International Standard

ISO/IEC 26132

Information technology — OpenID connect — OpenID connect discovery 1.0 incorporating errata set 2

First edition
2024-10

Reference number
ISO/IEC 26132:2024(en)

Foreword

ISO (the International Organization for Standardization) and IEC (the International Electrotechnical Commission) form the specialized system for worldwide standardization. National bodies that are members of ISO or IEC participate in the development of International Standards through technical committees established by the respective organization to deal with particular fields of technical activity. ISO and IEC technical committees collaborate in fields of mutual interest. Other international organizations, governmental and non-governmental, in liaison with ISO and IEC, also take part in the work.

The procedures used to develop this document and those intended for its further maintenance are described in the ISO/IEC Directives, Part 1. In particular, the different approval criteria needed for the different types of document should be noted (see www.iso.org/directives or www.iec.ch/members_experts/refdocs).

ISO and IEC draw attention to the possibility that the implementation of this document may involve the use of (a) patent(s). ISO and IEC take no position concerning the evidence, validity or applicability of any claimed patent rights in respect thereof. As of the date of publication of this document, ISO and IEC had not received notice of (a) patent(s) which may be required to implement this document. However, implementers are cautioned that this may not represent the latest information, which may be obtained from the patent database available at www.iso.org/patents and https://patents.iec.ch. ISO and IEC shall not be held responsible for identifying any or all such patent rights.

Any trade name used in this document is information given for the convenience of users and does not constitute an endorsement.

For an explanation of the voluntary nature of standards, the meaning of ISO specific terms and expressions related to conformity assessment, as well as information about ISO's adherence to the World Trade Organization (WTO) principles in the Technical Barriers to Trade (TBT), see www.iso.org/iso/foreword.html. In the IEC, see www.iec.ch/understanding-standards.

This document was prepared by the OpenID Foundation (OIDF) (as OpenID Connect Discovery 1.0 incorporating errata set 2) and drafted in accordance with its editorial rules. It was adopted, under the JTC 1 PAS procedure, by Joint Technical Committee ISO/IEC JTC 1, *Information technology*.

Any feedback or questions on this document should be directed to the user's national standards body. A complete listing of these bodies can be found at www.iso.org/members.html and www.iec.ch/national-committees.

Abstract

OpenID Connect 1.0 is a simple identity layer on top of the OAuth 2.0 protocol. It enables 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 defines a mechanism for an OpenID Connect Relying Party to discover the End-User's OpenID Provider and obtain information needed to interact with it, including its OAuth 2.0 endpoint locations.

Table of Contents

Information technology — OpenID Connect — OpenID Connect Discovery 1.0 incorporating errata set 2

1. Introduction

TOC

OpenID Connect 1.0 is a simple identity layer on top of the OAuth 2.0 [RFC6749] protocol. It enables 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.

In order for an OpenID Connect Relying Party to utilize OpenID Connect services for an End-User, the RP needs to know where the OpenID Provider is. OpenID Connect uses WebFinger [RFC7033] to locate the OpenID Provider for an End-User. This process is described in Section 2.

Once the OpenID Provider has been identified, the configuration information for that OP is retrieved from a well-known location as a JSON [RFC8259] document, including its OAuth 2.0 endpoint locations. This process is described in Section 4.

The previous versions of this specification are:

- OpenID Connect Discovery 1.0 incorporating errata set 1 [OpenID.Discovery.Errata1]
- OpenID Connect Discovery 1.0 (final) [OpenID.Discovery.Final]

1.1. Requirements Notation and Conventions

TOC

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119 [RFC2119].

In the .txt version of this specification, 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. In the HTML version of this specification, values to be taken literally are indicated by the use of `this fixed-width font`.

All uses of JSON Web Signature (JWS) [JWS] and JSON Web Encryption (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.

1.2. Terminology

This specification uses the terms "Authorization Code", "Authorization Endpoint", "Authorization Server", "Client", "Client Authentication", "Client Secret", "Grant Type", "Response Type", and "Token Endpoint" defined by OAuth 2.0 [RFC6749], the terms "Claim Name", "Claim Value", and "JSON Web Token (JWT)" defined by JSON Web Token (JWT) [JWT], and the terms defined by OpenID Connect Core 1.0 [OpenID.Core] and OAuth 2.0 Multiple Response Type Encoding Practices [OAuth.Responses].

