.0 Authorization Framework,” October 2012.**)](#RFC6749) [RFC6749], Clients SHOULD ignore unrecognized response parameters.

If an ID Token is returned as a result of a token refresh request, the following requirements apply:

* its iss Claim value MUST be the same as in the ID Token issued when the original authentication occurred,
* its sub Claim value MUST be the same as in the ID Token issued when the original authentication occurred,
* its iat Claim MUST represent the time that the new ID Token is issued,
* its aud Claim value MUST be the same as in the ID Token issued when the original authentication occurred,
* if the ID Token contains an auth\_time Claim, its value MUST represent the time of the original authentication - not the time that the new ID token is issued,
* its azp Claim value MUST be the same as in the ID Token issued when the original authentication occurred; if no azp Claim was present in the original ID Token, one MUST NOT be present in the new ID Token, and
* otherwise, the same rules apply as apply when issuing an ID Token at the time of the original authentication.

|  |
| --- |
| [TOC](#toc) |

### 2.2.4.  Access Token Error Response

If the Token Request is invalid or unauthorized, the Authorization Server constructs the error response. The parameters of the Token Error Response are defined as in Section 5.2 of [OAuth 2.0 (**Hardt, D., “The OAuth 2.0 Authorization Framework,” October 2012.**)](#RFC6749) [RFC6749].

|  |
| --- |
| [TOC](#toc) |

### 2.3.  UserInfo Endpoint

The UserInfo Endpoint is an OAuth 2.0 Protected Resource that returns Claims about the authenticated End-User. The location of the UserInfo Endpoint MUST be a URL using the https scheme, which MAY contain port, path, and query parameter components.

|  |
| --- |
| [TOC](#toc) |

### 2.3.1.  UserInfo Request

Clients send requests to the UserInfo Endpoint to obtain Claims about the End-User. The UserInfo Endpoint is an [OAuth 2.0 (**Hardt, D., “The OAuth 2.0 Authorization Framework,” October 2012.**)](#RFC6749) [RFC6749] Protected Resource that complies with the [OAuth 2.0 Bearer Token Usage (**Jones, M. and D. Hardt, “The OAuth 2.0 Authorization Framework: Bearer Token Usage,” October 2012.**)](#RFC6750) [RFC6750] specification. The Access Token SHOULD be sent using the Authorization header field. The following parameters are defined for use in UserInfo Requests:

access\_token

REQUIRED. Access Token obtained from an OpenID Connect Authorization Request.

|  |
| --- |
| [TOC](#toc) |

### 2.3.2.  UserInfo Response

The UserInfo Claims MUST be returned as the members of a JSON object unless a signed or encrypted response was requested during Client Registration. The Claims defined in [Section 2.5 (**Standard Claims**)](#StandardClaims) can be returned, as can additional Claims not specified there.

If a Claim is not returned, that Claim Name SHOULD be omitted from the JSON object representing the Claims; it SHOULD NOT be present with a null or empty string value.

The sub (subject) Claim MUST always be returned in the UserInfo Response.

NOTE: The UserInfo Endpoint response is not guaranteed to be about the End-User identified by the sub (subject) element of the ID Token. The sub Claim in the UserInfo Endpoint response MUST be verified to exactly match the sub Claim in the ID Token before using additional UserInfo Endpoint Claims.

|  |
| --- |
| [TOC](#toc) |

### 2.3.3.  UserInfo Error Response

When an error condition occurs, the UserInfo Endpoint returns an Error Response as defined in Section 3 of [OAuth 2.0 Bearer Token Usage (**Jones, M. and D. Hardt, “The OAuth 2.0 Authorization Framework: Bearer Token Usage,” October 2012.**)](#RFC6750) [RFC6750].

|  |
| --- |
| [TOC](#toc) |

### 2.4.  Scope Values

OpenID Connect Clients use scope values as defined in 3.3 of [OAuth 2.0 (**Hardt, D., “The OAuth 2.0 Authorization Framework,” October 2012.**)](#RFC6749) [RFC6749] to specify what access privileges are being requested for Access Tokens. The scopes associated with Access Tokens determine what resources will be available when they are used to access OAuth 2.0 protected endpoints. For OpenID Connect, scopes can be used to request that specific sets of information be made available as Claim Values. This specification describes only the scope values used by OpenID Connect.

OpenID Connect allows additional scope values to be defined and used. Scope values used that are not understood by an implementation SHOULD be ignored.

Claims requested by the following scopes are treated by Authorization Servers as Voluntary Claims.

OpenID Connect defines the following scope values:

openid

REQUIRED. Informs the Authorization Server that the Client is making an OpenID Connect request. If the openid scope value is not present, the behavior is entirely unspecified.

profile

OPTIONAL. This scope value requests access to the End-User's default profile Claims, which are: name, family\_name, given\_name, middle\_name, nickname, preferred\_username, profile, picture, website, gender, birthdate, zoneinfo, locale, and updated\_at.

email

OPTIONAL. This scope value requests access to the email and email\_verified Claims.

address

OPTIONAL. This scope value requests access to the address Claim.

phone

OPTIONAL. This scope value requests access to the phone\_number and phone\_number\_verified Claims.

offline\_access

OPTIONAL. This scope value requests that an OAuth 2.0 Refresh Token be issued that can be used to obtain an Access Token that grants access to the End-User's UserInfo Endpoint even when the End-User is not present (not logged in).

Multiple scope values MAY be used by creating a space delimited, case sensitive list of ASCII scope values.

The Claims requested by the profile, email, address, and phone scope values are returned from the UserInfo Endpoint, as described in [Section 2.3.2 (**UserInfo Response**)](#UserInfoResponse), when a response\_type value is used that results in an Access Token being issued. However, when the response\_type value used is id\_token (which issues no Access Token), the resulting Claims are returned in the ID Token.

In some cases, the End-User will be given the option to have the OpenID Provider decline to provide some or all information requested by Clients. To minimize the amount of information that the End-User is being asked to disclose, a Client can elect to only request a subset of the information available from the UserInfo Endpoint.

The following is a non-normative example of a scope Request.

scope=openid profile email phone

|  |
| --- |
| [TOC](#toc) |

### 2.5.  Standard Claims

This specification defines a set of standard Claims. They can be requested to be returned either in the UserInfo Response or in the ID Token.

|  |  |  |
| --- | --- | --- |
| **Member** | **Type** | **Description** |
| sub | string | Subject - Identifier for the End-User at the Issuer. |
| name | string | End-User's full name in displayable form including all name parts, possibly including titles and suffixes, ordered according to the End-User's locale and preferences. |
| given\_name | string | Given name(s) or first name(s) of the End-User. Note that in some cultures, people can have multiple given names; all can be present, with the names being separated by space characters. |
| family\_name | string | Surname(s) or last name(s) of the End-User. Note that in some cultures, people can have multiple family names; all can be present, with the names being separated by space characters. |
| middle\_name | string | Middle name(s) of the End-User. Note that in some cultures, people can have multiple middle names; all can be present, with the names being separated by space characters. Also note that in some cultures, middle names are not used. |
| nickname | string | Casual name of the End-User that MAY or MAY not be the same as the given\_name. For instance, a nickname value of Mike might be returned alongside a given\_name value of Michael. |
| preferred\_username | string | Shorthand name that the End-User wishes to be referred to at the RP, such as janedoe or j.doe. This value MAY be any valid JSON string including special characters such as @, /, or whitespace. This value MUST NOT be relied upon to be unique by the RP. (See [**Section 2.5.3 (Claim Stability and Uniqueness)**](#claim.stability).) |
| profile | string | URL of the End-User's profile page. The contents of this Web page SHOULD be about the End-User. |
| picture | string | URL of the End-User's profile picture. This URL MUST refer to an image file (for example, a PNG, JPEG, or GIF image file), rather than to a Web page containing an image. Note that this URL SHOULD specifically reference a profile photo of the End-User suitable for displaying when describing the End-User, rather than an arbitrary photo taken by the End-User. |
| website | string | URL of the End-User's Web page or blog. This Web page SHOULD contain information published by the End-User or an organization that the End-User is affiliated with. |
| email | string | End-User's preferred e-mail address. Its value MUST conform to the [**RFC 5322 (Resnick, P., Ed., “Internet Message Format,” October 2008.)**](#RFC5322) [RFC5322] addr-spec syntax. This value MUST NOT be relied upon to be unique by the RP, as discussed in [**Section 2.5.3 (Claim Stability and Uniqueness)**](#claim.stability). |
| email\_verified | boolean | True if the End-User's e-mail address has been verified; otherwise false. When this Claim Value is true, this means that the OP took affirmative steps to ensure that this e-mail address was controlled by the End-User at the time the verification was performed. The means by which an e-mail address is verified is context-specific, and dependent upon the trust framework or contractual agreements within which the parties are operating. |
| gender | string | End-User's gender. Values defined by this specification are female and male. Other values MAY be used when neither of the defined values are applicable. |
| birthdate | string | End-User's birthday, represented as an [**ISO 8601:2004 (International Organization for Standardization, “ISO 8601:2004. Data elements and interchange formats - Information interchange - Representation of dates and times,” 2004.)**](#ISO8601-2004) [ISO8601‑2004] YYYY-MM-DD format. The year MAY be 0000, indicating that it is omitted. To represent only the year, YYYY format is allowed. Note that depending on the underlying platform's date related function, providing just year can result in varying month and day, so the implementers need to take this factor into account to correctly process the dates. |
| zoneinfo | string | String from zoneinfo [**[zoneinfo] (Public Domain, “The tz database,” June 2011.)**](#zoneinfo) time zone database representing the End-User's time zone. For example, Europe/Paris or America/Los\_Angeles. |
| locale | string | End-User's locale, represented as a [**BCP47 (Phillips, A. and M. Davis, “Tags for Identifying Languages,” September 2009.)**](#RFC5646) [RFC5646] language tag. This is typically an [**ISO 639-1 Alpha-2 (International Organization for Standardization, “ISO 639-1:2002. Codes for the representation of names of languages -- Part 1: Alpha-2 code,” 2002.)**](#ISO639-1) [ISO639‑1] language code in lowercase and an [**ISO 3166-1 Alpha-2 (International Organization for Standardization, “ISO 3166-1:1997. Codes for the representation of names of countries and their subdivisions -- Part 1: Country codes,” 1997.)**](#ISO3166-1) [ISO3166‑1] country code in uppercase, separated by a dash. For example, en-US or fr-CA. As a compatibility note, some implementations have used an underscore as the separator rather than a dash, for example, en\_US; Implementations MAY choose to accept this locale syntax as well. |
| phone\_number | string | End-User's preferred telephone number. [**E.164 (International Telecommunication Union, “E.164: The international public telecommunication numbering plan,” 2010.)**](#E.164) [E.164] is RECOMMENDED as the format of this Claim, for example, +1 (425) 555-1212 or +56 (2) 687 2400. If the phone number contains an extension, it is RECOMMENDED that the extension be represented using the [**RFC 3966 (Schulzrinne, H., “The tel URI for Telephone Numbers,” December 2004.)**](#RFC3966) [RFC3966] extension syntax, for example, +1 (604) 555-1234;ext=5678. |
| phone\_number\_verified | boolean | True if the End-User's phone number has been verified; otherwise false. When this Claim Value is true, this means that the OP took affirmative steps to ensure that this phone number was controlled by the End-User at the time the verification was performed. The means by which a phone number is verified is context-specific, and dependent upon the trust framework or contractual agreements within which the parties are operating. When true, the phone\_number Claim MUST be in E.164 format and any extensions MUST be represented in RFC 3966 format. |
| address | JSON object | End-User's preferred address. The value of the address member is a [**JSON (Crockford, D., “The application/json Media Type for JavaScript Object Notation (JSON),” July 2006.)**](#RFC4627) [RFC4627] structure containing some or all of the members defined in [**Section 2.5.1 (Address Claim)**](#address_claim). |
| updated\_at | number | Time the End-User's information was last updated. The time is represented as the number of seconds from 1970-01-01T0:0:0Z as measured in UTC until the date/time. |

|  |
| --- |
| **Table 1: Reserved Member Definitions** |

For privacy reasons, OpenID Providers MAY elect to not return values for some requested Claims.

The sub (subject) Claim in the UserInfo Endpoint response MUST exactly match the sub Claim in the ID Token, before using additional UserInfo Endpoint Claims.

The UserInfo Endpoint MUST return Claims in JSON format unless a different format was specified during Registration [[OpenID.Registration] (**Sakimura, N., Bradley, J., and M. Jones, “OpenID Connect Dynamic Client Registration 1.0,” May 2013.**)](#OpenID.Registration). The UserInfo Endpoint MAY return Claims in JWT format, which can be signed and/or encrypted. The UserInfo Endpoint MUST return a content-type header to indicate the format that is being returned. The following are accepted content types:

|  |  |
| --- | --- |
| **Content-Type** | **Format Returned** |
| application/json | plain text JSON object |
| application/jwt | JSON Web Token (JWT) |

The following is a non-normative example of such a response:

{ "sub": "248289761001", "name": "Jane Doe", "given\_name": "Jane", "family\_name": "Doe", "preferred\_username": "j.doe", "email": "janedoe@example.com", "picture": "http://example.com/janedoe/me.jpg" }

|  |
| --- |
| [TOC](#toc) |

### 2.5.1.  Address Claim

The components of a physical mailing address. Implementations MAY return only a subset of the fields of an address, depending upon the information available and the End-User's privacy preferences. For example, the country and region might be returned without returning more fine-grained address information.

Implementations MAY return just the full address as a single string in the formatted sub-field, or they MAY return just the individual component fields using the other sub-fields, or they MAY return both. If both variants are returned, they SHOULD be describing the same address, with the formatted address indicating how the component fields are combined.

formatted

Full mailing address, formatted for display or use on a mailing label. This field MAY contain multiple lines, separated by newline characters. This is the primary member of this Claim value, for the purposes of sorting and filtering.

street\_address

Full street address component, which MAY include house number, street name, Post Office Box, and multi-line extended street address information. This field MAY contain multiple lines, separated by newline characters.

locality

City or locality component.

region

State, province, prefecture or region component.

postal\_code

Zip code or postal code component.

country

Country name component.

|  |
| --- |
| [TOC](#toc) |

### 2.5.2.  Claims Languages and Scripts

Human-readable Claim Values and Claim Values that reference human-readable values MAY be represented in multiple languages and scripts. To specify the languages and scripts, [BCP47 (**Phillips, A. and M. Davis, “Tags for Identifying Languages,” September 2009.**)](#RFC5646) [RFC5646] language tags are added to member names, delimited by a # character. For example, family\_name#ja-Kana-JP expresses the Family Name in Katakana in Japanese, which is commonly used to index and represent the phonetics of the Kanji representation of the same represented as family\_name#ja-Hani-JP. As another example, both website and website#de Claim Values might be returned, referencing a Web site in an unspecified language and a Web site in German.

Since Claim Names are case sensitive, it is strongly RECOMMENDED that language tag values used in Claim Names be spelled using the character case with which they are registered in the [IANA Language Subtag Registry (**Internet Assigned Numbers Authority (IANA), “Language Subtag Registry,” 2005.**)](#IANA.Language) [IANA.Language]. In particular, normally language names are spelled with lowercase characters, region names are spelled with uppercase characters, and languages are spelled with mixed case characters. However, since BCP47 language tag values are case insensitive, implementations SHOULD interpret the language tag values supplied in a case insensitive manner.

Per the recommendations in BCP47, language tag values for Claims SHOULD only be as specific as necessary. For instance, using fr might be sufficient in many contexts, rather than fr-CA or fr-FR. Where possible, OPs SHOULD try to match requested Claim locales with Claims it has. For instance, if the Client asks for a Claim with a de (German) language tag and the OP has a value tagged with de-CH (Swiss German) and no generic German value, it would be appropriate for the OP to return the Swiss German value to the Client. (This intentionally moves as much of the complexity of language tag matching to the OP as possible, to simplify Clients.)

A claims\_locales request can be used to specify the preferred languages and scripts to use for the returned Claims. [Section 2.6.2 (**Languages and Scripts for Individual Claims**)](#IndividualClaimsLanguages) describes how to request that specific Claims use particular languages and scripts.

When the OP determines, either through the claims\_locales parameter, or by other means, that the End-User and Client are requesting Claims in only one set of languages and scripts, it is RECOMMENDED that OPs return Claims without language tags when they employ this language and script. It is also RECOMMENDED that Clients be written in a manner that they can handle and utilize Claims using language tags.

|  |
| --- |
| [TOC](#toc) |

### 2.5.3.  Claim Stability and Uniqueness

The sub (subject) and iss (issuer) Claims are the only Claims that a Client can rely upon as a stable identifier for the End-User, since the sub Claim MUST be locally unique and never reassigned within the Issuer for a particular End-User, as described in [Section 2.1.2.1 (**ID Token**)](#id_token). Therefore, the only guaranteed unique identifier for a given End-User is the combination of the iss Claim and the sub Claim.

All other Claims carry no such guarantees across different issuers in terms of stability over time or uniqueness across users, and Issuers are permitted to apply local restrictions and policies. For instance, an Issuer MAY re-use an email Claim value across different End-Users at different points in time, and the claimed email address for a given End-User MAY change over time. Therefore, other Claims such as email, phone\_number, and preferred\_username and MUST NOT be used as unique identifiers for the End-User.

|  |
| --- |
| [TOC](#toc) |

### 2.5.4.  Additional Claims

While this specification defines only small set of Claims as standard Claims, other Claims MAY be used in conjunction with the standard Claims. When using such Claims, it is RECOMMENDED that collision resistant names be used for the Claim Names, as described in Section 4.2 (Public Claim Names) of the [JSON Web Token (JWT) (**Jones, M., Bradley, J., and N. Sakimura, “JSON Web Token (JWT),” May 2013.**)](#JWT) [JWT] specification. Alternatively, Private Claim Names can be safely used when naming conflicts are unlikely to arise, as described in 4.3 of the JWT specification. Or, if specific additional Claims will have broad and general applicability, they can be registered with Reserved Claim Names, per Sections 4.1 and 9.1 of the JWT specification.

|  |
| --- |
| [TOC](#toc) |

### 2.6.  Claims Request

The claims parameter requests that specific Claims be returned from the UserInfo Endpoint and/or in the ID Token. It is represented as a JSON object containing lists of Claims being requested from these locations. Properties of the Claims being requested MAY also be specified.

Support for the claims parameter is OPTIONAL. Should an OP not support this parameter and an RP uses it, the OP SHOULD return a set of Claims to the RP that it believes would be useful to the RP and the End-User using whatever heuristics it believes are appropriate. The claims\_parameter\_supported Discovery result indicates whether the OP supports this parameter.

The claims parameter value is represented in an OAuth 2.0 request as UTF-8 encoded JSON (which ends up being form-urlencoded when passed as an OAuth parameter). When used in a Request Object value, per [Section 2.9 (**Request Object**)](#RequestObject), the JSON is used as the value of the claims member.

The top-level members of the Claims request JSON object are:

userinfo

OPTIONAL. Requests that the listed individual Claims be returned from the UserInfo Endpoint. If present, the listed Claims are being requested to be added to any Claims that are being requested using scope values. If not present, the Claims being requested from the UserInfo Endpoint are only those requested using scope values.

When the userinfo member is used, the request MUST also use a response\_type value that results in an Access Token being issued to use at the UserInfo Endpoint.

id\_token

OPTIONAL. Requests that the listed individual Claims be returned in the ID Token. If present, the listed Claims are being requested to be added to the default Claims in the ID Token. If not present, the default ID Token Claims are requested.

Other members MAY be present. Any members used that are not understood MUST be ignored.

An example Claims request is as follows:

{ "userinfo": { "given\_name": {"essential": true}, "nickname": null, "email": {"essential": true}, "email\_verified": {"essential": true}, "picture": null, "http://example.info/claims/groups": null }, "id\_token": { "auth\_time": {"essential": true}, "acr": {"values": ["urn:mace:incommon:iap:silver"] } } }

Note that a Claim that is not in the standard set defined in [**Section 2.5 (Standard Claims)**](#StandardClaims), the (example) http://example.info/claims/groups Claim, is being requested. Using the claims parameter is the only way to request Claims outside the standard set. It is also the only way to request specific combinations of the standard Claims that cannot be specified using scope values.

|  |
| --- |
| [TOC](#toc) |

### 2.6.1.  Individual Claims Requests

The userinfo and id\_token members of the claims request both are JSON objects with the names of the individual Claims being requested as the member names. The member values MUST be one of the following:

null

Indicates that this Claim is being requested in the default manner. In particular, this is a Voluntary Claim. For instance, the Claim request:

"given\_name": null

requests the given\_name Claim in the default manner.

JSON Object

Used to provide additional information about the Claim being requested. This specification defines the following members:

essential

OPTIONAL. Indicates whether the Claim being requested is an Essential Claim. If the value is true, this indicates that the Claim is an Essential Claim. For instance, the Claim request:

"auth\_time": {"essential": true}

can be used to specify that it is Essential to return an auth\_time Claim Value.

If the value is false, it indicates that it is a Voluntary Claim. The default is false.

By requesting Claims as Essential Claims the Client indicates to the End-User that releasing these Claims will ensure a smooth authorization for the specific task requested by the End-User. Note that even if the Claims are not available because the End-User did not authorize their release or they are not present, the Authorization Server MUST NOT generate an error when Claims are not returned, whether they are Essential or Voluntary.

value

OPTIONAL. Requests that the Claim be returned with a particular value. For instance the Claim request:

"sub": {"value": "248289761001"}

can be used to specify that the request apply to the End-User with subject identifier 248289761001.

The value of the value member MUST be a valid value for the Claim being requested. Definitions of individual Claims can include requirements on how and whether the value qualifier is to be used when requesting that Claim.

values

OPTIONAL. Requests that the Claim be returned with one of a set of values, with the values appearing in order of preference. For instance the Claim request:

"acr": {"essential": true, "values": ["urn:mace:incommon:iap:silver", "urn:mace:incommon:iap:bronze"]}

specifies that it is Essential that the acr Claim be returned with either the value urn:mace:incommon:iap:silver or urn:mace:incommon:iap:bronze.

The values in the values member array MUST be valid values for the Claim being requested. Definitions of individual Claims can include requirements on how and whether the values qualifier is to be used when requesting that Claim.

Other members MAY be defined to provide additional information about the requested Claims. Any members used that are not understood MUST be ignored.

Note that when the claims request parameter is supported, the scope values that request Claims, as defined in [Section 2.4 (**Scope Values**)](#scopes), are effectively shorthand methods for requesting sets of individual Claims. For example, using the scope value openid email and a response\_type that returns an Access Token is equivalent to using the scope value openid and the following request for individual Claims.

Equivalent of using the email scope value:

{ "userinfo": { "email": null, "email\_verified": null } }

|  |
| --- |
| [TOC](#toc) |

### 2.6.2.  Languages and Scripts for Individual Claims

As described in [Section 2.5.2 (**Claims Languages and Scripts**)](#ClaimsLanguagesAndScripts), human-readable Claims values and Claim Values that reference human-readable values MAY be represented in multiple languages and scripts. Within a request for individual Claims, requested languages and scripts for particular Claims MAY be requested by including Claim Names that contain #-separated [BCP47 (**Phillips, A. and M. Davis, “Tags for Identifying Languages,” September 2009.**)](#RFC5646) [RFC5646] language tags in the Claims request, using the Claim Name syntax specified in [Section 2.5.2 (**Claims Languages and Scripts**)](#ClaimsLanguagesAndScripts). For example, a Family Name in Katakana in Japanese can be requested using the Claim Name family\_name#ja-Kana-JP and a Kanji representation of the Family Name in Japanese can be requested using the Claim Name family\_name#ja-Hani-JP. A German-language Web site can be requested with the Claim Name website#de.

If an OP receives a request for human-readable Claims in a language and script that it doesn't have, any versions of those Claims returned that don't use the requested language and script SHOULD use a language tag in the Claim Name.

|  |
| --- |
| [TOC](#toc) |

### 2.7.  Claim Types

The UserInfo Endpoint MAY return the following three types of Claims:

Normal Claims

Claims that are directly asserted by the OpenID Provider.

Aggregated Claims

Claims that are asserted by a Claims Provider other than the OpenID Provider but are returned by OpenID Provider.

Distributed Claims

Claims that are asserted by a Claims Provider other than the OpenID Provider but are returned as references by the OpenID Provider.

The UserInfo Endpoint MUST support Normal Claims.

Aggregated and Distributed Claims support is OPTIONAL.

|  |
| --- |
| [TOC](#toc) |

### 2.7.1.  Normal Claims

Normal Claims are represented as members in a JSON object. The Claim Name is the member name and the Claim Value is the member value.

The following is a non-normative response containing Normal Claims:

{ "name": "Jane Doe" "given\_name": "Jane", "family\_name": "Doe", "email": "janedoe@example.com", "picture": "http://example.com/janedoe/me.jpg" }

|  |
| --- |
| [TOC](#toc) |

### 2.7.2.  Aggregated and Distributed Claims

Aggregated and distributed Claims are represented by using special \_claim\_names and \_claim\_sources members of the JSON object containing the Claims.

\_claim\_names

JSON object whose member names are the Claim Names for the Aggregated and Distributed Claims. The member values are references to the member names in the \_claim\_sources member from which the actual Claim Values can be retrieved.

\_claim\_sources

JSON object whose member names are referenced by the member values of the \_claim\_names member. The member values contain sets of Aggregated Claims or reference locations for Distributed Claims. The member values can have one of the following formats depending on whether it is providing Aggregated or Distributed Claims:

Aggregated Claims

JSON object that MUST contain the JWT member whose value is a [**JWT (Jones, M., Bradley, J., and N. Sakimura, “JSON Web Token (JWT),” May 2013.)**](#JWT) [JWT] that MUST contain all the Claims in the \_claim\_names object that references the corresponding \_claim\_sources member. Other members MAY be present. Any members used that are not understood MUST be ignored.

JWT

REQUIRED. JWT containing Claim Values.

The JWT SHOULD NOT contain a sub (subject) Claim unless its value is an identifier for the End-User at the Claims Provider (and not for the OpenID Provider or another party); this typically means that a sub Claim SHOULD NOT be provided.

Distributed Claims

JSON object that contains the following members and values:

endpoint

REQUIRED. OAuth 2.0 resource endpoint from which the associated Claim can be retrieved. The endpoint URL MUST return the Claim as a JWT.

access\_token

OPTIONAL. Access Token enabling retrieval of the Claims from the endpoint URL by using the [**OAuth 2.0 Bearer Token Usage (Jones, M. and D. Hardt, “The OAuth 2.0 Authorization Framework: Bearer Token Usage,” October 2012.)**](#RFC6750) [RFC6750] protocol. Claims SHOULD be requested using the Authorization Request header field and Claims Providers MUST support this method. If the Access Token is not available, Clients MAY need to retrieve the Access Token out of band or use an a priori Access Token that was negotiated between the Claims Provider and Client, or the Claims Provider MAY reauthenticate the End-User and/or reauthorize the Client.

A sub (subject) Claim SHOULD NOT be returned from the Claims Provider unless its value is an identifier for the End-User at the Claims Provider (and not for the OpenID Provider or another party); this typically means that a sub Claim SHOULD NOT be provided.

|  |
| --- |
| [TOC](#toc) |

### 2.7.2.1.  Example of Aggregated Claims

In this non-normative example, Claims from Claims Provider A are combined with other Claims held by the OpenID provider, with the Claims from Claims Provider A being returned as Aggregated Claims.

In this example, these Claims about Jane Doe have been issued by Claims Provider A:

{ "address": { "street\_address": "1234 Hollywood Blvd.", "locality": "Los Angeles", "region": "CA", "postal\_code": "90210", "country": "US"}, "phone\_number": "+1 (310) 123-4567" }

Claims Provider A signs the JSON Claims, representing them in a signed JWT: jwt\_header.jwt\_part2.jwt\_part3. It is this JWT that is used by the OpenID Provider.

In this example, this JWT containing Jane Doe's Aggregated Claims from Claims Provider A is combined with other Normal Claims, and returned as the following set of Claims:

{ "name": "Jane Doe", "given\_name": "Jane", "family\_name": "Doe", "birthdate": "0000-03-22", "eye\_color": "blue", "email": "janedoe@example.com", "\_claim\_names": { "address": "src1", "phone\_number": "src1" }, "\_claim\_sources": { "src1": {"JWT": "jwt\_header.jwt\_part2.jwt\_part3"} } }

|  |
| --- |
| [TOC](#toc) |

### 2.7.2.2.  Example of Distributed Claims

In this non-normative example, the OpenID Provider combines Normal Claims that it holds with references to Claims held by two different Claims Providers, B and C, incorporating references to some of the Claims held by B and C as Distributed Claims.

In this example, these Claims about Jane Doe are held by Claims Provider B (Jane Doe's bank):

{ "shipping\_address": { "street\_address": "1234 Hollywood Blvd.", "locality": "Los Angeles", "region": "CA", "postal\_code": "90210", "country": "US"}, "payment\_info": "Some\_Card 1234 5678 9012 3456", "phone\_number": "+1 (310) 123-4567" }

Also in this example, this Claim about Jane Doe is held by Claims Provider C (a credit agency):

{ "credit\_score": 650 }

The OpenID Provider returns Jane Doe's Claims along with references to the Distributed Claims from Claims Provider B and Claims Provider C by sending the Access Tokens and URLs of locations from which the Distributed Claims can be retrieved:

{ "name": "Jane Doe", "given\_name": "Jane", "family\_name": "Doe", "email": "janedoe@example.com", "birthdate": "0000-03-22", "eye\_color": "blue", "\_claim\_names": { "payment\_info": "src1", "shipping\_address": "src1", "credit\_score": "src2" }, "\_claim\_sources": { "src1": {"endpoint": "https://bank.example.com/claim\_source"}, "src2": {"endpoint": "https://creditagency.example.com/claims\_here", "access\_token": "ksj3n283dke"} } }

|  |
| --- |
| [TOC](#toc) |

### 2.8.  Subject Identifier Types

The OpenID Provider's Discovery document SHOULD list its supported identifier types in the subject\_types\_supported element. If there is more than one type listed in the array, the Client MAY elect to provide its preferred identifier type using the subject\_type parameter during Registration. The types supported by this specification are:

public

This provides the same sub (subject) value to all Clients. It is the default if the provider has no subject\_types\_supported element in its discovery document.

pairwise

This provides a different sub value to each Client, to prevent correlation of the End-User's activities by Clients without his permission.

|  |
| --- |
| [TOC](#toc) |

### 2.8.1.  Pairwise Identifier Algorithm

The OpenID Provider MUST calculate a unique sub (subject) value for each Sector Identifier. The subject value MUST NOT be reversible by any party other than the OpenID Provider.

Providers who use pairwise sub values SHOULD support the sector\_identifier\_uri in [Dynamic Client Registration (**Sakimura, N., Bradley, J., and M. Jones, “OpenID Connect Dynamic Client Registration 1.0,” May 2013.**)](#OpenID.Registration) [OpenID.Registration]. It provides a way for a group of websites under common administrative control to have consistent pairwise sub values independent of the individual domain names. It also provides a way for Clients to change redirect\_uri domains without having to reregister all of their users.

If the Client has not provided a value for sector\_identifier\_uri in [Dynamic Client Registration (**Sakimura, N., Bradley, J., and M. Jones, “OpenID Connect Dynamic Client Registration 1.0,” May 2013.**)](#OpenID.Registration) [OpenID.Registration], the Sector Identifier used for pairwise identifier calculation is the host component of the registered redirect\_uri. If there are multiple hostnames in the registered redirect\_uris, the Client MUST register a sector\_identifier\_uri.

When a sector\_identifier\_uri is provided, the host component of that URL is used as the Sector Identifier for the pairwise identifier calculation. The value of the sector\_identifier\_uri MUST be a URL using the https scheme that points to a JSON file containing an array of redirect\_uri values. The values of the registered redirect\_uris MUST be included in the elements of the array, or the registration MUST fail.

A number of algorithms can be used by OpenID Providers to calculate pairwise identifiers. Three example methods are:

1. The Sector Identifier can be concatenated with a local account ID and a salt value that is kept secret by the Provider. The concatenated string is then hashed using an appropriate algorithm.   
     
   Calculate sub = SHA-256 ( sector\_identifier | local\_account\_id | salt ).
2. The Sector Identifier can be concatenated with a local account ID and a salt value that is kept secret by the Provider. The concatenated string is then encrypted using an appropriate algorithm.   
     
   Calculate sub = AES-128 ( sector\_identifier | local\_account\_id | salt ).
3. The Issuer creates a Globally Unique Identifier (GUID) for the pair of Sector Identifier and local account ID and stores this value.

|  |
| --- |
| [TOC](#toc) |

### 2.9.  Request Object

The request parameter enables OpenID Connect requests to be passed in a single, self-contained parameter and to be signed and optionally encrypted. It represents the request as a JWT whose Claims are the request parameters specified in [Section 2.1.1.1 (**Request Parameters**)](#RequestParameters). This JWT is called a Request Object.

Support for the request parameter is OPTIONAL. The request\_parameter\_supported Discovery result indicates whether the OP supports this parameter. Should an OP not support this parameter and an RP uses it, the OP MUST return the request\_not\_supported error.

When the request parameter is used, the OpenID Connect request parameter values contained in the JWT supersede those passed using the OAuth 2.0 request syntax. However, some parameters MAY 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.

Even if a scope parameter is present in the Request Object value, a scope parameter MUST always be passed using the OAuth 2.0 request syntax containing the openid scope value to indicate to the underlying OAuth 2.0 logic that this is an OpenID Connect request.

The Request Object MAY be signed or unsigned (plaintext). When it is plaintext, this is indicated by use of the none algorithm [[JWA] (**Jones, M., “JSON Web Algorithms (JWA),” May 2013.**)](#JWA) in the JWS header. If signed, the Request Object SHOULD contain the Claims iss (issuer) and aud (audience) as members, with their semantics being as defined in the [JWT (**Jones, M., Bradley, J., and N. Sakimura, “JSON Web Token (JWT),” May 2013.**)](#JWT) [JWT] specification.

The Request Object MAY also be encrypted using [JWE (**Jones, M., Rescorla, E., and J. Hildebrand, “JSON Web Encryption (JWE),” May 2013.**)](#JWE) [JWE], with nested signing and encryption performed as described in the JWT [[JWT] (**Jones, M., Bradley, J., and N. Sakimura, “JSON Web Token (JWT),” May 2013.**)](#JWT) specification.

request and request\_uri parameters MUST NOT be included in Request Objects.

An example set of Request Object Claims before base64url encoding and JWS signing is as follows:

{ "response\_type": "code id\_token", "client\_id": "s6BhdRkqt3", "redirect\_uri": "https://client.example.org/cb", "scope": "openid", "state": "af0ifjsldkj", "login\_hint": "janedoe@example.org", "max\_age": 86400, "claims": { "userinfo": { "given\_name": {"essential": true}, "nickname": null, "email": {"essential": true}, "email\_verified": {"essential": true}, "picture": null }, "id\_token": { "auth\_time": {"essential": true}, "acr": { "values":["urn:mace:incommon:iap:silver"] } } } }

|  |
| --- |
| [TOC](#toc) |

### 2.10.  Using the "request\_uri" Parameter

The request\_uri parameter enables OpenID Connect 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.

The request\_uri\_parameter\_supported Discovery result indicates whether the OP supports this parameter. Should an OP not support this parameter and an RP uses it, the OP MUST return the request\_uri\_not\_supported error.

When the request\_uri parameter is used, the OpenID Connect request parameter values contained in the referenced JWT supersede those passed using the OAuth 2.0 request syntax. However, some parameters MAY 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.

Even if a scope parameter is present in the referenced Request Object, a scope parameter MUST always be passed using the OAuth 2.0 request syntax containing the openid scope value to indicate to the underlying OAuth 2.0 logic that this is an OpenID Connect request.

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 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.