This specification also defines the following terms:

Resource

> Entity that is the target of a request in WebFinger.

Host

> Server where a WebFinger service is hosted.

Identifier

> Value that uniquely characterizes an Entity in a specific context.
>
> NOTE: This specification defines various kinds of Identifiers, designed for use in different contexts. Examples include URLs using the `https` scheme and e-mail addresses.

IMPORTANT NOTE TO READERS: The terminology definitions in this section are a normative portion of this specification, imposing requirements upon implementations. All the capitalized words in the text of this specification, such as "Identifier", reference these defined terms. Whenever the reader encounters them, their definitions found in this section must be followed.

2. OpenID Provider Issuer Discovery

OpenID Provider Issuer discovery is the process of determining the location of the OpenID Provider.

Issuer discovery is OPTIONAL; if a Relying Party knows the OP's Issuer location through an out-of-band mechanism, it can skip this step and proceed to Section 4.

The following information is needed to perform issuer discovery using WebFinger [RFC7033]:

resource

Identifier for the target End-User that is the subject of the discovery request.

host

Server where a WebFinger service is hosted.

rel

URI identifying the type of service whose location is being requested.

OpenID Connect uses the following discoverable `rel` value in WebFinger [RFC7033]:

Rel Type	**URI**
OpenID Connect Issuer	http://openid.net/specs/connect/1.0/issuer

To start discovery of OpenID endpoints, the End-User supplies an Identifier to the Relying Party. Any Web input form MUST employ Cross-Site Request Forgery (CSRF) prevention [OWASP.CSRF].

The RP applies normalization rules to the Identifier to determine the Resource and Host. Then it makes an HTTP `GET` request to the Host's WebFinger [RFC7033] endpoint with the `resource` parameter to obtain the location of the requested service. Use of the `rel` parameter in the request with a value of `http://openid.net/specs/connect/1.0/issuer` is also RECOMMENDED to narrow the response to the specific link relation type needed.

All WebFinger communication MUST utilize TLS in the manner described in Section 7.1. The WebFinger endpoint SHOULD support the use of Cross-Origin Resource Sharing (CORS) [CORS] and/or other methods as appropriate to enable JavaScript Clients and other Browser-Based Clients to access it.

The Issuer location MUST be returned in the WebFinger response as the value of the `href` member of a `links` array element with `rel` member value `http://openid.net/specs/connect/1.0/issuer`. (Per Section 7 of WebFinger [RFC7033], obtaining the WebFinger response may first involve following some redirects.)

The returned Issuer location MUST be a URI RFC 3986 [RFC3986] with a scheme component that MUST be `https`, a host component, and optionally, port and path components and no query or fragment components. Note that since the Host and Resource values determined from the user input Identifier, as described in Section 2.1, are used as input to a WebFinger request, which can return an Issuer value using a completely different scheme, host, port, and path, no relationship can be assumed between the user input Identifier string and the resulting Issuer location.

2.1. Identifier Normalization

TOC

The purpose of Identifier normalization is to determine normalized Resource and Host values from the user input Identifier. These are then used as WebFinger request parameters to discover the Issuer location.

The user input Identifier SHOULD be a URL or URI relative reference defined in RFC 3986 [RFC3986]. The user input Identifier MUST include the authority component.

NOTE: A URI relative reference includes a string that looks like an e-mail address in the form of `userinfo@host`. This is a valid authority component of a URI but excludes various possible extra strings allowed in `addr-spec` syntax of RFC 5322 [RFC5322].

The Identifier normalization rules MAY be extended by additional specifications to enable other identifier types such as telephone numbers or XRIs [XRI_Syntax_2.0] to also be used.

2.1.1. User Input Identifier Types

TOC

A user input Identifier can be categorized into the following types, which require different normalization processes:

1. User input Identifiers starting with the XRI [XRI_Syntax_2.0] global context symbols ('=','@', and '!') are RESERVED. Processing of these identifiers is out of scope for this specification.