Note that Clients MAY pre-register request\_uri values using the request\_uris parameter defined in Section 2 of the [OpenID Connect Dynamic Client Registration 1.0 (**Sakimura, N., Bradley, J., and M. Jones, “OpenID Connect Dynamic Client Registration 1.0,” May 2013.**)](#OpenID.Registration) [OpenID.Registration] specification.

|  |
| --- |
| [TOC](#toc) |

### 2.10.1.  "request\_uri" Rationale

There are several reasons that one might choose to use the request\_uri parameter:

1. The set of request parameters can become large, and can exceed browser URI size limitations. Passing the request parameters by reference can solve this problem.
2. Passing a request\_uri value, rather than a complete request by value, can reduce request latency.
3. Most requests for Claims from an RP are constant. The request\_uri is a way of creating and sometimes also signing and encrypting a constant set of request parameters in advance. (The request\_uri value becomes an "artifact" representing a particular fixed set of request parameters.)
4. Pre-registering a fixed set of request parameters at registration time enables OPs to cache and pre-validate the request parameters at registration time, meaning they need not be retrieved at request time.

|  |
| --- |
| [TOC](#toc) |

### 3.  Signatures and Encryption

Depending on the transport through which the messages are sent, the integrity of the message might not be guaranteed and the originator of the message might not be authenticated. To mitigate these risks, Request Object, Token Request, ID Token, and UserInfo Response values MAY utilize [JSON Web Signature (JWS) (**Jones, M., Bradley, J., and N. Sakimura, “JSON Web Signature (JWS),” May 2013.**)](#JWS) [JWS] to sign the contents.

To achieve message confidentiality, Request Object, Token Request, ID Token, and UserInfo Response values MAY use [JSON Web Encryption (JWE) (**Jones, M., Rescorla, E., and J. Hildebrand, “JSON Web Encryption (JWE),” May 2013.**)](#JWE) [JWE] to encrypt the content.

When the message is both signed and encrypted, it MUST be signed first and then encrypted, per [Section 9.13 (**Signing and Encryption Order**)](#signing_order), with nesting performed in the same manner as specified for JWTs [[JWT] (**Jones, M., Bradley, J., and N. Sakimura, “JSON Web Token (JWT),” May 2013.**)](#JWT). Note that all JWE encryption methods perform integrity checking.

|  |
| --- |
| [TOC](#toc) |

### 3.1.  Supported Algorithms

The server advertises its supported signing and encryption algorithms in its discovery document. The algorithm identifiers are specified in [JWA (**Jones, M., “JSON Web Algorithms (JWA),” May 2013.**)](#JWA) [JWA]. The related elements are:

userinfo\_signing\_alg\_values\_supported

JSON array containing a list of the JWS [**[JWS] (Jones, M., Bradley, J., and N. Sakimura, “JSON Web Signature (JWS),” May 2013.)**](#JWS) signing algorithms (alg values) [**[JWA] (Jones, M., “JSON Web Algorithms (JWA),” May 2013.)**](#JWA) supported by the UserInfo Endpoint to encode the Claims in a JWT [**[JWT] (Jones, M., Bradley, J., and N. Sakimura, “JSON Web Token (JWT),” May 2013.)**](#JWT).

userinfo\_encryption\_alg\_values\_supported

JSON array containing a list of the JWE [**[JWE] (Jones, M., Rescorla, E., and J. Hildebrand, “JSON Web Encryption (JWE),” May 2013.)**](#JWE) encryption algorithms (alg values) [**[JWA] (Jones, M., “JSON Web Algorithms (JWA),” May 2013.)**](#JWA) supported by the UserInfo Endpoint to encode the Claims in a JWT [**[JWT] (Jones, M., Bradley, J., and N. Sakimura, “JSON Web Token (JWT),” May 2013.)**](#JWT).

userinfo\_encryption\_enc\_values\_supported

JSON array containing a list of the JWE encryption algorithms (enc values) [**[JWA] (Jones, M., “JSON Web Algorithms (JWA),” May 2013.)**](#JWA) supported by the UserInfo Endpoint to encode the Claims in a JWT [**[JWT] (Jones, M., Bradley, J., and N. Sakimura, “JSON Web Token (JWT),” May 2013.)**](#JWT).

id\_token\_signing\_alg\_values\_supported

JSON array containing a list of the JWS signing algorithms (alg values) supported by the Authorization Server for the ID Token to encode the Claims in a JWT [**[JWT] (Jones, M., Bradley, J., and N. Sakimura, “JSON Web Token (JWT),” May 2013.)**](#JWT).

id\_token\_encryption\_alg\_values\_supported

JSON array containing a list of the JWE encryption algorithms (alg values) supported by the Authorization Server for the ID Token to encode the Claims in a JWT [**[JWT] (Jones, M., Bradley, J., and N. Sakimura, “JSON Web Token (JWT),” May 2013.)**](#JWT).

id\_token\_encryption\_enc\_values\_supported

JSON array containing a list of the JWE encryption algorithms (enc values) supported by the Authorization Server for the ID Token to encode the Claims in a JWT [**[JWT] (Jones, M., Bradley, J., and N. Sakimura, “JSON Web Token (JWT),” May 2013.)**](#JWT).

request\_object\_signing\_alg\_values\_supported

JSON array containing a list of the JWS signing algorithms (alg values) supported by the Authorization Server for Request Object values. Servers SHOULD support none and RS256.

request\_object\_encryption\_alg\_values\_supported

JSON array containing a list of the JWE encryption algorithms (alg values) supported by the Authorization Server for Request Object values.

request\_object\_encryption\_enc\_values\_supported

JSON array containing a list of the JWE encryption algorithms (enc values) supported by the Authorization Server for Request Object values.

token\_endpoint\_auth\_signing\_alg\_values\_supported

JSON array containing a list of the JWS signing algorithms (alg values) supported by the Token Endpoint for the private\_key\_jwt and client\_secret\_jwt methods to encode the JWT [**[JWT] (Jones, M., Bradley, J., and N. Sakimura, “JSON Web Token (JWT),” May 2013.)**](#JWT). Servers SHOULD support RS256.

The Client registers its REQUIRED algorithms for Signing and Encryption using the following Registration parameters:

request\_object\_signing\_alg

OPTIONAL. JWS signature algorithm [**[JWA] (Jones, M., “JSON Web Algorithms (JWA),” May 2013.)**](#JWA) REQUIRED for Request Objects by the Authorization Server. All Request Objects from this client\_id MUST be rejected if not signed by this algorithm. Servers SHOULD support RS256.

userinfo\_signed\_response\_alg

OPTIONAL. JWS signature algorithm [**[JWA] (Jones, M., “JSON Web Algorithms (JWA),” May 2013.)**](#JWA) REQUIRED for UserInfo Responses. If this is specified the response will be [**JWT (Jones, M., Bradley, J., and N. Sakimura, “JSON Web Token (JWT),” May 2013.)**](#JWT) [JWT] serialized.

userinfo\_encrypted\_response\_alg

OPTIONAL. JWE alg algorithm [**[JWA] (Jones, M., “JSON Web Algorithms (JWA),” May 2013.)**](#JWA) REQUIRED for UserInfo Responses. If this is requested in combination with signing, the response MUST be signed first then encrypted, per [**Section 9.13 (Signing and Encryption Order)**](#signing_order). If this is specified, the response will be [**JWT (Jones, M., Bradley, J., and N. Sakimura, “JSON Web Token (JWT),” May 2013.)**](#JWT) [JWT] serialized.

userinfo\_encrypted\_response\_enc

OPTIONAL. JWE enc algorithm [**[JWA] (Jones, M., “JSON Web Algorithms (JWA),” May 2013.)**](#JWA) REQUIRED for UserInfo Responses. If userinfo\_encrypted\_response\_alg is specified the default for this value is A128CBC-HS256.

id\_token\_signed\_response\_alg

OPTIONAL. JWS signature algorithm [**[JWA] (Jones, M., “JSON Web Algorithms (JWA),” May 2013.)**](#JWA) REQUIRED for ID Tokens issued to this client\_id. The default if not specified is RS256. The public key for validating the signature is provided by retrieving the JWK Set referenced by the jwks\_uri element from Discovery.

id\_token\_encrypted\_response\_alg

OPTIONAL. JWE alg algorithm [**[JWA] (Jones, M., “JSON Web Algorithms (JWA),” May 2013.)**](#JWA) REQUIRED for ID Tokens issued to this client\_id. If this is requested, the response MUST be signed then encrypted. The default if not specified is no encryption.

id\_token\_encrypted\_response\_enc

OPTIONAL. JWE enc algorithm [**[JWA] (Jones, M., “JSON Web Algorithms (JWA),” May 2013.)**](#JWA) REQUIRED for ID Tokens issued to this client\_id. If id\_token\_encrypted\_response\_alg is specified the default for this value is A128CBC-HS256.

|  |
| --- |
| [TOC](#toc) |

### 3.2.  Keys

The OpenID Provider provides its public keys during Discovery using the following element:

jwks\_uri

REQUIRED. URL of the OP's JSON Web Key Set [**[JWK] (Jones, M., “JSON Web Key (JWK),” May 2013.)**](#JWK) document. This contains the signing key(s) the Client uses to validate signatures from the OP. The JWK Set MAY also contain the Server's encryption key(s), which are used by Clients to encrypt requests to the Server. When both signing and encryption keys are made available, a use (Key Use) parameter value is REQUIRED for all keys in the document to indicate each key's intended usage.

Likewise, the Client can provide its public keys during Registration using the following element:

jwks\_uri

OPTIONAL. URL for the Client's JSON Web Key Set [**[JWK] (Jones, M., “JSON Web Key (JWK),” May 2013.)**](#JWK) document. If the Client signs requests to the Server, it contains the signing key(s) the Server uses to validate signatures from the Client. The JWK Set MAY also contain the Client's encryption keys(s), which are used by the Server to encrypt responses to the Client. When both signing and encryption keys are made available, a use (Key Use) parameter value is REQUIRED for all keys in the document to indicate each key's intended usage.

When both signing and encryption keys are made available, the use (Key Use) parameter value is REQUIRED for all keys in the JWK Set at the jwks\_uri to indicate each key's intended usage. Although some algorithms allow the same key pair to be used for both signatures and encryption, doing so is NOT RECOMMENDED, as it is less secure.

In both cases, the JWK x5c parameter MAY be used to provide X.509 representations of keys provided. When used, the bare key values MUST still be present and MUST match those in the certificate.

|  |
| --- |
| [TOC](#toc) |

### 3.3.  Signing

The signing party MUST select a signature algorithm based on the supported algorithms of the recipient in [Section 3.1 (**Supported Algorithms**)](#sigenc.alg).

Asymmetric Signatures

When using RSA or ECDSA Signatures, the alg Claim of the JWS header MUST be set to the appropriate algorithm as defined in [**JSON Web Algorithms (Jones, M., “JSON Web Algorithms (JWA),” May 2013.)**](#JWA) [JWA]. The private key MUST be the one associated with the Public Signing Key provided in [**Section 3.2 (Keys)**](#sigenc.key). If there are multiple keys in the referenced JWK document, a kid value MUST be provided in the JWS header. The key usage of the respective keys MUST support signature.

Symmetric Signatures

When using MAC-based signatures, the alg Claim of the JWS header MUST be set to a MAC algorithm, as defined in [**JSON Web Algorithms (Jones, M., “JSON Web Algorithms (JWA),” May 2013.)**](#JWA) [JWA]. The MAC key used is the octets of the UTF-8 representation of the client\_secret value. See [**Section 9.17 (Symmetric Key Entropy)**](#SymmetricKeyEntropy) for a discussion of entropy requirements for client\_secret values.

See [Section 9.18 (**Need for Signed Requests**)](#NeedForSignedRequests) for Security Considerations about the need for signed requests.

|  |
| --- |
| [TOC](#toc) |

### 3.3.1.  Rotation of Asymmetric Signing Keys

Rotation of signing keys can be accomplished with the following approach. The signer publishes its keys in a JWK Set at the jwks\_uri location and includes the kid of the signing key in the JWS header of each message to indicate to the verifier which key is to be used to validate the signature. Keys can be rolled over by periodically adding new keys to the JWK Set at jwks\_uri. The signer can begin using a new key at its discretion and signals the change to the verifier using the kid value. The verifier knows to go back to the jwks\_uri to re-retrieve the keys when it sees an unfamiliar kid value. The JWK Set document at the jwks\_uri SHOULD retain recently decommissioned signing keys for a reasonable period of time to facilitate a smooth transition.

|  |
| --- |
| [TOC](#toc) |

### 3.4.  Encryption

The encrypting party MUST select an encryption algorithm based on the supported algorithms of the recipient in [Section 3.1 (**Supported Algorithms**)](#sigenc.alg). All JWTs MUST be signed before encryption to enable verification of the Issuer.

Asymmetric Encryption: RSA

Use the link registered/discovered in [**Section 3.2 (Keys)**](#sigenc.key) to retrieve the relevant key. If there are multiple keys in the referenced JWK document, a kid value MUST be provided in the JWE header. Use the supported RSA key wrapping algorithm to wrap a random Content Master Key to be used for encrypting the signed JWT. The key usage of the respective keys MUST include encryption.

Asymmetric Encryption: Elliptic Curve

Create an ephemeral Elliptic Curve public key for the epk element of the JWE header. Use the link registered/discovered in [**Section 3.2 (Keys)**](#sigenc.key) to retrieve the relevant key. If there are multiple keys in the referenced JWK document, a kid value MUST be provided in the JWE header. Use the ECDH-ES algorithm to wrap a random Content Master Key to be used for encrypting the signed JWT. The key usage of the respective keys MUST support encryption.

Symmetric Encryption

The symmetric encryption key is derived from the client\_secret value by using a left truncated SHA-256 hash of the octets of the UTF-8 representation of the client\_secret. The SHA-256 value MUST be left truncated to the appropriate bit length for the AES key wrapping algorithm used, for instance, to 128 bits for A128KW. If a key wrapping key with greater than 256 bits is needed, a different method of deriving the key from the client\_secret would have to be defined by an extension.

See [Section 9.19 (**Need for Encrypted Requests**)](#NeedForEncryptedRequests) for Security Considerations about the need for encrypted requests.

|  |
| --- |
| [TOC](#toc) |

### 3.4.1.  Rotation of Asymmetric Encryption Keys

Rotating encryption keys is necessarily a different process than for signing keys because the encrypting party starts the process and thus cannot rely on a change in kid as a signal to know that keys need to change. The encrypting party still uses the kid header in the JWE to tell the decrypting party which private key to use to decrypt, however, the encrypting party needs to first select the most appropriate key from those provided in the JWK Set at jwks\_uri. To rotate keys, the decrypting party can publish new keys at jwks\_uri and remove from the JWK Set those that are being decommissioned. The jwks\_uri SHOULD include a Cache-Control header in the response that contains a max-age directive, as defined in [RFC 2616 (**Fielding, R., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P., and T. Berners-Lee, “Hypertext Transfer Protocol -- HTTP/1.1,” June 1999.**)](#RFC2616) [RFC2616], which allows the encrypting party to safely cache the JWK Set and not have to re-retrieve the document for every encryption event. The decrypting party SHOULD remove decommissioned keys from the JWK Set at jwks\_uri but retain them internally for some reasonable period of time, coordinated with the cache duration, to facilitate a smooth transition between keys by allowing the encrypting party some time to obtain the new keys. The cache duration SHOULD also be coordinated with the issuance of new signing keys as described in [Section 3.3.1 (**Rotation of Asymmetric Signing Keys**)](#rotate.sig.keys).

|  |
| --- |
| [TOC](#toc) |

### 4.  Validation

If any of the validation procedures defined in this specification fail, any operations requiring the information that failed to correctly validate MUST be aborted and the information that failed to validate MUST NOT be used.

|  |
| --- |
| [TOC](#toc) |

### 4.1.  Authorization Request Validation

Authorization Request Validation consists of two main steps: (1) decryption and signature validation of the value of request or the content of request\_uri, and (2) parameter validation. If a Request Object value was sent in the request parameter or by reference in the request\_uri parameter, the Request Object MUST validate as [JWS (**Jones, M., Bradley, J., and N. Sakimura, “JSON Web Signature (JWS),” May 2013.**)](#JWS) [JWS] or [JWE (**Jones, M., Rescorla, E., and J. Hildebrand, “JSON Web Encryption (JWE),” May 2013.**)](#JWE) [JWE] encoded objects, for which nested encryption and signing can be utilized in the manner described in the [JWT (**Jones, M., Bradley, J., and N. Sakimura, “JSON Web Token (JWT),” May 2013.**)](#JWT) [JWT] specification.

|  |
| --- |
| [TOC](#toc) |

### 4.1.1.  Encrypted Request Object

If the Authorization Server has advertised JWE encryption algorithms in the request\_object\_encryption\_alg\_values\_supported and request\_object\_encryption\_enc\_values\_supported elements of its Discovery Document, these are used by the Client to encrypt the JWT.

The Authorization Server MUST decode the JWT in accordance with the [JSON Web Encryption (**Jones, M., Rescorla, E., and J. Hildebrand, “JSON Web Encryption (JWE),” May 2013.**)](#JWE) [JWE] specification. The result MAY be either a signed or unsigned (plaintext) Request Object. In the former case, signature validation MUST be performed as defined in [Section 4.1.2 (**Signed Request Object**)](#signed.req.obj.var).

The Authorization Server MUST return the error if there is a decryption error.

|  |
| --- |
| [TOC](#toc) |

### 4.1.2.  Signed Request Object

To perform Signature Validation, the alg parameter in the JWS header MUST match the value of the request\_object\_signing\_alg set during [Client Registration (**Sakimura, N., Bradley, J., and M. Jones, “OpenID Connect Dynamic Client Registration 1.0,” May 2013.**)](#OpenID.Registration) [OpenID.Registration] or a value that was pre-registered by other means.

The signature MUST be validated against the key registered for that client\_id and algorithm, in accordance with the [JSON Web Signature (**Jones, M., Bradley, J., and N. Sakimura, “JSON Web Signature (JWS),” May 2013.**)](#JWS) [JWS] specification.

The Authorization Server MUST return the Authorization Error Response if there is a signature validation error.

|  |
| --- |
| [TOC](#toc) |

### 4.1.3.  Parameter Validation

The Authorization Server MUST construct the Authorization Request Message from the Request Object value and the OAuth 2.0 Authorization Request 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 this Authorization Request Message, the Authorization Server performs the following steps of the request validation:

1. The Authorization Server MUST validate all the OAuth 2.0 parameters according to the OAuth 2.0 specification.
2. The Authorization Server MUST verify that all the REQUIRED parameters are present.
3. If the sub (subject) Claim as a member of id\_token element is requested with a specific value, the Authorization Server MUST only send a positive response if that user has an active session with the Authorization Server. The Authorization Server MUST NOT reply with an ID Token or Access Token for a different user, even if they have an active session with the Authorization Server.
4. If the acr Claim is requested as an Essential Claim for the ID Token with a values parameter requesting specific Authentication Context Class Reference values, then the Authorization Server MUST return an acr Claim Value that matches one of the requested values. The Authorization Server MAY ask the End-User to re-authenticate with additional factors to meet this requirement. If this is an Essential Claim and the requirement cannot be met, then the Authorization Server MUST treat that outcome as a failed authentication attempt.
5. The Client MAY request this Claim as an optional Claim by using the acr\_values request parameter or by not including "essential": true in the individual acr Claim request. If the Claim is not Essential and the requested value cannot be provided, the Authorization Server SHOULD return the session's current acr as the value of the acr Claim. If the Claim is not Essential, the Authorization Server is not required to provide this Claim in its response.

If the Authorization Server encounters any error, it MUST return the error response.

|  |
| --- |
| [TOC](#toc) |

### 4.2.  ID Token Validation

To validate the ID Token in the Authorization or Token Endpoint Response, the Client MUST do the following:

1. If the Client has provided an id\_token\_encrypted\_response\_alg parameter during Registration, decrypt the ID Token using the key pair specified during Registration.
2. The Client MUST validate that the aud (audience) Claim contains its client\_id value registered at the Issuer identified by the iss (issuer) Claim as an audience. The aud (audience) Claim MAY contain an array with more than one element. The ID Token MUST be rejected if the ID Token does not list the Client as a valid audience, or if it contains additional audiences not trusted by the Client.
3. If the ID Token contains multiple audiences, the Client SHOULD verify that an azp Claim is present.
4. If an azp (authorized party) Claim is present, the Client SHOULD verify and that its client\_id is the Claim value.
5. If the id\_token is received via direct communication between the Client and the Token Endpoint, the TLS server validation MAY be used to validate the issuer in place of checking the token signature. The Client MUST validate the signature of all other ID Tokens according to [**JWS (Jones, M., Bradley, J., and N. Sakimura, “JSON Web Signature (JWS),” May 2013.)**](#JWS) [JWS] using the algorithm specified in the alg parameter of the JWT header.
6. The alg value SHOULD be the default of RS256 or the algorithm sent by the Client in the id\_token\_signed\_response\_alg parameter during Registration.
7. If the alg parameter of the JWT header is a MAC based algorithm such as HS256, HS384, or HS512, the octets of the UTF-8 representation of the client\_secret corresponding to the client\_id contained in the aud (audience) Claim are used as the key to validate the signature. Multiple audiences are not supported for MAC based algorithms.
8. For other Signing algorithms, the Client MUST use the signing key provided in Discovery by the Issuer. The issuer MUST exactly match the value of the iss (issuer) Claim.
9. The current time MUST be less than the value of the exp Claim.
10. The iat Claim can be used to reject tokens that were issued too far away from the current time, limiting the amount of time that nonces need to be stored to prevent attacks. The acceptable range is Client specific.
11. If a nonce value was sent in the Authorization Request, a nonce Claim MUST be present and its value checked to verify that it is the same value as the one that was sent in the Authorization Request. The Client SHOULD check the nonce value for replay attacks. The precise method for detecting replay attacks is Client specific.
12. If the acr Claim was requested, the Client SHOULD check that the asserted Claim Value is appropriate. The meaning and processing of acr Claim Values is out of scope for this specification.
13. If the auth\_time Claim was requested, either through a specific request for this Claim or by using the max\_age parameter, the Client SHOULD check the auth\_time Claim value and request re-authentication if it determines too much time has elapsed since the last End-User authentication.

|  |
| --- |
| [TOC](#toc) |

### 4.3.  UserInfo Response Validation

To validate the UserInfo Response, the Client MUST do the following:

1. If the Client has provided a userinfo\_encrypted\_response\_alg parameter during Registration, decrypt the UserInfo Response using the key pair specified during Registration.
2. If the response was signed, the Client SHOULD validate the signature according to [**JWS (Jones, M., Bradley, J., and N. Sakimura, “JSON Web Signature (JWS),” May 2013.)**](#JWS) [JWS].
3. Check that the OP that responded was the intended OP through a TLS server certificate check, per [**RFC 6125 (Saint-Andre, P. and J. Hodges, “Representation and Verification of Domain-Based Application Service Identity within Internet Public Key Infrastructure Using X.509 (PKIX) Certificates in the Context of Transport Layer Security (TLS),” March 2011.)**](#RFC6125) [RFC6125].

|  |
| --- |
| [TOC](#toc) |

### 4.4.  Access Token Validation

To validate an Access Token issued from the Authorization Endpoint with an ID Token in response to a request containing a response\_type of token id\_token or code token id\_token, the Client SHOULD do the following:

1. Hash the octets of the ASCII representation of the access\_token with the hash algorithm specified in [**JWS (Jones, M., “JSON Web Algorithms (JWA),” May 2013.)**](#JWA) [JWA] for the alg parameter in the ID Token's [**JWS (Jones, M., Bradley, J., and N. Sakimura, “JSON Web Signature (JWS),” May 2013.)**](#JWS) [JWS] header.
2. Take the left-most half of the hash and base64url encode it.
3. The value of at\_hash in the ID Token MUST match the value produced in the previous step if at\_hash is present in the ID Token.

|  |
| --- |
| [TOC](#toc) |

### 4.5.  Code Validation

To validate a code issued from the Authorization Endpoint with an ID Token in response to a request containing a response\_type of code id\_token or code token id\_token, the Client SHOULD do the following:

1. Hash the octets of the ASCII representation of the code with the hash algorithm specified in [**JWS (Jones, M., “JSON Web Algorithms (JWA),” May 2013.)**](#JWA) [JWA] for the alg parameter in the ID Token's [**JWS (Jones, M., Bradley, J., and N. Sakimura, “JSON Web Signature (JWS),” May 2013.)**](#JWS) [JWS] header.
2. Take the left-most half of the hash and base64url encode it.
3. The value of c\_hash in the ID Token MUST match the value produced in the previous step if c\_hash is present in the ID Token.

|  |
| --- |
| [TOC](#toc) |

### 5.  Offline Access

The offline\_access scope value requests that an OAuth 2.0 Refresh Token be issued that can be used to obtain an Access Token that grants access to the End-User's UserInfo Endpoint even when the End-User is not present (not logged in). When offline access is requested, a prompt parameter value of consent MUST be used. The user MUST always explicitly consent to the return of a Refresh Token that enables offline access. A previously saved user consent is not sufficient to grant offline access.

Upon receipt of a scope parameter containing the offline\_access value, the Authorization Server:

* MUST ensure that the prompt parameter contains consent; if the prompt parameter does not contain consent then it MUST ignore the offline\_access request,
* MUST ignore the offline\_access request if the Client is not using a response\_type value that would result in an Authorization Code being returned.
* MUST explicitly receive user consent for all Clients when the registered application\_type is web,
* SHOULD explicitly receive user consent for all Clients when the registered application\_type is native.

The use of Refresh Tokens is not exclusive to the offline\_access use case. The Authorization Server MAY grant Refresh Tokens in other contexts that are beyond the scope of this specification.

|  |
| --- |
| [TOC](#toc) |

### 6.  Self-Issued OpenID Provider

OpenID Connect supports Self-Issued OpenID Providers - personal OPs that issue self-signed ID Tokens. Self-Issued OPs use the special Issuer Identifier https://self-issued.me.

|  |
| --- |
| [TOC](#toc) |

### 6.1.  Self-Issued OpenID Provider Discovery

If the input identifier for the discovery process contains the domain self-issued.me, dynamic discovery is not performed. Instead, then the following static configuration values are used:

{ "authorization\_endpoint": "openid:", "issuer": "https://self-issued.me", "scopes\_supported": ["openid", "profile", "email", "address", "phone"], "response\_types\_supported": ["id\_token"], "subject\_types\_supported": ["pairwise"], "id\_token\_signing\_alg\_values\_supported": ["RS256"], "request\_object\_signing\_alg\_values\_supported": ["none", "RS256"] }

Note: The OpenID Foundation may consider hosting a site https://self-issued.me/ that returns the above static configuration file so that the Client would not need any special treatment for discovery of the Self-Issued OP.

|  |
| --- |
| [TOC](#toc) |

### 6.2.  Self-Issued OpenID Provider Registration

When using a Self-Issued OP, the Client is deemed to have registered with the OP and obtained following Client Registration Response.

client\_id

redirect\_uri value of the Client.

client\_secret\_expires\_at

0

Note: The OpenID Foundation may consider hosting the (stateless) endpoint https://self-issued.me/registration/1.0/ that returns the response above so that the Client would not need to perform any special processing for registration of a Self-Issued OP.

|  |
| --- |
| [TOC](#toc) |

### 6.2.1.  Providing Additional Registration Information

The registration request parameter is used by the Client to provide information about itself to a Self-Issued OP that would normally be provided to an OP during Dynamic Client Registration. The value is a JSON object containing name/value pairs defined in Section 2.1 of the [OpenID Connect Dynamic Client Registration 1.0 (**Sakimura, N., Bradley, J., and M. Jones, “OpenID Connect Dynamic Client Registration 1.0,” May 2013.**)](#OpenID.Registration) [OpenID.Registration] specification. None of this information is REQUIRED by Self-Issued OPs, so the use of this parameter is OPTIONAL.

The registration parameter value is represented in an OAuth 2.0 request as UTF-8 encoded JSON (which ends up being form-urlencoded when passed as an OAuth parameter). When used in a Request Object value, per [Section 2.9 (**Request Object**)](#RequestObject), the JSON is used as the value of the registration member.

The Registration parameters that would typically be used in requests to Self-Issued OPs are policy\_uri, tos\_uri, and logo\_uri. If the Client uses more than one redirect URI, the redirect\_uris parameter would be used to register them. Finally, if the Client is requesting encrypted responses, it would use the jwks\_uri, id\_token\_encrypted\_response\_alg and id\_token\_encrypted\_response\_enc parameters.

|  |
| --- |
| [TOC](#toc) |

### 6.3.  Self-Issued OpenID Provider Request

The Client sends the Authorization Request to the Authorization Endpoint with the following parameters:

response\_type

REQUIRED. Constant string value id\_token.

client\_id

REQUIRED. Client ID value for the Client, which in this case contains the redirect\_uri value of the Client. Since the Client's redirect\_uri URI value is communicated as the Client ID, a redirect\_uri parameter is NOT REQUIRED to also be included in the request.

scope

REQUIRED. scope parameter value, as specified in [**Section 2.4 (Scope Values)**](#scopes).

id\_token\_hint

OPTIONAL. Previously issued ID Token passed to the Authorization Server as a hint about the End-User's current or past authenticated session with the Client. This SHOULD be present when prompt=none is used. If the End-User identified by the ID Token is logged in or is logged in by the request, then the Authorization Server returns a positive response; otherwise, it SHOULD return a negative response.

If the ID Token received by the RP is encrypted, the Client MUST decrypt the signed ID Token contained within the encrypted ID Token. The Client MAY re-encrypt the signed ID token to the Authentication Server using a key that enables the server to decrypt the ID Token. In this case, the sub (subject) of the signed ID Token MUST be sent as the kid (Key ID) of the JWE. Encrypting content to Self-Issued OPs is currently only supported when the OP's JWK key type is RSA and the encryption algorithm used is RSA1\_5.

claims

OPTIONAL. This parameter is used to request that specific Claims be returned. The value is a JSON object, as specified in [**Section 2.6 (Claims Request)**](#ClaimsRequest).

registration

OPTIONAL. This parameter is used by the Client to provide information about itself to a Self-Issued OP that would normally be provided to an OP during Dynamic Client Registration, as specified in [**Section 6.2.1 (Providing Additional Registration Information)**](#SelfIssuedRegistrationRequest).

request

OPTIONAL. Request Object value, as specified in [**Section 2.9 (Request Object)**](#RequestObject). The Request Object MAY be encrypted in a JWE by the Client. In this case, the sub (subject) of a previously issued ID Token for this Client MUST be sent as the kid (Key ID) of the JWE. Encrypting content to Self-Issued OPs is currently only supported when the OP's JWK key type is RSA and the encryption algorithm used is RSA1\_5.

Other parameters MAY be sent. Note that all Claims are returned in the ID Token.

The entire URL MUST NOT exceed 2048 ASCII characters.

Following is a non-normative example (with line wraps for display purposes only):

HTTP/1.1 302 Found Location: openid:// ?response\_type=id\_token &client\_id=https%3A%2F%2Fclient.example.org%2Fcb &scope=openid%20profile &state=af0ifjsldkj &nonce=n-0S6\_WzA2Mj registration=&%7B%22logo\_uri%22%3A%22https%3A%2F%2F client.example.org%2Flogo.png%22%7D

|  |
| --- |
| [TOC](#toc) |

### 6.4.  Self-Issued OpenID Provider Response

The Self-Issued OpenID Provider response is the same as the normal implicit flow response with the following refinements. Since it is an implicit flow response, the response parameters will be returned in a fragment.

1. The iss (issuer) Claim Value is https://self-issued.me.
2. A sub\_jwk Claim is present, with its value being the public key value used to check the signature of the ID Token.
3. The sub (subject) Claim value is the base64url encoded SHA-256 hash of the concatenation of the octets of the UTF-8 representations of the base64url encoded key values in the sub\_jwk Claim. When the kty value is RSA, the key values n and e are concatenated in that order. When the kty value is EC, the key values crv, x, and y are concatenated in that order.
4. No Access Token is returned for accessing a UserInfo Endpoint, so all Claims returned MUST be in the ID Token.

|  |
| --- |
| [TOC](#toc) |

### 6.5.  Self-Issued ID Token Validation

If any of the validation procedures defined in this specification fail, any operations requiring the information that failed to correctly validate MUST be aborted and the information that failed to validate MUST NOT be used.

To validate the ID Token in the Authorization or Token Endpoint Response, the Client MUST do the following:

1. The Client MUST validate that the value of the iss (issuer) Claim is https://self-isued.me. If iss contains a different value, the ID Token is not Self-Issued, and instead it MUST be validated according to [**Section 4.2 (ID Token Validation)**](#id.token.validation).
2. The Client MUST validate that the aud (audience) Claim contains the value of the redirect\_uri that the Client sent in the authentication request as an audience.
3. The Client MUST validate the signature of the ID Token according to [**JWS (Jones, M., Bradley, J., and N. Sakimura, “JSON Web Signature (JWS),” May 2013.)**](#JWS) [JWS] using the algorithm specified in the alg parameter of the JWT header [**[JWT] (Jones, M., Bradley, J., and N. Sakimura, “JSON Web Token (JWT),” May 2013.)**](#JWT), using the key in the sub\_jwk Claim; the key is a bare key in JWK format (not an X.509 certificate value).
4. The alg value SHOULD be the default of RS256. It MAY also be ES256.
5. The Client MUST validate that the sub (subject) Claim value is the base64url encoded SHA-256 hash of the concatenation of the octets of the UTF-8 representations of the base64url encoded key values in the sub\_jwk Claim. When the kty value is RSA, the key values n and e are concatenated in that order. When the kty value is EC, the key values crv, x, and y are concatenated in that order.
6. The current time MUST be less than the value of the exp Claim (possibly allowing for some small leeway to account for clock skew).
7. The iat Claim can be used to reject tokens that were issued too far away from the current time, limiting the amount of time that nonces need to be stored to prevent attacks. The acceptable range is Client specific.
8. If a nonce value was sent in the Authorization Request, a nonce Claim MUST be present and its value of the checked to verify that it is the same value as the one that was sent in the Authorization Request. The Client SHOULD check the nonce value for replay attacks. The precise method for detecting replay attacks is Client specific.

The following is a non-normative example of a base64url decoded Self-Issued ID Token (with line wraps for display purposes only):

{ "iss": "https://self-issued.me", "sub": "wBy8QvHbPzUnL0x63h13QqvUYcOur1X0cbQpPVRqX5k", "aud": "https://client.example.org/cb", "nonce": "n-0S6\_WzA2Mj", "exp": 1311281970, "iat": 1311280970, "sub\_jwk": { "kty":"RSA", "n": "0vx7agoebGcQSuuPiLJXZptN9nndrQmbXEps2aiAFbWhM78LhWx 4cbbfAAtVT86zwu1RK7aPFFxuhDR1L6tSoc\_BJECPebWKRXjBZCiFV4n3oknjhMs tn64tZ\_2W-5JsGY4Hc5n9yBXArwl93lqt7\_RN5w6Cf0h4QyQ5v-65YGjQR0\_FDW2 QvzqY368QQMicAtaSqzs8KJZgnYb9c7d0zgdAZHzu6qMQvRL5hajrn1n91CbOpbI SD08qNLyrdkt-bFTWhAI4vMQFh6WeZu0fM4lFd2NcRwr3XPksINHaQ-G\_xBniIqb w0Ls1jF44-csFCur-kEgU8awapJzKnqDKgw", "e":"AQAB" } }

|  |
| --- |
| [TOC](#toc) |

### 7.  String Operations

Processing some OpenID Connect messages requires comparing values in the messages to known values. For example, the Claim Names returned by the UserInfo Endpoint might be compared to specific Claim Names such as sub. Comparing Unicode strings, however, has significant security implications.

Therefore, comparisons between JSON strings and other Unicode strings MUST be performed as specified below:

1. Remove any JSON applied escaping to produce an array of Unicode code points.
2. [**Unicode Normalization (Davis, M., Whistler, K., and M. Dürst, “Unicode Normalization Forms,” 09 2009.)**](#USA15) [USA15] MUST NOT be applied at any point to either the JSON string or to the string it is to be compared against.
3. Comparisons between the two strings MUST be performed as a Unicode code point to code point equality comparison.