2. All other user input Identifiers MUST be treated as a URI in one of the forms `scheme "://" authority path-abempty [ "?" query ] [ "#" fragment ]` or `authority path-abempty [ "?" query ] [ "#" fragment ]` or `scheme ":" path-rootless`, per RFC 3986 [RFC3986].

NOTE: The user input Identifier MAY be in the form of `userinfo@host`. For the End-User, this would normally be perceived as being an e-mail address. However, it is also a valid userpart "@" host section of an `acct` URI [RFC7565], and this specification treats it such as to exclude various extra strings allowed in `addr-spec` of RFC 5322 [RFC5322].

2.1.2. Normalization Steps

TOC

A string of any other type is interpreted as a URI in one of the forms `scheme "://" authority path-abempty [ "?" query ] [ "#" fragment ]` or `authority path-abempty [ "?" query ] [ "#" fragment ]` or `scheme ":" path-rootless` per RFC 3986 [RFC3986] and is normalized according to the following rules:

1. If the user input Identifier does not have an RFC 3986 [RFC3986] scheme component, the string is interpreted as `[userinfo "@"] host [":" port] path-abempty [ "?" query ] [ "#" fragment ]` per RFC 3986 [RFC3986]. Examples are `example.com`, `joe@example.com`, `example.com/joe`, and `example.com:8080`.

2. If the userinfo and host components are present and all of the scheme, path, query, port, and fragment components are absent, the `acct` scheme is assumed. In this case, the normalized URI is formed by prefixing `acct:` to the string as the scheme. Per The 'acct' URI Scheme [RFC7565], if there is an at-sign character ('@') in the userinfo component, it needs to be percent-encoded, as described in RFC 3986 [RFC3986]. Examples are `joe@example.com` and `Jane.Doe@example.com`.

3. For all other inputs without a scheme component, the `https` scheme is assumed, and the normalized URI is formed by prefixing `https://` to the string as the scheme. Examples are `example.com`, `example.com/joe`, `example.com:8080`, and `joe@example.com:8080`.

4. When the input contains an explicit scheme such as `acct` or `https` that matches the RFC 3986 `scheme ":" path-rootless` syntax, no input normalization is performed. Examples are `https://example.com`, `https://example.com/joe`, `https://joe@example.com:8080`, and `acct:joe@example.com`.

5. If the resulting URI contains a fragment component, it MUST be stripped off, together with the fragment delimiter character "#".

The WebFinger [RFC7033] Resource in this case is the resulting URI, and the WebFinger Host is the authority component.

NOTE: Since the definition of `authority` in RFC 3986 [RFC3986] is `[ userinfo "@" ] host [ ":" port ]`, it is legal to have a user input identifier like `userinfo@host:port`, e.g., `alice@example.com:8080`.

2.2. Non-Normative Examples

TOC

2.2.1. User Input using E-Mail Address Syntax

TOC

To find the Issuer for the given user input in the form of an e-mail address `joe@example.com`, the WebFinger parameters are as follows:

WebFinger Parameter	Value
resource	acct:joe@example.com
host	example.com
rel	http://openid.net/specs/connect/1.0/issuer

Note that in this case, the `acct:` scheme [RFC7565] is prepended to the Identifier.

The RP would make the following WebFinger request to discover the Issuer location (with line wraps within lines for display purposes only):

```
  GET /.well-known/webfinger
    ?resource=acct%3Ajoe%40example.com

&rel=http%3A%2F%2Fopenid.net%2Fspecs%2Fconnect%2F1.0%2Fi
ssuer
    HTTP/1.1
  Host: example.com

  HTTP/1.1 200 OK
  Content-Type: application/jrd+json

  {
   "subject": "acct:joe@example.com",
   "links":
    [
     {
      "rel":
"http://openid.net/specs/connect/1.0/issuer",
      "href": "https://server.example.com"
      }
    ]
  }
```

2.2.2. User Input using URL Syntax

TOC

To find the Issuer for the given URL, `https://example.com/joe`, the WebFinger parameters are as follows:

WebFinger Parameter	Value
resource	https://example.com/joe
host	example.com
rel	http://openid.net/specs/connect/1.0/issuer

The RP would make the following WebFinger request to discover the Issuer location (with line wraps within lines for display purposes only):

```
  GET /.well-known/webfinger
    ?resource=https%3A%2F%2Fexample.com%2Fjoe

&rel=http%3A%2F%2Fopenid.net%2Fspecs%2Fconnect%2F1.0%2Fi
ssuer
    HTTP/1.1
  Host: example.com

  HTTP/1.1 200 OK
  Content-Type: application/jrd+json

  {
   "subject": "https://example.com/joe",
   "links":
    [
     {
      "rel":
"http://openid.net/specs/connect/1.0/issuer",
      "href": "https://server.example.com"
     }
    ]
  }
```

2.2.3. User Input using Hostname and Port Syntax

TOC

If the user input is in the form of `host:port`, e.g., example.com:8080, then it is assumed as the authority component of the URL.

To find the Issuer for the given hostname, `example.com:8080`, the WebFinger parameters are as follows:

WebFinger Parameter	Value
resource	https://example.com:8080/
host	example.com:8080
rel	http://openid.net/specs/connect/1.0/issuer

The RP would make the following WebFinger request to discover the Issuer location (with line wraps within lines for display purposes only):

```
  GET /.well-known/webfinger
    ?resource=https%3A%2F%2Fexample.com%3A8080%2F

&rel=http%3A%2F%2Fopenid.net%2Fspecs%2Fconnect%2F1.0%2Fi
ssuer
    HTTP/1.1
  Host: example.com:8080

  HTTP/1.1 200 OK
  Content-Type: application/jrd+json

  {
   "subject": "https://example.com:8080/",
   "links":
    [
     {
      "rel":
"http://openid.net/specs/connect/1.0/issuer",
      "href": "https://server.example.com"
     }
    ]
  }
```

2.2.4. User Input using "acct" URI Syntax

TOC

To find the Issuer for the given user input in the form of an account URI `acct:juliet%40capulet.example@shopping.example.com`, the WebFinger parameters are as follows:

WebFinger Parameter	Value
resource	acct:juliet%40capulet.example@shopping.example.com
host	shopping.example.com
rel	http://openid.net/specs/connect/1.0/issuer

The RP would make the following WebFinger request to discover the Issuer location (with line wraps within lines for display purposes only):

```
  GET /.well-known/webfinger

?resource=acct%3Ajuliet%2540capulet.example%40shopping.e
xample.com

&rel=http%3A%2F%2Fopenid.net%2Fspecs%2Fconnect%2F1.0%2Fi
ssuer
    HTTP/1.1
  Host: shopping.example.com

  HTTP/1.1 200 OK
  Content-Type: application/jrd+json

  {
   "subject":
"acct:juliet%40capulet.example@shopping.example.com",
   "links":
    [
     {
      "rel":
"http://openid.net/specs/connect/1.0/issuer",
      "href": "https://server.example.com"
     }
    ]
  }
```

NOTE: It is common for sites to use e-mail addresses as local identifiers for accounts at those sites, even though the domain in the e-mail address is not one controlled by the site. For instance, the site `example.org` might have a local account named `joe@example.com`. This specification uses `acct:` URIs as defined by [RFC7565] to represent such accounts during WebFinger discovery. Such an example is `acct:joe%40example.com@example.org`. End-Users MAY input values like `joe@example.com@example.org` to initiate discovery on such accounts.