In several places, this specification uses space delimited lists of strings. In all such cases, only the ASCII space character (0x20) MAY be used for this purpose.

|  |
| --- |
| [TOC](#toc) |

### 8.  Implementation Considerations

This specification defines features used by both Relying Parties and OpenID Providers. Features that are mandatory to implement for Relying Parties are already described in the [OpenID Connect Basic Client Profile 1.0 (**Sakimura, N., Bradley, J., Jones, M., de Medeiros, B., and C. Mortimore, “OpenID Connect Basic Client Profile 1.0,” May 2013.**)](#OpenID.Basic) [OpenID.Basic] and [OpenID Connect Implicit Client Profile 1.0 (**Sakimura, N., Bradley, J., Jones, M., de Medeiros, B., Mortimore, C., and E. Jay, “OpenID Connect Implicit Client Profile 1.0,” May 2013.**)](#OpenID.Implicit) [OpenID.Implicit] specifications, and so are not discussed again here.

It is expected that some OpenID Providers will require static, out-of-band configuration of RPs using them, whereas others will support dynamic usage by RPs without a pre-established relationship between them. For that reason, the mandatory-to-implement features for OPs are listed below in two groups: the first for all OPs and the second for "Dynamic" OpenID Providers.

|  |
| --- |
| [TOC](#toc) |

### 8.1.  Mandatory to Implement Features for All OpenID Providers

All OpenID Providers MUST implement the following features defined in this specification. This list augments the set of features that are already listed elsewhere as being "REQUIRED" or are described with a "MUST", and so is not, by itself, a comprehensive set of implementation requirements for OPs.

Signing ID Tokens with RSA SHA-256

OPs MUST support signing ID Tokens with the RSA SHA-256 algorithm (an alg value of RS256).

Prompt Parameter

OPs MUST support the prompt parameter, as defined in [**Section 2.1.1.1 (Request Parameters)**](#RequestParameters), including the specified user interface behaviors such as none and login.

Display Parameter

OPs MUST support the display parameter, as defined in [**Section 2.1.1.1 (Request Parameters)**](#RequestParameters). (Note that the minimum level of support required for this parameter is simply to have its use not result in an error.)

Preferred Locales

OPs MUST support requests for preferred languages and scripts for the user interface and for Claims via the ui\_locales and claims\_locales request parameters, as defined in [**Section 2.1.1.1 (Request Parameters)**](#RequestParameters). (Note that the minimum level of support required for these parameters is simply to have their use not result in errors.)

Authentication Time

OPs MUST support returning the time at which the End-User authenticated via the auth\_time Claim, as defined in [**Section 2.1.2.1 (ID Token)**](#id_token).

Maximum Authentication Age

OPs MUST support enforcing a maximum authentication age via the max\_age parameter, as defined in [**Section 2.1.1.1 (Request Parameters)**](#RequestParameters).

Authentication Context Class Reference

OPs MUST support requests for specific Authentication Context Class Reference values via the acr\_values parameter, as defined in [**Section 2.1.1.1 (Request Parameters)**](#RequestParameters). (Note that the minimum level of support required for this parameter is simply to have its use not result in an error.)

|  |
| --- |
| [TOC](#toc) |

### 8.2.  Mandatory to Implement Features for Dynamic OpenID Providers

In addition to the features listed above, OpenID Providers supporting dynamic establishment of relationships with RPs that they do not have a pre-configured relationship with MUST also implement the following features defined in this and related specifications.

Discovery

These OPs MUST support Discovery, as defined in [**OpenID Connect Discovery 1.0 (Sakimura, N., Bradley, J., Jones, M., and E. Jay, “OpenID Connect Discovery 1.0,” May 2013.)**](#OpenID.Discovery) [OpenID.Discovery].

Dynamic Registration

These OPs MUST support Dynamic Client Registration, as defined in [**OpenID Connect Dynamic Client Registration 1.0 (Sakimura, N., Bradley, J., and M. Jones, “OpenID Connect Dynamic Client Registration 1.0,” May 2013.)**](#OpenID.Registration) [OpenID.Registration].

UserInfo Endpoint

All dynamic OPs that issue Access Tokens MUST support the UserInfo Endpoint, as defined in [**Section 2.3 (UserInfo Endpoint)**](#userinfo). (Self-Issued OPs do not issue Access Tokens.)

Public Keys Published as Bare Keys

These OPs MUST publish their public keys as bare keys, rather than in in X.509 format.

Request URI

These OPs MUST support requests made using a Request Object value that is retrieved from a Request URI that is provided with the request\_uri parameter, as defined in [**Section 2.1.1.1 (Request Parameters)**](#RequestParameters).

|  |
| --- |
| [TOC](#toc) |

### 8.3.  Related Specifications

This specification is an abstract specification. It needs to be bound to a protocol to be used in practice. One such example of protocol binding is:

* [**OpenID Connect Standard 1.0 (Sakimura, N., Bradley, J., Jones, M., de Medeiros, B., Mortimore, C., and E. Jay, “OpenID Connect Standard 1.0,” May 2013.)**](#OpenID.Standard) [OpenID.Standard] - Protocol binding for the full set of OpenID Connect messages

These related OpenID Connect specifications MAY OPTIONALLY be used in combination with this specification to provide additional functionality:

* [**OpenID Connect Discovery 1.0 (Sakimura, N., Bradley, J., Jones, M., and E. Jay, “OpenID Connect Discovery 1.0,” May 2013.)**](#OpenID.Discovery) [OpenID.Discovery] - Dynamic discovery for user and Authorization Server endpoints and information
* [**OpenID Connect Dynamic Client Registration 1.0 (Sakimura, N., Bradley, J., and M. Jones, “OpenID Connect Dynamic Client Registration 1.0,” May 2013.)**](#OpenID.Registration) [OpenID.Registration] - Dynamic registration of OpenID Connect Clients with OpenID Providers
* [**OpenID Connect Basic Client Profile 1.0 (Sakimura, N., Bradley, J., Jones, M., de Medeiros, B., and C. Mortimore, “OpenID Connect Basic Client Profile 1.0,” May 2013.)**](#OpenID.Basic) [OpenID.Basic] - Protocol binding for a subset of the OpenID Connect Messages that is intended for use by basic Web-based Relying Parties using the OAuth authorization\_code grant type.
* [**OpenID Connect Implicit Client Profile 1.0 (Sakimura, N., Bradley, J., Jones, M., de Medeiros, B., Mortimore, C., and E. Jay, “OpenID Connect Implicit Client Profile 1.0,” May 2013.)**](#OpenID.Implicit) [OpenID.Implicit] - Protocol binding for a subset of the OpenID Connect Messages that is intended for use by basic Web-based Relying Parties using the OAuth implicit grant type.
* [**OpenID Connect Session Management 1.0 (Sakimura, N., Bradley, J., Jones, M., de Medeiros, B., Mortimore, C., and E. Jay, “OpenID Connect Session Management 1.0,” May 2013.)**](#OpenID.Session) [OpenID.Session] - Session management for OpenID Connect sessions

|  |
| --- |
| [TOC](#toc) |

### 9.  Security Considerations

[OAuth 2.0 Threat Model and Security Considerations (**Lodderstedt, T., McGloin, M., and P. Hunt, “OAuth 2.0 Threat Model and Security Considerations,” January 2013.**)](#RFC6819) [RFC6819] provides an extensive list of threats and controls that applies to this standard as well. [ISO/IEC 29115 (**International Organization for Standardization, “ISO/IEC FDIS 29115 -- Information technology - Security techniques - Entity authentication assurance framework,” December 2012.**)](#ISO29115) [ISO29115] also provides threats and controls that implementers need to take into account. In addition, this standard provides additional control measures listed below.

|  |
| --- |
| [TOC](#toc) |

### 9.1.  Request Disclosure

If appropriate measures are not taken, a request might be disclosed to an attacker, posing security and privacy threats.

In addition to what is stated in Section 5.1.1 of [[RFC6819] (**Lodderstedt, T., McGloin, M., and P. Hunt, “OAuth 2.0 Threat Model and Security Considerations,” January 2013.**)](#RFC6819), this standard provides a way to provide the confidentiality of the request end to end through the use of request or request\_uri parameters, where the content of the request is an encrypted JWT with the appropriate key and cipher. This protects even against a compromised User-Agent in the case of indirect request.

|  |
| --- |
| [TOC](#toc) |

### 9.2.  Server Masquerading

A malicious Server might masquerade as the legitimate server using various means. To detect such an attack, the Client needs to authenticate the server.

In addition to what is stated in Section 5.1.2 of [[RFC6819] (**Lodderstedt, T., McGloin, M., and P. Hunt, “OAuth 2.0 Threat Model and Security Considerations,” January 2013.**)](#RFC6819), this standard provides a way to authenticate the Server through either the use of Signed or Encrypted JWTs with an appropriate key and cipher.

|  |
| --- |
| [TOC](#toc) |

### 9.3.  Token Manufacture/Modification

An Attacker might generate a bogus token or modify the token content (such as the authentication or attribute statements) of an existing parseable token, causing the RP to grant inappropriate access to the Client. For example, an Attacker might modify the parseable token to extend the validity period; a Client might modify the parseable token to have access to information that they should not be able to view.

There are two ways to mitigate this attack:

1. The token can be digitally signed by the OP. The Relying Party SHOULD validate the digital signature to verify that it was issued by a legitimate OP.
2. The token can be sent over a protected channel such as TLS. See [**Section 9.15 (TLS Requirements)**](#TLS_requirements) for more information on using TLS. In this specification, the token is always sent over a TLS protected channel. Note however, that this measure is only a defense against third party attackers and is not applicable to the case where the Client is the attacker.

|  |
| --- |
| [TOC](#toc) |

### 9.4.  Server Response Disclosure

The server response might contain authentication and attribute statements that include sensitive Client information. Disclosure of the response contents can make the Client vulnerable to other types of attacks.

The server response disclosure can be mitigated in the following two ways:

1. Using the code response type. The response is sent over a TLS protected channel, where the Client is authenticated by the client\_id and client\_secret.
2. For other response types, the signed response can be encrypted with the Client's public key or a shared secret as an encrypted JWT with an appropriate key and cipher.

|  |
| --- |
| [TOC](#toc) |

### 9.5.  Server Response Repudiation

A response might be repudiated by the server if the proper mechanisms are not in place. For example, if a Server does not digitally sign a response, the Server can claim that it was not generated through the services of the Server.

To mitigate this threat, the response MAY be digitally signed by the Server using a key that supports non-repudiation. The Client SHOULD validate the digital signature to verify that it was issued by a legitimate Server and its integrity is intact.

|  |
| --- |
| [TOC](#toc) |

### 9.6.  Request Repudiation

Since it is possible for a compromised or malicious Client to send a request to the wrong party, a Client that was authenticated using only a bearer token can repudiate any transaction.

To mitigate this threat, the Server MAY require that the request be digitally signed by the Client using a key that supports non-repudiation. The Server SHOULD validate the digital signature to verify that it was issued by a legitimate Client and the integrity is intact.

|  |
| --- |
| [TOC](#toc) |

### 9.7.  Access Token Redirect

An Attacker uses the Access Token generated for one resource to obtain access to a second resource.

To mitigate this threat, the Access Token SHOULD be audience and scope restricted. One way of implementing it is to include the identifier of the resource for whom it was generated as audience. The resource verifies that incoming tokens include its identifier as the audience of the token.

|  |
| --- |
| [TOC](#toc) |

### 9.8.  Token Reuse

An Attacker attempts to use a one-time use token such as an Authorization Code that has already been used once with the intended Resource. To mitigate this threat, the token SHOULD include a timestamp and a short validity lifetime. The Relying Party then checks the timestamp and lifetime values to ensure that the token is currently valid.

Alternatively, the server MAY record the state of the use of the token and check the status for each request.

|  |
| --- |
| [TOC](#toc) |

### 9.9.  Eavesdropping or Leaking Authorization Codes (Secondary Authenticator Capture)

In addition to the attack patterns described in Section 4.4.1.1 of [[RFC6819] (**Lodderstedt, T., McGloin, M., and P. Hunt, “OAuth 2.0 Threat Model and Security Considerations,” January 2013.**)](#RFC6819), an Authorization Code can be captured in the User-Agent where the TLS session is terminated if the User-Agent is infected by malware. However, capturing it is not useful as long as the profile uses either Client authentication or an encrypted response.

|  |
| --- |
| [TOC](#toc) |

### 9.10.  Token Substitution

Token Substitution is a class of attacks in which a malicious user swaps various tokens, including swapping an Authorization Code for a legitimate user with another token that the attacker has.

The implicit flow of [OAuth 2.0 (**Hardt, D., “The OAuth 2.0 Authorization Framework,” October 2012.**)](#RFC6749) [RFC6749] is not designed to mitigate this risk. In Section 10.16, it normatively requires that any use of the authorization process as a form of delegated End-User authentication to the Client MUST NOT use the implicit flow without employing additional security mechanisms that enable the Client to determine whether the Access Token was issued for its use.

In OpenID Connect, this is mitigated through mechanisms provided through the ID Token. The ID Token is a signed security token that provides Claims such as iss (issuer), sub (subject), aud (audience), azp (authorized party), at\_hash (access token hash), and c\_hash (code hash). Using the ID Token, the Client is capable of detecting the Token Substitution Attack.

The c\_hash in the ID Token enables Clients to prevent code substitution.

Also, a malicious user may attempt to impersonate a more privileged user by subverting the communication channel between the Authorization Endpoint and Client, or the Token Endpoint and Client, for example by swapping the code or reordering the messages, to convince the Token Endpoint that the attacker's authorization grant corresponds to a grant sent on behalf of a more privileged user.

For HTTP bindings such as OpenID Connect Standard 1.0, the responses to Token Requests are bound to the corresponding requests by message order in HTTP, as both the response containing the token and requests are protected by TLS, which will detect and prevent packet reordering.

When designing another binding of OpenID Connect Messages to a protocol incapable of strongly binding Token Endpoint requests to responses, additional mechanisms to address this issue MUST be utilized. One such mechanism could be to include an ID Token with a c\_hash Claim in the token request and response.

|  |
| --- |
| [TOC](#toc) |

### 9.11.  Timing Attack

A timing attack allows the attacker to obtain an unnecessary large amount of information through the elapsed time differences in the code paths taken by successful and unsuccessful decryption operations or successful and unsuccessful signature validation of a message. It can be used to reduce the effective key length of the cipher used.

Implementations SHOULD NOT terminate the validation process at the instant of the finding an error but SHOULD continue running until all the octets have been processed to avoid this attack.

|  |
| --- |
| [TOC](#toc) |

### 9.12.  Other Crypto Related Attacks

There are various crypto related attacks possible depending on the method used for encryption and signature / integrity checking. Implementers need to consult the Security Considerations for the [JWT (**Jones, M., Bradley, J., and N. Sakimura, “JSON Web Token (JWT),” May 2013.**)](#JWT) [JWT] specification and specifications that it references to avoid the vulnerabilities identified in these specifications.

|  |
| --- |
| [TOC](#toc) |

### 9.13.  Signing and Encryption Order

Signatures over encrypted text are not considered valid in many jurisdictions. Therefore, for integrity and non-repudiation, this specification requires signing the plain text JSON Claims.

|  |
| --- |
| [TOC](#toc) |

### 9.14.  Issuer Identifier

OpenID Connect supports multiple issuers per Host and Port combination. The issuer returned by discovery MUST exactly match the value of iss in the ID Token.

OpenID Connect treats the path component of any URI as part of the user identifier. For instance, the subject "1234" with an issuer of "https://example.com" is not equivalent to the subject "1234" with an issuer of "https://example.com/sales".

It is RECOMMENDED that only a single issuer per host be used.

|  |
| --- |
| [TOC](#toc) |

### 9.15.  TLS Requirements

Implementations MUST support TLS. Which version(s) ought to be implemented will vary over time, and depend on the widespread deployment and known security vulnerabilities at the time of implementation. At the time of this writing, TLS version 1.2 [[RFC5246] (**Dierks, T. and E. Rescorla, “The Transport Layer Security (TLS) Protocol Version 1.2,” August 2008.**)](#RFC5246) is the most recent version, but has very limited actual deployment, and might not be readily available in implementation toolkits. TLS version 1.0 [[RFC2246] (**Dierks, T. and C. Allen, “The TLS Protocol Version 1.0,” January 1999.**)](#RFC2246) is the most widely deployed version, and will give the broadest interoperability.

To protect against information disclosure and tampering, confidentiality protection MUST be applied using TLS with a ciphersuite that provides confidentiality and integrity protection.

Whenever TLS is used, a TLS server certificate check MUST be performed, per [RFC 6125 (**Saint-Andre, P. and J. Hodges, “Representation and Verification of Domain-Based Application Service Identity within Internet Public Key Infrastructure Using X.509 (PKIX) Certificates in the Context of Transport Layer Security (TLS),” March 2011.**)](#RFC6125) [RFC6125].

|  |
| --- |
| [TOC](#toc) |

### 9.16.  Lifetimes of Access Tokens and Refresh Tokens

Access Token grants are not revocable by the Authorization Server. Access Token grant lifetimes SHOULD be kept to single use or very short lifetimes.

If access to the UserInfo Endpoint or other protected resources is required, a Refresh Token SHOULD be used. The Client MAY then exchange the Refresh Token at the Token Endpoint for a fresh short-lived Access Token that can be used to access the resource.

The Authorization Server SHOULD clearly identify long-term grants to the User during Authorization. The Authorization Server SHOULD provide a mechanism for the End-User to revoke Refresh Tokens granted to a Client.

|  |
| --- |
| [TOC](#toc) |

### 9.17.  Symmetric Key Entropy

In [Section 3.3 (**Signing**)](#sigs) and [Section 3.4 (**Encryption**)](#enc), keys are derived from the client\_secret value. Thus, when used with symmetric signing or encryption operations, client\_secret values MUST contain sufficient entropy to generate cryptographically strong keys. Also, client\_secret values MUST also contain at least the minimum of number of octets required for MAC keys for the particular algorithm used. So for instance, for HS256, the client\_secret value MUST contain at least 8 octets (and almost certainly SHOULD contain more, since client\_secret values are likely to use a restricted alphabet.

|  |
| --- |
| [TOC](#toc) |

### 9.18.  Need for Signed Requests

In some situations, Clients might need to use signed requests to ensure that the desired request parameters are delivered to the OP without having been tampered with. For instance, the max\_age and acr\_values provide more assurance about the nature of the authentication performed when delivered in signed requests.

|  |
| --- |
| [TOC](#toc) |

### 9.19.  Need for Encrypted Requests

In some situations, knowing the contents of an OpenID Connect request can, in and of itself, reveal sensitive information about the End-User. For instance, knowing that the Client is requesting a particular Claim or that it is requesting that a particular authentication method be used can reveal sensitive information about the End-User. OpenID Connect enables requests to be encrypted to the OpenID Provider to prevent such potentially sensitive information from being revealed.

|  |
| --- |
| [TOC](#toc) |

### 10.  Privacy Considerations

The UserInfo Response typically contains Personally Identifiable Information (PII). As such, End-User consent for the release of the information for the specified purpose SHOULD be obtained at or prior to the authorization time in accordance with relevant regulations. The purpose of use is typically registered in association with the redirect\_uris.

Only necessary UserInfo data should be stored at the Client and the Client SHOULD associate the received data with the purpose of use statement.

The Resource Server SHOULD make the UserInfo access log available to the End-User so that the End-User can monitor who accessed his data.

To protect the End-User from a possible correlation among Clients, the use of a Pairwise Pseudonymous Identifier (PPID) as the sub SHOULD be considered.

|  |
| --- |
| [TOC](#toc) |

### 11.  IANA Considerations

|  |
| --- |
| [TOC](#toc) |

### 11.1.  JSON Web Token Claims Registry

This specification registers the Claims defined in [Section 2.5 (**Standard Claims**)](#StandardClaims) and [Section 2.1.2.1 (**ID Token**)](#id_token) in the IANA JSON Web Token Claims registry defined in [[JWT] (**Jones, M., Bradley, J., and N. Sakimura, “JSON Web Token (JWT),” May 2013.**)](#JWT).

|  |
| --- |
| [TOC](#toc) |

### 11.1.1.  Registry Contents

* Claim Name: name
* Change Controller: OpenID Foundation Artifact Binding Working Group - openid-specs-ab@lists.openid.net
* Specification Document(s): [**Section 2.5 (Standard Claims)**](#StandardClaims) of this document
* Claim Name: given\_name
* Change Controller: OpenID Foundation Artifact Binding Working Group - openid-specs-ab@lists.openid.net
* Specification Document(s): [**Section 2.5 (Standard Claims)**](#StandardClaims) of this document
* Claim Name: family\_name
* Change Controller: OpenID Foundation Artifact Binding Working Group - openid-specs-ab@lists.openid.net
* Specification Document(s): [**Section 2.5 (Standard Claims)**](#StandardClaims) of this document
* Claim Name: middle\_name
* Change Controller: OpenID Foundation Artifact Binding Working Group - openid-specs-ab@lists.openid.net
* Specification Document(s): [**Section 2.5 (Standard Claims)**](#StandardClaims) of this document
* Claim Name: nickname
* Change Controller: OpenID Foundation Artifact Binding Working Group - openid-specs-ab@lists.openid.net
* Specification Document(s): [**Section 2.5 (Standard Claims)**](#StandardClaims) of this document
* Claim Name: preferred\_username
* Change Controller: OpenID Foundation Artifact Binding Working Group - openid-specs-ab@lists.openid.net
* Specification Document(s): [**Section 2.5 (Standard Claims)**](#StandardClaims) of this document
* Claim Name: profile
* Change Controller: OpenID Foundation Artifact Binding Working Group - openid-specs-ab@lists.openid.net
* Specification Document(s): [**Section 2.5 (Standard Claims)**](#StandardClaims) of this document
* Claim Name: picture
* Change Controller: OpenID Foundation Artifact Binding Working Group - openid-specs-ab@lists.openid.net
* Specification Document(s): [**Section 2.5 (Standard Claims)**](#StandardClaims) of this document
* Claim Name: website
* Change Controller: OpenID Foundation Artifact Binding Working Group - openid-specs-ab@lists.openid.net
* Specification Document(s): [**Section 2.5 (Standard Claims)**](#StandardClaims) of this document
* Claim Name: email
* Change Controller: OpenID Foundation Artifact Binding Working Group - openid-specs-ab@lists.openid.net
* Specification Document(s): [**Section 2.5 (Standard Claims)**](#StandardClaims) of this document
* Claim Name: email\_verified
* Change Controller: OpenID Foundation Artifact Binding Working Group - openid-specs-ab@lists.openid.net
* Specification Document(s): [**Section 2.5 (Standard Claims)**](#StandardClaims) of this document
* Claim Name: gender
* Change Controller: OpenID Foundation Artifact Binding Working Group - openid-specs-ab@lists.openid.net
* Specification Document(s): [**Section 2.5 (Standard Claims)**](#StandardClaims) of this document
* Claim Name: birthdate
* Change Controller: OpenID Foundation Artifact Binding Working Group - openid-specs-ab@lists.openid.net
* Specification Document(s): [**Section 2.5 (Standard Claims)**](#StandardClaims) of this document
* Claim Name: zoneinfo
* Change Controller: OpenID Foundation Artifact Binding Working Group - openid-specs-ab@lists.openid.net
* Specification Document(s): [**Section 2.5 (Standard Claims)**](#StandardClaims) of this document
* Claim Name: locale
* Change Controller: OpenID Foundation Artifact Binding Working Group - openid-specs-ab@lists.openid.net
* Specification Document(s): [**Section 2.5 (Standard Claims)**](#StandardClaims) of this document
* Claim Name: phone\_number
* Change Controller: OpenID Foundation Artifact Binding Working Group - openid-specs-ab@lists.openid.net
* Specification Document(s): [**Section 2.5 (Standard Claims)**](#StandardClaims) of this document
* Claim Name: address
* Change Controller: OpenID Foundation Artifact Binding Working Group - openid-specs-ab@lists.openid.net
* Specification Document(s): [**Section 2.5 (Standard Claims)**](#StandardClaims) of this document
* Claim Name: updated\_at
* Change Controller: OpenID Foundation Artifact Binding Working Group - openid-specs-ab@lists.openid.net
* Specification Document(s): [**Section 2.5 (Standard Claims)**](#StandardClaims) of this document
* Claim Name: azp
* Change Controller: OpenID Foundation Artifact Binding Working Group - openid-specs-ab@lists.openid.net
* Specification Document(s): [**Section 2.1.2.1 (ID Token)**](#id_token) of this document
* Claim Name: nonce
* Change Controller: OpenID Foundation Artifact Binding Working Group - openid-specs-ab@lists.openid.net
* Specification Document(s): [**Section 2.1.2.1 (ID Token)**](#id_token) of this document
* Claim Name: auth\_time
* Change Controller: OpenID Foundation Artifact Binding Working Group - openid-specs-ab@lists.openid.net
* Specification Document(s): [**Section 2.1.2.1 (ID Token)**](#id_token) of this document
* Claim Name: at\_hash
* Change Controller: OpenID Foundation Artifact Binding Working Group - openid-specs-ab@lists.openid.net
* Specification Document(s): [**Section 2.1.2.1 (ID Token)**](#id_token) of this document
* Claim Name: c\_hash
* Change Controller: OpenID Foundation Artifact Binding Working Group - openid-specs-ab@lists.openid.net
* Specification Document(s): [**Section 2.1.2.1 (ID Token)**](#id_token) of this document
* Claim Name: acr
* Change Controller: OpenID Foundation Artifact Binding Working Group - openid-specs-ab@lists.openid.net
* Specification Document(s): [**Section 2.1.2.1 (ID Token)**](#id_token) of this document
* Claim Name: amr
* Change Controller: OpenID Foundation Artifact Binding Working Group - openid-specs-ab@lists.openid.net
* Specification Document(s): [**Section 2.1.2.1 (ID Token)**](#id_token) of this document
* Claim Name: sub\_jwk
* Change Controller: OpenID Foundation Artifact Binding Working Group - openid-specs-ab@lists.openid.net
* Specification Document(s): [**Section 2.1.2.1 (ID Token)**](#id_token) of this document

|  |
| --- |
| [TOC](#toc) |

### 11.2.  OAuth Parameters Registry

This specification registers the following parameters in the IANA OAuth Parameters registry defined in [RFC 6749 (**Hardt, D., “The OAuth 2.0 Authorization Framework,” October 2012.**)](#RFC6749) [RFC6749].

|  |
| --- |
| [TOC](#toc) |

### 11.2.1.  Registry Contents

* Parameter name: nonce
* Parameter usage location: Authorization Request
* Change controller: OpenID Foundation Artifact Binding Working Group - openid-specs-ab@lists.openid.net
* Specification document(s): [**Section 2.1.1.1 (Request Parameters)**](#RequestParameters) of this document
* Related information: None
* Parameter name: display
* Parameter usage location: Authorization Request
* Change controller: OpenID Foundation Artifact Binding Working Group - openid-specs-ab@lists.openid.net
* Specification document(s): [**Section 2.1.1.1 (Request Parameters)**](#RequestParameters) of this document
* Related information: None
* Parameter name: prompt
* Parameter usage location: Authorization Request
* Change controller: OpenID Foundation Artifact Binding Working Group - openid-specs-ab@lists.openid.net
* Specification document(s): [**Section 2.1.1.1 (Request Parameters)**](#RequestParameters) of this document
* Related information: None
* Parameter name: max\_age
* Parameter usage location: Authorization Request
* Change controller: OpenID Foundation Artifact Binding Working Group - openid-specs-ab@lists.openid.net
* Specification document(s): [**Section 2.1.1.1 (Request Parameters)**](#RequestParameters) of this document
* Related information: None
* Parameter name: ui\_locales
* Parameter usage location: Authorization Request
* Change controller: OpenID Foundation Artifact Binding Working Group - openid-specs-ab@lists.openid.net
* Specification document(s): [**Section 2.1.1.1 (Request Parameters)**](#RequestParameters) of this document
* Related information: None
* Parameter name: claims\_locales
* Parameter usage location: Authorization Request
* Change controller: OpenID Foundation Artifact Binding Working Group - openid-specs-ab@lists.openid.net
* Specification document(s): [**Section 2.1.1.1 (Request Parameters)**](#RequestParameters) of this document
* Related information: None
* Parameter name: id\_token\_hint
* Parameter usage location: Authorization Request
* Change controller: OpenID Foundation Artifact Binding Working Group - openid-specs-ab@lists.openid.net
* Specification document(s): [**Section 2.1.1.1 (Request Parameters)**](#RequestParameters) of this document
* Related information: None
* Parameter name: login\_hint
* Parameter usage location: Authorization Request
* Change controller: OpenID Foundation Artifact Binding Working Group - openid-specs-ab@lists.openid.net
* Specification document(s): [**Section 2.1.1.1 (Request Parameters)**](#RequestParameters) of this document
* Related information: None
* Parameter name: acr\_values
* Parameter usage location: Authorization Request
* Change controller: OpenID Foundation Artifact Binding Working Group - openid-specs-ab@lists.openid.net
* Specification document(s): [**Section 2.1.1.1 (Request Parameters)**](#RequestParameters) of this document
* Related information: None
* Parameter name: claims
* Parameter usage location: Authorization Request
* Change controller: OpenID Foundation Artifact Binding Working Group - openid-specs-ab@lists.openid.net
* Specification document(s): [**Section 2.1.1.1 (Request Parameters)**](#RequestParameters) of this document
* Related information: None
* Parameter name: registration
* Parameter usage location: Authorization Request
* Change controller: OpenID Foundation Artifact Binding Working Group - openid-specs-ab@lists.openid.net
* Specification document(s): [**Section 2.1.1.1 (Request Parameters)**](#RequestParameters) of this document
* Related information: None
* Parameter name: request
* Parameter usage location: Authorization Request
* Change controller: OpenID Foundation Artifact Binding Working Group - openid-specs-ab@lists.openid.net
* Specification document(s): [**Section 2.1.1.1 (Request Parameters)**](#RequestParameters) of this document
* Related information: None
* Parameter name: request\_uri
* Parameter usage location: Authorization Request
* Change controller: OpenID Foundation Artifact Binding Working Group - openid-specs-ab@lists.openid.net
* Specification document(s): [**Section 2.1.1.1 (Request Parameters)**](#RequestParameters) of this document
* Related information: None
* Parameter name: id\_token
* Parameter usage location: Authorization Response, Access Token Response
* Change controller: OpenID Foundation Artifact Binding Working Group - openid-specs-ab@lists.openid.net
* Specification document(s): [**Section 2.2.3 (Access Token Response)**](#access_token_response) of this document
* Related information: None

|  |
| --- |
| [TOC](#toc) |

### 11.3.  OAuth Extensions Error Registry

This specification registers the following errors in the IANA OAuth Extensions Error registry defined in [RFC 6749 (**Hardt, D., “The OAuth 2.0 Authorization Framework,” October 2012.**)](#RFC6749) [RFC6749].

|  |
| --- |
| [TOC](#toc) |

### 11.3.1.  Registry Contents

* Error name: invalid\_redirect\_uri
* Error usage location: Authorization Endpoint
* Related protocol extension: OpenID Connect
* Change controller: OpenID Foundation Artifact Binding Working Group - openid-specs-ab@lists.openid.net
* Specification document(s): [**Section 2.1.3 (Authorization Error Response)**](#AuthError) of this document
* Error name: interaction\_required
* Error usage location: Authorization Endpoint
* Related protocol extension: OpenID Connect
* Change controller: OpenID Foundation Artifact Binding Working Group - openid-specs-ab@lists.openid.net
* Specification document(s): [**Section 2.1.3 (Authorization Error Response)**](#AuthError) of this document
* Error name: login\_required
* Error usage location: Authorization Endpoint
* Related protocol extension: OpenID Connect
* Change controller: OpenID Foundation Artifact Binding Working Group - openid-specs-ab@lists.openid.net
* Specification document(s): [**Section 2.1.3 (Authorization Error Response)**](#AuthError) of this document
* Error name: session\_selection\_required
* Error usage location: Authorization Endpoint
* Related protocol extension: OpenID Connect
* Change controller: OpenID Foundation Artifact Binding Working Group - openid-specs-ab@lists.openid.net
* Specification document(s): [**Section 2.1.3 (Authorization Error Response)**](#AuthError) of this document
* Error name: consent\_required
* Error usage location: Authorization Endpoint
* Related protocol extension: OpenID Connect
* Change controller: OpenID Foundation Artifact Binding Working Group - openid-specs-ab@lists.openid.net
* Specification document(s): [**Section 2.1.3 (Authorization Error Response)**](#AuthError) of this document
* Error name: invalid\_request\_uri
* Error usage location: Authorization Endpoint
* Related protocol extension: OpenID Connect
* Change controller: OpenID Foundation Artifact Binding Working Group - openid-specs-ab@lists.openid.net
* Specification document(s): [**Section 2.1.3 (Authorization Error Response)**](#AuthError) of this document
* Error name: invalid\_request\_object
* Error usage location: Authorization Endpoint
* Related protocol extension: OpenID Connect
* Change controller: OpenID Foundation Artifact Binding Working Group - openid-specs-ab@lists.openid.net
* Specification document(s): [**Section 2.1.3 (Authorization Error Response)**](#AuthError) of this document