3. OpenID Provider Metadata

TOC

OpenID Providers have metadata describing their configuration. These OpenID Provider Metadata values are used by OpenID Connect:

issuer

REQUIRED. URL using the `https` scheme with no query or fragment components that the OP asserts as its Issuer Identifier. If Issuer discovery is supported (see Section 2), this value MUST be identical to the issuer value returned by WebFinger. This also MUST be identical to the `iss` Claim value in ID Tokens issued from this Issuer.

authorization_endpoint

REQUIRED. URL of the OP's OAuth 2.0 Authorization Endpoint [OpenID.Core]. This URL MUST use the `https` scheme and MAY contain port, path, and query parameter components.

token_endpoint

URL of the OP's OAuth 2.0 Token Endpoint [OpenID.Core]. This is REQUIRED unless only the Implicit Flow is used. This URL MUST use the `https` scheme and MAY contain port, path, and query parameter components.

userinfo_endpoint

RECOMMENDED. URL of the OP's UserInfo Endpoint [OpenID.Core]. This URL MUST use the `https` scheme and MAY contain port, path, and query parameter components.

jwks_uri

REQUIRED. URL of the OP's JWK Set [JWK] document, which MUST use the `https` scheme. This contains the signing key(s) the RP uses to validate signatures from the OP. The JWK Set MAY also contain the Server's encryption key(s), which are used by RPs to encrypt requests to the Server. When both signing and encryption keys are made available, a `use` (public key use) parameter value is REQUIRED for all keys in the referenced JWK Set to indicate each key's intended usage. Although some algorithms allow the same key to be used for both signatures and encryption, doing so is NOT RECOMMENDED, as it is less secure. 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. The JWK Set MUST NOT contain private or symmetric key values.

registration_endpoint

RECOMMENDED. URL of the OP's Dynamic Client Registration Endpoint [OpenID.Registration], which MUST use the `https` scheme.

scopes_supported

RECOMMENDED. JSON array containing a list of the OAuth 2.0 [RFC6749] scope values that this server supports. The server MUST support the `openid` scope value. Servers MAY choose not to advertise some supported scope values even when this parameter is used, although those defined in [OpenID.Core] SHOULD be listed, if supported.

response_types_supported

REQUIRED. JSON array containing a list of the OAuth 2.0 `response_type` values that this OP supports. Dynamic OpenID Providers MUST support the `code`, `id_token`, and the `id_token token` Response Type values.

response_modes_supported

OPTIONAL. JSON array containing a list of the OAuth 2.0 `response_mode` values that this OP supports, as specified in OAuth 2.0 Multiple Response Type Encoding Practices [OAuth.Responses]. If omitted, the default for Dynamic OpenID Providers is `["query", "fragment"]`.

grant_types_supported

OPTIONAL. JSON array containing a list of the OAuth 2.0 Grant Type values that this OP supports. Dynamic OpenID Providers MUST support the `authorization_code` and `implicit` Grant Type values and MAY support other Grant Types. If omitted, the default value is `["authorization_code", "implicit"]`.

acr_values_supported

OPTIONAL. JSON array containing a list of the Authentication Context Class References that this OP supports.

subject_types_supported

REQUIRED. JSON array containing a list of the Subject Identifier types that this OP supports. Valid types include `pairwise` and `public`.

id_token_signing_alg_values_supported

REQUIRED. JSON array containing a list of the JWS signing algorithms (`alg` values) supported by the OP for the ID Token to encode the Claims in a JWT [JWT]. The algorithm `RS256` MUST be included. The value `none` MAY be supported but MUST NOT be used unless the Response Type used returns no ID Token from the Authorization Endpoint (such as when using the Authorization Code Flow).

id_token_encryption_alg_values_supported

OPTIONAL. JSON array containing a list of the JWE encryption algorithms (`alg` values) supported by the OP for the ID Token to encode the Claims in a JWT [JWT].

id_token_encryption_enc_values_supported

OPTIONAL. JSON array containing a list of the JWE encryption algorithms (`enc` values) supported by the OP for the ID Token to encode the Claims in a JWT [JWT].

userinfo_signing_alg_values_supported

OPTIONAL. JSON array containing a list of the JWS [JWS] signing algorithms (`alg` values) [JWA] supported by the UserInfo Endpoint to encode the Claims in a JWT [JWT]. The value `none` MAY be included.

userinfo_encryption_alg_values_supported

OPTIONAL. JSON array containing a list of the JWE [JWE] encryption algorithms (`alg` values) [JWA] supported by the UserInfo Endpoint to encode the Claims in a JWT [JWT].