|  |
| --- |
| [TOC](#toc) |

### 12.  References

|  |
| --- |
| [TOC](#toc) |

### 12.1. Normative References

|  |  |
| --- | --- |
| **[E.164]** | International Telecommunication Union, “[E.164: The international public telecommunication numbering plan](http://www.itu.int/rec/T-REC-E.164-201011-I/en),” 2010. |
| **[IANA.Language]** | Internet Assigned Numbers Authority (IANA), “[Language Subtag Registry](http://www.iana.org/assignments/language-subtag-registry),” 2005. |
| **[ISO29115]** | International Organization for Standardization, “[ISO/IEC FDIS 29115 -- Information technology - Security techniques - Entity authentication assurance framework](http://www.iso.org/iso/iso_catalogue/catalogue_tc/catalogue_detail.htm?csnumber=45138),” ISO/IEC 29115, December 2012. |
| **[ISO3166-1]** | International Organization for Standardization, “[ISO 3166-1:1997. Codes for the representation of names of countries and their subdivisions -- Part 1: Country codes](http://www.w3.org/WAI/ER/IG/ert/iso639.htm),” 1997. |
| **[ISO639-1]** | International Organization for Standardization, “ISO 639-1:2002. Codes for the representation of names of languages -- Part 1: Alpha-2 code,” 2002. |
| **[ISO8601-2004]** | International Organization for Standardization, “ISO 8601:2004. Data elements and interchange formats - Information interchange - Representation of dates and times,” 2004. |
| **[JWA]** | Jones, M., “[JSON Web Algorithms (JWA)](http://tools.ietf.org/html/draft-ietf-jose-json-web-algorithms),” draft-ietf-jose-json-web-algorithms (work in progress), May 2013 ([HTML](http://tools.ietf.org/html/draft-ietf-jose-json-web-algorithms-11)). |
| **[JWE]** | Jones, M., Rescorla, E., and J. Hildebrand, “[JSON Web Encryption (JWE)](http://tools.ietf.org/html/draft-ietf-jose-json-web-encryption),” draft-ietf-jose-json-web-encryption (work in progress), May 2013 ([HTML](http://tools.ietf.org/html/draft-ietf-jose-json-web-encryption-11)). |
| **[JWK]** | Jones, M., “[JSON Web Key (JWK)](http://tools.ietf.org/html/draft-ietf-jose-json-web-key),” draft-ietf-jose-json-web-key (work in progress), May 2013 ([HTML](http://tools.ietf.org/html/draft-ietf-jose-json-web-key-11)). |
| **[JWS]** | Jones, M., Bradley, J., and N. Sakimura, “[JSON Web Signature (JWS)](http://tools.ietf.org/html/draft-ietf-jose-json-web-signature),” draft-ietf-jose-json-web-signature (work in progress), May 2013 ([HTML](http://tools.ietf.org/html/draft-ietf-jose-json-web-signature-11)). |
| **[JWT]** | Jones, M., Bradley, J., and N. Sakimura, “[JSON Web Token (JWT)](http://tools.ietf.org/html/draft-ietf-oauth-json-web-token),” draft-ietf-oauth-json-web-token (work in progress), May 2013 ([HTML](http://tools.ietf.org/html/draft-ietf-oauth-json-web-token-08)). |
| **[OAuth.Assertions]** | Campbell, B., Mortimore, C., Jones, M., and Y. Goland, “[Assertion Framework for OAuth 2.0](http://tools.ietf.org/html/draft-ietf-oauth-assertions),” draft-ietf-oauth-assertions (work in progress), March 2013 ([HTML](http://tools.ietf.org/html/draft-ietf-oauth-assertions-11)). |
| **[OAuth.JWT]** | Jones, M., Campbell, B., and C. Mortimore, “[JSON Web Token (JWT) Bearer Token Profiles for OAuth 2.0](http://tools.ietf.org/html/draft-ietf-oauth-jwt-bearer),” draft-ietf-oauth-jwt-bearer (work in progress), March 2013 ([HTML](http://tools.ietf.org/html/draft-ietf-oauth-jwt-bearer-05)). |
| **[OAuth.Responses]** | de Medeiros, B., Scurtescu, M., and P. Tarjan, “[OAuth 2.0 Multiple Response Type Encoding Practices](http://openid.net/specs/oauth-v2-multiple-response-types-1_0-07.html),” June 2013. |
| **[OpenID.Discovery]** | Sakimura, N., Bradley, J., Jones, M., and E. Jay, “[OpenID Connect Discovery 1.0](http://openid.net/specs/openid-connect-discovery-1_0-16.html),” May 2013. |
| **[OpenID.Registration]** | Sakimura, N., Bradley, J., and M. Jones, “[OpenID Connect Dynamic Client Registration 1.0](http://openid.net/specs/openid-connect-registration-1_0-18.html),” May 2013. |
| **[RFC2119]** | [Bradner, S.](mailto:sob@harvard.edu), “[Key words for use in RFCs to Indicate Requirement Levels](http://tools.ietf.org/html/rfc2119),” BCP 14, RFC 2119, March 1997 ([TXT](http://www.rfc-editor.org/rfc/rfc2119.txt), [HTML](http://xml.resource.org/public/rfc/html/rfc2119.html), [XML](http://xml.resource.org/public/rfc/xml/rfc2119.xml)). |
| **[RFC2246]** | [Dierks, T.](mailto:tdierks@certicom.com) and [C. Allen](mailto:callen@certicom.com), “[The TLS Protocol Version 1.0](http://tools.ietf.org/html/rfc2246),” RFC 2246, January 1999 ([TXT](http://www.rfc-editor.org/rfc/rfc2246.txt)). |
| **[RFC2616]** | [Fielding, R.](mailto:fielding@ics.uci.edu), [Gettys, J.](mailto:jg@w3.org), [Mogul, J.](mailto:mogul@wrl.dec.com), [Frystyk, H.](mailto:frystyk@w3.org), [Masinter, L.](mailto:masinter@parc.xerox.com), [Leach, P.](mailto:paulle@microsoft.com), and [T. Berners-Lee](mailto:timbl@w3.org), “[Hypertext Transfer Protocol -- HTTP/1.1](http://tools.ietf.org/html/rfc2616),” RFC 2616, June 1999 ([TXT](http://www.rfc-editor.org/rfc/rfc2616.txt), [PS](http://www.rfc-editor.org/rfc/rfc2616.ps), [PDF](http://www.rfc-editor.org/rfc/rfc2616.pdf), [HTML](http://xml.resource.org/public/rfc/html/rfc2616.html), [XML](http://xml.resource.org/public/rfc/xml/rfc2616.xml)). |
| **[RFC3339]** | [Klyne, G., Ed.](mailto:GK@ACM.ORG) and [C. Newman](mailto:chris.newman@sun.com), “[Date and Time on the Internet: Timestamps](http://tools.ietf.org/html/rfc3339),” RFC 3339, July 2002 ([TXT](http://www.rfc-editor.org/rfc/rfc3339.txt), [HTML](http://xml.resource.org/public/rfc/html/rfc3339.html), [XML](http://xml.resource.org/public/rfc/xml/rfc3339.xml)). |
| **[RFC3966]** | Schulzrinne, H., “[The tel URI for Telephone Numbers](http://tools.ietf.org/html/rfc3966),” RFC 3966, December 2004 ([TXT](http://www.rfc-editor.org/rfc/rfc3966.txt)). |
| **[RFC4627]** | Crockford, D., “[The application/json Media Type for JavaScript Object Notation (JSON)](http://tools.ietf.org/html/rfc4627),” RFC 4627, July 2006 ([TXT](http://www.rfc-editor.org/rfc/rfc4627.txt)). |
| **[RFC5246]** | Dierks, T. and E. Rescorla, “[The Transport Layer Security (TLS) Protocol Version 1.2](http://tools.ietf.org/html/rfc5246),” RFC 5246, August 2008 ([TXT](http://www.rfc-editor.org/rfc/rfc5246.txt)). |
| **[RFC5322]** | [Resnick, P., Ed.](mailto:presnick@qualcomm.com), “[Internet Message Format](http://tools.ietf.org/html/rfc5322),” RFC 5322, October 2008 ([TXT](http://www.rfc-editor.org/rfc/rfc5322.txt), [HTML](http://xml.resource.org/public/rfc/html/rfc5322.html), [XML](http://xml.resource.org/public/rfc/xml/rfc5322.xml)). |
| **[RFC5646]** | Phillips, A. and M. Davis, “[Tags for Identifying Languages](http://tools.ietf.org/html/rfc5646),” BCP 47, RFC 5646, September 2009 ([TXT](http://www.rfc-editor.org/rfc/rfc5646.txt)). |
| **[RFC6125]** | Saint-Andre, P. and J. Hodges, “[Representation and Verification of Domain-Based Application Service Identity within Internet Public Key Infrastructure Using X.509 (PKIX) Certificates in the Context of Transport Layer Security (TLS)](http://tools.ietf.org/html/rfc6125),” RFC 6125, March 2011 ([TXT](http://www.rfc-editor.org/rfc/rfc6125.txt)). |
| **[RFC6711]** | Johansson, L., “[An IANA Registry for Level of Assurance (LoA) Profiles](http://tools.ietf.org/html/rfc6711),” RFC 6711, August 2012 ([TXT](http://www.rfc-editor.org/rfc/rfc6711.txt)). |
| **[RFC6749]** | Hardt, D., “[The OAuth 2.0 Authorization Framework](http://tools.ietf.org/html/rfc6749),” RFC 6749, October 2012 ([TXT](http://www.rfc-editor.org/rfc/rfc6749.txt)). |
| **[RFC6750]** | Jones, M. and D. Hardt, “[The OAuth 2.0 Authorization Framework: Bearer Token Usage](http://tools.ietf.org/html/rfc6750),” RFC 6750, October 2012 ([TXT](http://www.rfc-editor.org/rfc/rfc6750.txt)). |
| **[RFC6819]** | Lodderstedt, T., McGloin, M., and P. Hunt, “[OAuth 2.0 Threat Model and Security Considerations](http://tools.ietf.org/html/rfc6819),” RFC 6819, January 2013 ([TXT](http://www.rfc-editor.org/rfc/rfc6819.txt)). |
| **[USA15]** | [Davis, M.](mailto:markdavis@google.com), [Whistler, K.](mailto:ken@unicode.org), and M. Dürst, “Unicode Normalization Forms,” Unicode Standard Annex 15, 09 2009. |
| **[zoneinfo]** | Public Domain, “[The tz database](http://www.twinsun.com/tz/tz-link.htm),” June 2011. |

|  |
| --- |
| [TOC](#toc) |

### 12.2. Informative References

|  |  |
| --- | --- |
| **[OpenID.2.0]** | OpenID Foundation, “OpenID Authentication 2.0,” December 2007 ([TXT](http://www.openid.net/specs/openid-authentication-2_0.txt), [HTML](http://www.openid.net/specs/openid-authentication-2_0.html)). |
| **[OpenID.Basic]** | Sakimura, N., Bradley, J., Jones, M., de Medeiros, B., and C. Mortimore, “[OpenID Connect Basic Client Profile 1.0](http://openid.net/specs/openid-connect-basic-1_0-27.html),” May 2013. |
| **[OpenID.Implicit]** | Sakimura, N., Bradley, J., Jones, M., de Medeiros, B., Mortimore, C., and E. Jay, “[OpenID Connect Implicit Client Profile 1.0](http://openid.net/specs/openid-connect-implicit-1_0-10.html),” May 2013. |
| **[OpenID.PAPE]** | [Recordon, D.](mailto:david@sixapart.com), [Jones, M.](mailto:mbj@microsoft.com), [Bufu, J., Ed.](mailto:johnny.bufu@gmail.com), [Daugherty, J., Ed.](mailto:cygnus@janrain.com), and [N. Sakimura](mailto:n-sakimura@nri.co.jp), “OpenID Provider Authentication Policy Extension 1.0,” December 2008 ([TXT](http://openid.net/specs/openid-provider-authentication-policy-extension-1_0.txt), [HTML](http://openid.net/specs/openid-provider-authentication-policy-extension-1_0.html)). |
| **[OpenID.Session]** | Sakimura, N., Bradley, J., Jones, M., de Medeiros, B., Mortimore, C., and E. Jay, “[OpenID Connect Session Management 1.0](http://openid.net/specs/openid-connect-session-1_0-14.html),” May 2013. |
| **[OpenID.Standard]** | Sakimura, N., Bradley, J., Jones, M., de Medeiros, B., Mortimore, C., and E. Jay, “[OpenID Connect Standard 1.0](http://openid.net/specs/openid-connect-standard-1_0-20.html),” May 2013. |
| **[RFC4949]** | Shirey, R., “[Internet Security Glossary, Version 2](http://tools.ietf.org/html/rfc4949),” RFC 4949, August 2007 ([TXT](http://www.rfc-editor.org/rfc/rfc4949.txt)). |
| **[X.1252]** | International Telecommunication Union, “[ITU-T Recommendation X.1252 -- Cyberspace security -- Identity management -- Baseline identity management terms and definitions](http://www.itu.int/rec/dologin_pub.asp?lang=e&id=T-REC-X.1252-201004-I!!PDF-E&type=items),” ITU-T X.1252, November 2010. |

|  |
| --- |
| [TOC](#toc) |

### Appendix A.  Acknowledgements

As a successor version of OpenID, this specification heavily relies on ideas explored in [OpenID Authentication 2.0 (**OpenID Foundation, “OpenID Authentication 2.0,” December 2007.**)](#OpenID.2.0) [OpenID.2.0]. Please refer to Appendix C of OpenID Authentication 2.0 for the full list of the contributors for that specification.

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

Naveen Agarwal (naa@google.com), Google

Amanda Anganes (aanganes@mitre.org), Mitre

Casper Biering (cb@peercraft.com), Peercraft

John Bradley (ve7jtb@ve7jtb.com), Ping Identity

Johnny Bufu (jbufu@janrain.com), Janrain

Tim Bray (tbray@textuality.com), Google

Brian Campbell (bcampbell@pingidentity.com), Ping Identity

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

Pamela Dingle (pdingle@pingidentity.com), Ping Identity

Vladimir Dzhuvinov (vladimir@nimbusds.com), NimbusDS

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

Roland Hedberg (roland.hedberg@adm.umu.se), Independent

Ryo Ito (ryo.ito@mixi.co.jp), mixi, Inc.

Edmund Jay (ejay@mgi1.com), Illumila

Michael B. Jones (mbj@microsoft.com), Microsoft

Torsten Lodderstedt (t.lodderstedt@telekom.de), Deutsche Telekom

Nov Matake (nov@matake.jp), Independent

Chuck Mortimore (cmortimore@salesforce.com), Salesforce

Anthony Nadalin (tonynad@microsoft.com), Microsoft

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

Axel Nennker (axel.nennker@telekom.de), Deutsche Telekom

David Recordon (dr@fb.com), Facebook

Justin Richer (jricher@mitre.org), Mitre

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

Luke Shepard (lshepard@fb.com), Facebook

Andreas Akre Solberg (andreas.solberg@uninett.no), UNINET

Paul Tarjan (pt@fb.com), Facebook

|  |
| --- |
| [TOC](#toc) |

### Appendix B.  Notices

Copyright (c) 2013 The OpenID Foundation.

The OpenID Foundation (OIDF) grants to any Contributor, developer, implementer, or other interested party a non-exclusive, royalty free, worldwide copyright license to reproduce, prepare derivative works from, distribute, perform and display, this Implementers Draft or Final Specification solely for the purposes of (i) developing specifications, and (ii) implementing Implementers Drafts and Final Specifications based on such documents, provided that attribution be made to the OIDF as the source of the material, but that such attribution does not indicate an endorsement by the OIDF.

The technology described in this specification was made available from contributions from various sources, including members of the OpenID Foundation and others. Although the OpenID Foundation has taken steps to help ensure that the technology is available for distribution, it takes no position regarding the validity or scope of any intellectual property or other rights that might be claimed to pertain to the implementation or use of the technology described in this specification or the extent to which any license under such rights might or might not be available; neither does it represent that it has made any independent effort to identify any such rights. The OpenID Foundation and the contributors to this specification make no (and hereby expressly disclaim any) warranties (express, implied, or otherwise), including implied warranties of merchantability, non-infringement, fitness for a particular purpose, or title, related to this specification, and the entire risk as to implementing this specification is assumed by the implementer. The OpenID Intellectual Property Rights policy requires contributors to offer a patent promise not to assert certain patent claims against other contributors and against implementers. The OpenID Foundation invites any interested party to bring to its attention any copyrights, patents, patent applications, or other proprietary rights that may cover technology that may be required to practice this specification.

|  |
| --- |
| [TOC](#toc) |

### Appendix C.  Document History

[[ To be removed from the final specification ]]

-19

* Stated that OPs should perform no other nonce processing than returning nonce values received in requests as claims in issued ID Tokens.
* Stated that sufficient entropy must be present in nonce values to prevent attackers from guessing values.
* Required that the aud value in an ID Token issued from a Refresh Token must match that in the originally issued ID Token.
* Stated that the Authorization Server need not be listed as an audience of the ID Token when it is used as an id\_token\_hint value.
* Restricted the meaning of the azp (authorized party) Claim to simply be the single party to which the ID Token was issued.
* Changed the requirement to support the UserInfo endpoint from applying to all OpenID Providers to only applying to dynamic OpenID Providers.
* Changed from using the term "byte" to either "octet" or "character".
* Fixed #835 - Clarified requirements on using value and values qualifiers when requesting specific values for individual claims.
* Removed definition of the x5c JWK key member since it is now defined in the JWK specification.
* Stated that the JWS Compact Serialization and the JWE Compact Serialization are always used for JWS and JWE data structures.
* Fixed #847 - Corrected type of updated\_at to number.
* Fixed #840 - Provided a better description of the Token Substitution attack.
* Fixed #844 - Provided a better Authentication definition.
* Fixed #845 - Added a reference to ITU-T X.1252.
* Said that we're using the terms "validate" and "verify" as defined by RFC 4949.

-18

* Replaced the PKIX JWK key type with the x5c JWK key member. (This will be removed from this specification once this member is defined in the JWK specification.)
* Fixed #820 - Removed assumption that Clients that want encrypted responses also sign requests.
* Fixed #821 - Moved definition of JSON Serialization to where it's used.
* Fixed #825 - Replaced updated\_time, which used the RFC 3339 textual time format, with updated\_at, using the numeric time format used by iat, etc.
* Fixed #826 - Clarified response\_type values for which an offline\_access request must be ignored.
* Fixed #827 - Added request\_uri\_not\_supported error code.
* Fixed #828 - Stated that an extension would be needed if a key wrapping key of greater than 256 bits for symmetric encryption needs to be derived.
* Fixed #829 - Stated that additional scope values can be defined and used and that scope values that are not understood should be ignored.
* Fixed #831 - Stated that JWS and JWE header parameters used to communicate key values and key references should not be used in ID Tokens, since these are communicated in advance using Discovery and Registration parameters.
* Fixed #712 and #830 - Clarified the azp description and made azp multi-valued, like aud.

-17

* Fixed #802 - Clarified recommendations and responsibilities for producing and consuming Claims with and without language tags.
* Fixed #797 - Clarified the intended semantics of e-mail verification and that the precise verification rules are context-specific.
* Fixed #806 - Added phone\_number\_verified Claim.
* Fixed #800 - Specified that phone number extensions are to be represented using RFC 3966 extension syntax.
* Fixed #795 - Specified that e-mail addresses must conform to the RFC 5322 addr-spec syntax.
* Fixed #808 - Specified that phone numbers may be used as login\_hint values.
* Fixed #801 - Removed schema and id parameters to UserInfo Endpoint. Also fixed related issue #791 - Removed invalid\_schema error.
* Fixed #793, #796, and #799 - Allow name Claims to contain multiple space-separated names.
* Fixed #794 - Required picture to refer to an image file that is a picture of the End-User.
* Fixed #805 - Placed requirements on use of the sub Claim when Aggregated Claims and Distributed Claims are used, to prevent unintended correlations.
* Fixed #811 - Specify that language tag components should be spelled using the character cases registered in the IANA Language Subtag Registry.
* Fixed #812 - Clarified that language tag values used need not be unnecessarily specific.
* Fixed #816 - Changed "must understand" language to "MUST be ignored if not understood".
* Added section on using Additional Claims.