userinfo_encryption_enc_values_supported

OPTIONAL. JSON array containing a list of the JWE encryption algorithms (`enc` values) [JWA] supported by the UserInfo Endpoint to encode the Claims in a JWT [JWT].

request_object_signing_alg_values_supported

OPTIONAL. JSON array containing a list of the JWS signing algorithms (`alg` values) supported by the OP for Request Objects, which are described in Section 6.1 of OpenID Connect Core 1.0 [OpenID.Core]. These algorithms are used both when the Request Object is passed by value (using the `request` parameter) and when it is passed by reference (using the `request_uri` parameter). Servers SHOULD support `none` and `RS256`.

request_object_encryption_alg_values_supported

OPTIONAL. JSON array containing a list of the JWE encryption algorithms (`alg` values) supported by the OP for Request Objects. These algorithms are used both when the Request Object is passed by value and when it is passed by reference.

request_object_encryption_enc_values_supported

OPTIONAL. JSON array containing a list of the JWE encryption algorithms (`enc` values) supported by the OP for Request Objects. These algorithms are used both when the Request Object is passed by value and when it is passed by reference.

token_endpoint_auth_methods_supported

OPTIONAL. JSON array containing a list of Client Authentication methods supported by this Token Endpoint. The options are `client_secret_post`, `client_secret_basic`, `client_secret_jwt`, and `private_key_jwt`, as described in Section 9 of OpenID Connect Core 1.0 [OpenID.Core]. Other authentication methods MAY be defined by extensions. If omitted, the default is `client_secret_basic` -- the HTTP Basic Authentication Scheme specified in Section 2.3.1 of OAuth 2.0 [RFC6749].

token_endpoint_auth_signing_alg_values_supported

> OPTIONAL. JSON array containing a list of the JWS signing algorithms (`alg` values) supported by the Token Endpoint for the signature on the JWT [JWT] used to authenticate the Client at the Token Endpoint for the `private_key_jwt` and `client_secret_jwt` authentication methods. Servers SHOULD support `RS256`. The value `none` MUST NOT be used.

display_values_supported

> OPTIONAL. JSON array containing a list of the `display` parameter values that the OpenID Provider supports. These values are described in Section 3.1.2.1 of OpenID Connect Core 1.0 [OpenID.Core].

claim_types_supported

> OPTIONAL. JSON array containing a list of the Claim Types that the OpenID Provider supports. These Claim Types are described in Section 5.6 of OpenID Connect Core 1.0 [OpenID.Core]. Values defined by this specification are `normal`, `aggregated`, and `distributed`. If omitted, the implementation supports only `normal` Claims.

claims_supported

> RECOMMENDED. JSON array containing a list of the Claim Names of the Claims that the OpenID Provider MAY be able to supply values for. Note that for privacy or other reasons, this might not be an exhaustive list.

service_documentation

> OPTIONAL. URL of a page containing human-readable information that developers might want or need to know when using the OpenID Provider. In particular, if the OpenID Provider does not support Dynamic Client Registration, then information on how to register Clients needs to be provided in this documentation.

claims_locales_supported

> OPTIONAL. Languages and scripts supported for values in Claims being returned, represented as a JSON array of BCP47 [RFC5646] language tag values. Not all languages and scripts are necessarily supported for all Claim values.

ui_locales_supported

OPTIONAL. Languages and scripts supported for the user interface, represented as a JSON array of BCP47 [RFC5646] language tag values.

claims_parameter_supported

OPTIONAL. Boolean value specifying whether the OP supports use of the `claims` parameter, with `true` indicating support. If omitted, the default value is `false`.

request_parameter_supported

OPTIONAL. Boolean value specifying whether the OP supports use of the `request` parameter, with `true` indicating support. If omitted, the default value is `false`.

request_uri_parameter_supported

OPTIONAL. Boolean value specifying whether the OP supports use of the `request_uri` parameter, with `true` indicating support. If omitted, the default value is `true`.

require_request_uri_registration

OPTIONAL. Boolean value specifying whether the OP requires any `request_uri` values used to be pre-registered using the `request_uris` registration parameter. Pre-registration is REQUIRED when the value is `true`. If omitted, the default value is `false`.

op_policy_uri

OPTIONAL. URL that the OpenID Provider provides to the person registering the Client to read about the OP's requirements on how the Relying Party can use the data provided by the OP. The registration process SHOULD display this URL to the person registering the Client if it is given.

op_tos_uri

OPTIONAL. URL that the OpenID Provider provides to the person registering the Client to read about the OpenID Provider's terms of service. The registration process SHOULD display this URL to the person registering the Client if it is given.

The Token Endpoint, UserInfo Endpoint, `jwks_uri` endpoint, Dynamic Client Registration Endpoint, and any other endpoints directly accessed by Clients SHOULD support the use of Cross-Origin Resource Sharing (CORS) [CORS] and/or other methods as appropriate to enable JavaScript Clients and other Browser-Based Clients to access them. The use of CORS at the Authorization Endpoint is NOT RECOMMENDED as it is redirected to by the client and not directly accessed.

Additional OpenID Provider Metadata parameters MAY also be used. Some are defined by other specifications, such as OpenID Connect Session Management 1.0 [OpenID.Session].

4. Obtaining OpenID Provider Configuration Information

TOC

Using the Issuer location discovered as described in Section 2 or by other means, the OpenID Provider's configuration information can be retrieved.

OpenID Providers supporting Discovery MUST make a JSON document available at the path formed by concatenating the string `/.well-known/openid-configuration` to the Issuer. The syntax and semantics of `.well-known` are defined in RFC 5785 [RFC5785] and apply to the Issuer value when it contains no path component. `openid-configuration` MUST point to a JSON document compliant with this specification and MUST be returned using the `application/json` content type. The `openid-configuration` endpoint SHOULD support the use of Cross-Origin Resource Sharing (CORS) [CORS] and/or other methods as appropriate to enable JavaScript Clients and other Browser-Based Clients to access it.

4.1. OpenID Provider Configuration Request

TOC

An OpenID Provider's configuration information MUST be retrieved using an HTTP `GET` request at the previously specified path.

The RP would make the following request to the Issuer `https://example.com` to obtain its Configuration information, since the Issuer contains no path component:

```
GET /.well-known/openid-configuration HTTP/1.1
Host: example.com
```

If the Issuer value contains a path component, any terminating / MUST be removed before appending /.well-known/openid-configuration. The RP would make the following request to the Issuer https://example.com/issuer1 to obtain its configuration information, since the Issuer contains a path component:

```
GET /issuer1/.well-known/openid-configuration HTTP/1.1
Host: example.com
```

Using path components enables supporting multiple issuers per host. This is required in some multi-tenant hosting configurations. This use of .well-known is for supporting multiple issuers per host; unlike its use in RFC 5785 [RFC5785], it does not provide general information about the host.

4.2. OpenID Provider Configuration Response

TOC

The response is a set of Claims about the OpenID Provider's configuration, including all necessary endpoints and public key location information. A successful response MUST use the 200 OK HTTP status code and return a JSON object using the application/json content type that contains a set of Claims as its members that are a subset of the Metadata values defined in Section 3. Other Claims MAY also be returned.

Claims that return multiple values are represented as JSON arrays. Claims with zero elements MUST be omitted from the response.

An error response uses the applicable HTTP status code value.

The following is a non-normative example response:

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

{
 "issuer":
   "https://server.example.com",
 "authorization_endpoint":
   "https://server.example.com/connect/authorize",
 "token_endpoint":
   "https://server.example.com/connect/token",
   "token_endpoint_auth_methods_supported":
     ["client_secret_basic", "private_key_jwt"],
   "token_endpoint_auth_signing_alg_values_supported":
     ["RS256", "ES256"],
   "userinfo_endpoint":
     "https://server.example.com/connect/userinfo",
   "check_session_iframe":
     "https://server.example.com/connect/check_session",
   "end_session_endpoint":
     "https://server.example.com/connect/end_session",
   "jwks_uri":
     "https://server.example.com/jwks.json",
   "registration_endpoint":
     "https://server.example.com/connect/register",
   "scopes_supported":
     ["openid", "profile", "email", "address",
      "phone", "offline_access"],
   "response_types_supported":
     ["code", "code id_token", "id_token", "id_token
token"],
   "acr_values_supported":
     ["urn:mace:incommon:iap:silver",
      "urn:mace:incommon:iap:bronze"],
   "subject_types_supported":
     ["public", "pairwise"],
   "userinfo_signing_alg_values_supported":
     ["RS256", "ES256", "HS256"],
   "userinfo_encryption_alg_values_supported":
     ["RSA-OAEP-256", "A128KW"],
   "userinfo_encryption_enc_values_supported":
     ["A128CBC-HS256", "A128GCM"],
   "id_token_signing_alg_values_supported":
     ["RS256", "ES256", "HS256"],
   "id_token_encryption_alg_values_supported":
     ["RSA-OAEP-256", "A128KW"],
   "id_token_encryption_enc_values_supported":
     ["A128CBC-HS256", "A128GCM"],
   "request_object_signing_alg_values_supported":
     ["none", "RS256", "ES256"],
   "display_values_supported":
     ["page", "popup"],
   "claim_types_supported":
     ["normal", "distributed"],
   "claims_supported":
     ["sub", "iss", "auth_time", "acr",
      "name", "given_name", "family_name", "nickname",
      "profile", "picture", "website",
      "email", "email_verified", "locale", "zoneinfo",
      "http://example.info/claims/groups"],
   "claims_parameter_supported":
     true,
   "service_documentation":
     
"http://server.example.com/connect/service_documentation
.html",
   "ui_locales_supported":
     ["en-US", "en-GB", "en-CA", "fr-FR", "fr-CA"]
  }
```

4.3. OpenID Provider Configuration Validation

TOC

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.

The `issuer` value returned MUST be identical to the Issuer URL that was used as the prefix to `/.well-known/openid-configuration` to retrieve the configuration information. This MUST also be identical to the `iss` Claim value in ID Tokens issued from this Issuer.

5. String Operations

TOC

Processing some OpenID Connect messages requires comparing values in the messages to known values. For example, the member names in the provider configuration response might be compared to specific member names such as `issuer`. Comparing Unicode [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 [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.

6. Implementation Considerations

TOC

This specification defines features used by both Relying Parties and OpenID Providers that choose to implement Discovery. All of these Relying Parties and OpenID Providers MUST implement the features that are listed in this specification as being "REQUIRED" or are described with a "MUST". No other implementation considerations for implementations of Discovery are defined by this specification.

6.1. Compatibility Notes

TOC

NOTE: Potential compatibility issues that were previously described in the original version of this specification have since been addressed.

7. Security Considerations

TOC

7.1. TLS Requirements

TOC

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. Implementations SHOULD follow the guidance in BCP 195 [RFC8996] [RFC9325], which provides recommendations and requirements for improving the security of deployed services that use TLS.

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 [RFC6125].

7.2. Impersonation Attacks

TOC

TLS certificate checking MUST be performed by the RP, as described in Section 7.1, when making an OpenID Provider Configuration Request. Checking that the server certificate is valid for the Issuer URL prevents man-in-middle and DNS-based attacks. These attacks could cause an RP to be tricked into using an attacker's keys and endpoints, which would enable impersonation of the legitimate Issuer. If an attacker can accomplish this, they can access the accounts of any existing users at the affected RP that can be logged into using the OP that they are impersonating.

An attacker may also attempt to impersonate an OpenID Provider by publishing a Discovery document that contains an `issuer` Claim using the Issuer URL of the OP being impersonated, but with its own endpoints and signing keys. This would enable it to issue ID Tokens as that OP, if accepted by the RP. To prevent this, RPs MUST ensure that the Issuer URL they are using for the Configuration Request exactly matches the value of the `issuer` Claim in the OP Metadata document received by the RP and that this also exactly matches the `iss` Claim value in ID Tokens that are supposed to be from that Issuer.

8. IANA Considerations

TOC

8.1. Well-Known URI Registry

TOC

This specification registers the well-known URI defined in Section 4 in the IANA "Well-Known URIs" registry [IANA.well-known] established by RFC 5785 [RFC5785].