-16

* Fixed #709 - OpenID Request Object - "registration" in non-Self-Issued case.
* Fixed #711 - Awkward phrase "The following Claims are REQUIRED and OPTIONAL".
* Fixed #712 - "azp" definition clarification.
* Fixed #713 - Explicitly require "sub" claim to be returned from UserInfo endpoint.
* Fixed #715 - Delete "profile" from request object example.
* Fixed #729 - ITU-T X.1254 | ISO/IEC 29115 now separate.
* Fixed #722 - Text on "id\_token\_hint" needs to be clarified.
* Fixed #718 - Text on re-encrypting should be clearer.
* Fixed #738 - Behavior when "openid" scope is omitted.
* Fixed #714 - Clarified text specifying response\_type behaviors, including prohibiting the use of the "token" response\_type, since it returns no ID Token.
* Added statement "whenever TLS is used, a TLS server certificate check MUST be performed, per [**RFC 6125 (Saint-Andre, P. and J. Hodges, “Representation and Verification of Domain-Based Application Service Identity within Internet Public Key Infrastructure Using X.509 (PKIX) Certificates in the Context of Transport Layer Security (TLS),” March 2011.)**](#RFC6125) [RFC6125]" to TLS Requirements section in Security Considerations.
* State that when any validations fail, any operations requiring the information that failed to correctly validate MUST be aborted and the information that failed to validate MUST NOT be used.
* Fixed #742 - Added new ui\_locales parameter.
* Fixed #743 - Promoted preferred\_locales to being a top-level parameter. Also renamed it to claims\_locales to disambiguate it from the new ui\_locales parameter.
* Fixed #744 - Promoted max\_age to being a top-level parameter.
* Fixed #748 - Promoted claims to being a top-level parameter separate from the OpenID Request Object.
* Fixed #765 - Created acr\_values top-level request parameter and changed default\_acr registration parameter to default\_acr\_values.
* Fixed #761 - client\_secret as the HMAC key and #762 - client\_secret to key. We now use the phrase "the bytes of the UTF-8 representation of the client\_secret value". Also added security considerations about symmetric key entropy.
* Fixed #597 - Changed representation of omitted year in birthdate from 9999 to 0000.
* Fixed #769 - Added Claim Type identifiers and definition.
* Fixed #773 - Added request\_uris registration parameter to pre-register request\_uri values. Also clarified that the referenced resource contents may be cached.
* Fixed #748 - Changed OpenID Request Object processing rules so that the Request Object parameters are combined with those passed as OAuth 2.0 parameters, with the Request Object parameters taking precedence. This enables fixed parameters to be passed in pre-signed, possibly pre-encrypted, and cached Request Objects, with parameters that will vary per request like state and nonce being passed as OAuth 2.0 parameters. This is particularly important now that request\_uri values can be pre-registered.
* Fixed #765 - Added Security Considerations about the need for signed and encrypted requests.
* Fixed #763 - OPs MUST treat the inability to return an Essential requested acr Claim Value as a failed authentication attempt.
* Fixed #739 - Added values for Self-Issued registration.
* Fixed #779 - Parameters missing from IANA Considerations.
* Fixed #760 - Added rationale for request\_uri usage.
* Fixed #782 - Changed uses of "\_url" in identifiers to "\_uri".
* Fixed #703 - Added the PKIX JWK key type (and example) for X.509 certificates and consolidated the x509\_uri, x509\_encryption\_uri, and jwk\_encryption\_uri parameters into a combined jwk\_uri parameter. Also Fixed #704 - Provided suggested guidance about how to do key rotation of asymmetric keys for both signing and encryption using jwk\_uri.
* Made UserInfo Endpoint MTI to support for all OPs that issue Access Tokens.
* Fixed #719 - Moved message definitions for Self-Issued OPs to the Messages spec.
* Fixed #671 - Specified that an Access Token must be requested when Claims are requested from the UserInfo endpoint.
* Fixed #717 - Parameters and values should be distinguished more clearly.
* Fixed #786 - Changed the name of jwk\_uri to jwks\_uri.
* Clarified when the http scheme can and can not be used in redirect\_uri values.
* Fixed #748 - Defined MTI features for OPs. Also added request\_not\_supported error code.
* Fixed #784 - Required publication of public keys as bare keys.
* Fixed #785 - Enabled scope values to be used to request Claims when using the response\_type value id\_token (for which no Access Token is issued).
* Fixed #787 - Don't prohibit returning an ID Token from the Token Endpoint when grant types other than authorization\_code are used.
* Fixed #710 - Gave an example of how requesting Claims with scope values is equivalent to requesting them with the claims request parameter.
* Stated that the azp Claim is only needed when the party requesting the ID Token is different than the audience of the ID Token.
* Fixed #788 - Renamed "OpenID Request Object" to "Request Object".
* Use legal acr values in examples.
* Fixed #789 - Added amr (authentication methods references) Claim.
* Fixed #790 - Removed "MUST understand" text about request parameters, since OAuth requires that unrecognized parameters MUST be ignored.
* Added requirements for ID Tokens returned as a result of a token refresh request.

-15

* Fixed #671 - Section 2.1.1 added test to require client to request an access token for the UserInfo Endpoint if requesting the default scopes.
* Fixed #637 - Removed requirement for hash of at\_token and code to be SHA2 in Section 2.1.2.1 and Section 5.2.
* Fixes #620 - Update Section 2.1.2 and Section 2.2.3 to allow for other token types, but make bearer mandatory to support for clients.
* Fixes #684 Removed error response in redirect to client if the redirect\_uri is wrong to align with OAuth.
* Fixed #695 - Contradictory OPTIONAL MUSTs in JWT Client Authentication.
* Fixed #600 - Register Connect Claims in JWT Claims Registry.
* Made the OpenID Foundation Artifact Binding Working Group the change controller for the values registered with IANA.
* Moved OAuth error registrations from Standard to Messages since the errors are defined in Messages and not in Standard.
* Use "OpenID Connect" as the "Related protocol extension" value in OAuth Extensions Error registry entries.
* Fixed #657 - Section 2.1.1 Specify the sub is used as the kid if the request object or id\_token\_hint is encrypted.
* Added Implementation Considerations section on Mandatory to Implement (MTI) features, per issue #604.
* Specified that dynamic OPs must publish their public keys in X.509 format, per issue #633.
* Fixed #648 - Specific response types now used in place of "implicit flow" and "code flow". The wording for at\_hash and c\_hash is now clearer.
* Fixed #698 - Inconsistent use of articles.
* Fixed #699 - OpenID Provider (OP) definition repetitive.
* Fixed #701 - Mention of SWD without a reference.
* Fixed #700 - Incomplete specification names.
* Fixed #702 - Make scopes a reference to the scopes section.
* Fixed #702 - Consent is for claims - not scopes.
* Fixed up Scopes section to make it clear that claims requested by the scopes are voluntary.
* OAuth Threat Model is now RFC 6819.
* Renamed the user\_jwk Claim to sub\_jwk, paralleling the change from user\_id to sub.
* Defined and registered the sub\_jwk claim.

-14

* Fixed #687 - Inconsistency between user\_id and prn claims. The fix changed these names: user\_id -> sub, user\_id\_types\_supported -> subject\_types\_supported, user\_id\_type -> subject\_type, and prn -> sub.
* Fixed #689 - Track JWT change that allows JWTs to have multiple audiences.
* Fixed #660 - Clarified that returning the sub value from the UserInfo endpoint is mandatory.
* Fixed #636 - ID Token authorized party claim.
* Fixed #539 - Add scope for offline access.
* Fixed #690 - Inconsistent language in requirement of id\_token response\_type.
* Clarified that jwk\_uri and jwk\_encryption\_uri refer to documents containing JWK Sets - not single JWK keys.
* Fixed #689 - Add caution about multiple audiences and azp.
* Fixed #692 typos.

-13

* Fixed #588 - Messages: Token lifetime not privacy consideration
* Fixed #597 - Messages: Changed claim name birthday to birthdate and made the format ISO 8601:2004
* Fixed #606 - Messages - 2.1.1. ID Token - acr missing the type. Type String.
* Fixed #617 - Messages - 1.2 Terminology: id\_token. Added sentence that it can contain other claims.
* Fixed #608 - Messages - Request ID Token and Response ID Token. Moved response ID Token from section 2.1 to the response section.
* Fixed #543 - Messages - Security Consideration. Added ref to X.1254 | ISO 29115.
* Fixed #611 - Messages - Changed the default test to indicate that the value of auth\_time needs to be essential
* Fixed #646 - Messages - Add login\_hint as an OAuth parameter and put in example in request object
* Fixed #607 - Messages - add example decoded id\_token.
* Fixed #658 - Messages - 2.1.1 "id\_token" name crash, id\_token renamed to id\_token\_hint
* Fixed #612 - Messages - 4.1 change request\_object\_algs\_supported to be RS256
* Fixed #662 - Messages - changed id\_token\_signed\_response\_algs to id\_token\_signed\_response\_alg in 5.1
* Fixed #678 - Messages - Changed 5.1.3 terminology of acr to reflect essential vs. required and fixed example
* Fixed #679 - Messages - update reference to LoA registry from ID to RFC6711.
* Fixed #614 - Discovery - 3.2 Distinguishing between signature and integrity parameters for HMAC algorithms. This fix tracks the parameter changes made to the JWE spec in draft-ietf-jose-json-web-encryption-06. It deletes the parameters {userinfo,id\_token}\_encrypted\_response\_int. It replaces the parameters {userinfo,id\_token,request\_object,token\_endpoint}\_algs\_supported with {userinfo,id\_token,request\_object,token\_endpoint}\_signing\_alg\_values\_supported and {userinfo,id\_token,request\_object,token\_endpoint}\_encryption\_{alg,enc}\_values\_supported.
* Fixed #673 - Registration 2.1: Rename require\_signed\_request\_object to request\_object\_alg. The actual change was to rename require\_signed\_request\_object to request\_object\_signing\_alg, following the naming convention used in the resolution to issue #614.
* Fixed #666 - JWS signature validation vs. verification.
* Referenced OAuth 2.0 RFCs -- RFC 6749 and RFC 6750.
* Fixed #663 Sec 5.2 to allow for non SHA2 HMAC algs

-12

* Added preferred\_username claim under profile scope
* Added section on claim stability
* Changed request\_uri to request\_uri in Section 2.1.2.1

-11

* Removed claims\_in\_id\_token scope value, per decision on June 15, 2012 special working group call

-10

* Changed verified to email\_verified, per issue #564
* Added scope value claims\_in\_id\_token as a switch to indicate that the UserInfo claims should be returned in the ID Token, per issue #561
* Removed optional claim request parameter and added essential claim request parameter, per issue #577. We changed terminology from "optional" to "voluntary" and "required" to "essential" to better match privacy policy requirements.
* Removed Check ID Endpoint, per issue #570
* Added PAPE Reference to the Informative References, per issue #574
* Added "id\_token" response type as being MTI for OpenID Providers
* Specified that parameters present in both the OpenID Request Object and the OAuth 2.0 Authorization Request MUST exactly match, per issue #575
* Changed OpenID Request Object from being specified as a JWT to being specified as a JWS signed base64url encoded JSON object, per issue #592
* Changed default ID Token signing algorithm to RS256, per issue #571
* Changed default OpenID Request Object signing algorithm to RS256, per issue #571
* Made use of the nonce REQUIRED when using the implicit flow and OPTIONAL when using the code flow, per issue #569
* Changed client.example.com to client.example.org, per issue #251
* Listed author of ISO29115 as "International Telecommunication Union and International Organization for Standardization", per issue #589
* Added method of calculating signing and encryption keys for symmetric algorithms, per issue #578
* Use standards track versions of JSON Web Token (JWT) and OAuth JWT Bearer Token Profiles specs (draft-ietf-oauth-json-web-token and draft-ietf-oauth-oauth-jwt-bearer)

-09

* Added error interaction\_required and removed user\_mismatched, per issue #523
* Changed invalid\_request\_redirect\_uri to invalid\_redirect\_uri, per issue #553
* Removed "embedded" display type, since its semantics were not well defined, per issue #514
* Added optional id\_token to authorization request parameters, per issue #535
* Now requested claims add to those requested with scope values, rather than replacing them, per issue #547
* Make changes to allow path in the issuer\_identifier, per issue #513
* Make changes to userinfo\_encrypted\_response\_\* and id\_token\_encrypted\_response\_\* to match registration
* Add hash and hash check of access\_token and code to id\_token, per issue #510
* Updated Notices
* Updated References

-08

* Updated the version number and date
* Fixed #551 Sec 2.1.2.1 to clarify the OpenID Request Object MUST NOT include "request" nor "request\_uri"
* Fixed #540 Sec 2.2.3 id\_token MUST NOT be returned for grant\_type=refresh
* Fixed #542 Sec 2.1.2.1 required fields for request object to match Standard
* Fixed Sec 2.2.1 to refer to client\_secret value rather than Client Password
* Fixed Sec 4.2, 4.3, 4.4 to replace requirement for using x.509 keyuse extension
* Added reference to RFC2459
* Fixed Sec 2.1.1.1.1 added rationale for sector\_identifier\_url from registration
* Fixed Sec 2.1.1.1.1 added examples of other ways to generate PPID
* Added iat as a required claim in ID Tokens
* Enumerated claims requested by the "profile" scope value
* Fixed Sec 2.1.2 response\_type references standard rather than repeating values that are binding specific
* Fixed Sec 2.1.2 remove outdated language about openid scope requiring id\_token to be returned with token response\_type

-07

* Removed definition and usage for assertion and claim object
* Consistent use of End-User
* Removed 'format' from userinfo and id\_token object of the OpenID Request Object
* email scope allows access to the 'verified' claim
* ID Token 'audience' claim MUST be client\_id
* Rename artifact to authorization code
* Removed language pertaining to custom userinfo schemas
* Check ID Endpoint returns only JSON
* Updated Check ID Response verification
* Remove 'audience' parameter from Authorization Request
* Moved display=none to prompt=none
* Added additional display parameter options
* Moved IANA considerations to Standard
* Added error codes to Authorization Endpoint
* Added client authentication section regarding various supported client authentication schemes and their validation. This includes symmetric and asymmetric authentication, JWT Bearer Token Profiles, OAuth 2.0 Assertion Profile
* Updated Check ID Response verification
* Added 'auth\_time' to ID Token
* Added validation for request object encryption and signature
* Added explanation for user\_id type and calculating pairwise identifiers
* Added steps for signature and validation and encryption and decryption
* Added verification of issuer identifier
* Redefined 'nonce' in Authorization Request. Changed to REQUIRED parameter.
* Changed usage of the word "approval" to "consent"
* Use RFC 6125 to verify TLS endpoints
* ID Token MUST be JWT
* Access Tokens should include an audience claim for the Resource Server
* Updated Security Considerations
* OpenID Request Object parameters takes precedence over the same parameters in the Authorization Request
* Allow other gender strings in UserInfo schema
* Changed UserInfo claim 'locale' to 'preferred\_locales' and changed it to be a list of values
* Changed UserInfo claim 'user\_id' to REQUIRED. Added requirement to compare user\_id from userinfo endpoint to id\_token
* RECOMMENDED E.164 format for UserInfo 'phone\_number' claim
* Changed UserInfo Error Response to augment and return OAuth 2.0 Bearer Token Error Response
* Expanded section regarding UserInfo 'address' claim
* Added Privacy considerations
* Added rational for signing then encrypting added to security considerations
* Added section about string comparison rules needed
* The Authorization Server MUST understand all the request parameters except for any unsupported claims.
* Make openid scope provide user\_id from userinfo endpoint
* Added explanation of select\_account
* Check ID Endpoint uses ID Token as Access Token according to Bearer Token spec
* Clients MUST verify client\_id in ID Token
* Bumped version + date
* Update John Bradley email and affiliation for Implementer's Draft
* Removed invalid\_authorization\_code, invalid\_id\_token error codes
* Section 2.3 client MUST NOT send encrypted JWT to the Check ID Endpoint
* Section 2.1.2.1.2 Added user\_id claim and moved iso29115 to claims element of id\_token member
* Defined Authentication Context, Authentication Context Class Reference (acr), replaced iso29115 with acr.
* Corrected instances of x509\_url\_encryption to x509\_encryption\_url and jwk\_url\_encryption to jwk\_encryption\_url

-06

* Changed section 3.1.4.1 to say the errors are returned as defined by the response type not always as query parameters. per ticket #174.
* Bumped version + date.
* Fixed section 3.3.3 to refer to errors in Bearer Token.
* Fixed 3.1.3 to ref the other response types ticket #173.
* Included reference to multiple response types.
* Fixed 3.1.2.1 to indicate default Claims in id\_token.
* Fixed section 3.2.2 to reference the access token response from the token endpoint 4.1.4.
* Fixed section 3.2.1 to include refresh tokens.
* Fixed section 3.1.1 to be clear on JWT being the token format per ticket #171.

-05

* Changed check\_session to check\_id.
* schema=openid now required when requesting UserInfo.
* Removed issued\_to, since not well defined.
* Removed display values popup, touch, and mobile, since not well defined.

-04

* Changes associated with renaming "Lite" to "Basic Client" and replacing "Core" and "Framework" with "Messages" and "Standard".
* Numerous cleanups, including updating references.

-03

* Added secret\_type to the Token endpoint.
* Minor edits to the samples.

-02

* Incorporates feedback from Nat Sakimura.

-01

* First Draft that incorporates the merge of the Core and Framework specs.

|  |
| --- |
| [TOC](#toc) |

### Authors' Addresses

|  |  |
| --- | --- |
|  | Nat Sakimura |
|  | Nomura Research Institute, Ltd. |
| **Email:** | [n-sakimura@nri.co.jp](mailto:n-sakimura@nri.co.jp) |
|  |  |
|  | John Bradley |
|  | Ping Identity |
| **Email:** | [ve7jtb@ve7jtb.com](mailto:ve7jtb@ve7jtb.com) |
|  |  |
|  | Michael B. Jones |
|  | Microsoft |
| **Email:** | [mbj@microsoft.com](mailto:mbj@microsoft.com) |
|  |  |
|  | Breno de Medeiros |
|  | Google Inc. |
| **Email:** | [breno@google.com](mailto:breno@google.com) |
|  |  |
|  | Chuck Mortimore |
|  | Salesforce |
| **Email:** | [cmortimore@salesforce.com](mailto:cmortimore@salesforce.com) |
|  |  |
|  | Edmund Jay |
|  | Illumila |
| **Email:** | [ejay@mgi1.com](mailto:ejay@mgi1.com) |