Final specs@openid.net
December 5, 2007

OpenID Authentication 2.0 - Final

Abstract

OpenID Authentication provides a way to prove that an end user controls an Identifier. It does this without the Relying Party needing access to end user credentials such as a password or to other sensitive information such as an email address.

OpenID is decentralized. No central authority must approve or register Relying Parties or OpenID Providers. An end user can freely choose which OpenID Provider to use, and can preserve their Identifier if they switch OpenID Providers.

While nothing in the protocol requires JavaScript or modern browsers, the authentication scheme plays nicely with "AJAX"-style setups. This means an end user can prove their Identity to a Relying Party without having to leave their current Web page.

OpenID Authentication uses only standard HTTP(S) requests and responses, so it does not require any special capabilities of the User-Agent or other client software. OpenID is not tied to the use of cookies or any other specific mechanism of Relying Party or OpenID Provider session management. Extensions to User-Agents can simplify the end user interaction, though are not required to utilize the protocol.

The exchange of profile information, or the exchange of other information not covered in this specification, can be addressed through additional service types built on top of this protocol to create a framework. OpenID Authentication is designed to provide a base service to enable portable, user-centric digital identity in a free and decentralized manner.

Table of Contents

---

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 [RFC2119] (Bradner, B., “Key words for use in RFCs to Indicate Requirement Levels,” .).

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

---

2. Terminology

Identifier:
: An Identifier is either a "http" or "https" URI, (commonly referred to as a "URL" within this document), or an XRI (Reed, D. and D. McAlpin, “Extensible Resource Identifier (XRI) Syntax V2.0,” .) [XRI_Syntax_2.0]. This document defines various kinds of Identifiers, designed for use in different contexts.

User-Agent:
: The end user's Web browser which implements HTTP/1.1 [RFC2616] (Fielding, R., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P., and T. Berners-Lee, “Hypertext Transfer Protocol -- HTTP/1.1,” .).

Relying Party:
: RP. A Web application that wants proof that the end user controls an Identifier.

OpenID Provider:
: OP. An OpenID Authentication server on which a Relying Party relies for an assertion that the end user controls an Identifier.

OP Endpoint URL:
: The URL which accepts OpenID Authentication protocol messages, obtained by performing discovery on the User-Supplied Identifier. This value MUST be an absolute HTTP or HTTPS URL.

OP Identifier:
: An Identifier for an OpenID Provider.

User-Supplied Identifier:
: An Identifier that was presented by the end user to the Relying Party, or selected by the user at the OpenID Provider. During the initiation phase of the protocol, an end user may enter either their own Identifier or an OP Identifier. If an OP Identifier is used, the OP may then assist the end user in selecting an Identifier to share with the Relying Party.

Claimed Identifier:
: An Identifier that the end user claims to own; the overall aim of the protocol is verifying this claim. The Claimed Identifier is either:

    - The Identifier obtained by normalizing (Normalization) the User-Supplied Identifier, if it was an URL.
    - The CanonicalID (XRI and the CanonicalID Element), if it was an XRI.

OP-Local Identifier:
: An alternate Identifier for an end user that is local to a particular OP and thus not necessarily under the end user's control.

3. Protocol Overview

1. The end user initiates authentication (Initiation) by presenting a User-Supplied Identifier to the Relying Party via their User-Agent.
2. After normalizing (Normalization) the User-Supplied Identifier, the Relying Party performs discovery (Discovery) on it and establishes the OP Endpoint URL that the end user uses for authentication. It should be noted that the User-Supplied Identifier may be an OP Identifier, as discussed in Section 7.3.1 (Discovered Information), which allows selection of a Claimed Identifier at the OP or for the protocol to proceed without a Claimed Identifier if something else useful is being done via an extension (Extensions).
3. (optional) The Relying Party and the OP establish an association (Establishing Associations) -- a shared secret established using Diffie-Hellman Key Exchange (Rescorla, E., “Diffie-Hellman Key Agreement Method,” .) [RFC2631]. The OP uses an association to sign subsequent messages and the Relying Party to verify those messages; this removes the need for subsequent direct requests to verify the signature after each authentication request/response.
4. The Relying Party redirects the end user's User-Agent to the OP with an OpenID Authentication request (Requesting Authentication).
5. The OP establishes whether the end user is authorized to perform OpenID Authentication and wishes to do so. The manner in which the end user authenticates to their OP and any policies surrounding such authentication is out of scope for this document.
6. The OP redirects the end user's User-Agent back to the Relying Party with either an assertion that authentication is approved (Positive Assertions) or a message that authentication failed (Negative Assertions).
7. The Relying Party verifies (Verifying Assertions) the information received from the OP including checking the Return URL, verifying the discovered information, checking the nonce, and verifying the signature by using either the shared key established during the association or by sending a direct request to the OP.

4. Data Formats

4.1. Protocol Messages

The OpenID Authentication protocol messages are mappings of plain-text keys to plain-text values. The keys and values permit the full Unicode character set (UCS). When the keys and values need to be converted to/from bytes, they MUST be encoded using UTF-8 (Yergeau, F., “UTF-8, a transformation format of Unicode and ISO 10646,” .) [RFC3629].

Messages MUST NOT contain multiple parameters with the same name.

Throughout this document, all OpenID message parameters are REQUIRED, unless specifically marked as OPTIONAL.

4.1.1. Key-Value Form Encoding

A message in Key-Value form is a sequence of lines. Each line begins with a key, followed by a colon, and the value associated with the key. The line is terminated by a single newline (UCS codepoint 10, "\n"). A key or value MUST NOT contain a newline and a key also MUST NOT contain a colon.

Additional characters, including whitespace, MUST NOT be added before or after the colon or newline. The message MUST be encoded in UTF-8 to produce a byte string.

Key-Value Form encoding is used for signature calculation and for direct responses (Direct Response) to Relying Parties.

4.1.2. HTTP Encoding

When a message is sent to an HTTP server, it MUST be encoded using a form encoding specified in Section 17.13.4 of [HTML401] (W3C, “HTML 4.01 Specification,” .). Likewise, if the "Content-Type" header is included in the request headers, its value MUST also be such an encoding.

All of the keys in the request message MUST be prefixed with "openid.". This prefix prevents interference with other parameters that are passed along with the OpenID

Authentication message. When a message is sent as a POST, OpenID parameters MUST only be sent in, and extracted from, the POST body.

All messages that are sent as HTTP requests (GET or POST) MUST contain the following fields:

- openid.ns

  Value: "http://specs.openid.net/auth/2.0"

  This particular value MUST be present for the request to be a valid OpenID Authentication 2.0 request. Future versions of the specification may define different values in order to allow message recipients to properly interpret the request.

  If this value is absent or set to one of "http://openid.net/signon/1.1" or "http://openid.net/signon/1.0", then this message SHOULD be interpreted using OpenID Authentication 1.1 Compatibility mode (OpenID Authentication 1.1 Compatibility).

- openid.mode

  Value: Specified individually for each message type.

  The "openid.mode" parameter allows the recipient of the message to know what kind of message it is processing. If "openid.mode" is absent, the party processing the message SHOULD assume that the request is not an OpenID message.

This model applies to messages from the User-Agent to both the Relying Party and the OP, as well as messages from the Relying Party to the OP.

4.1.3. Example

Non-normative

The following examples encode the following information:

```
Key     | Value
--------+--------------------------
mode    | error
error   | This is an example message
```

Key-Value Form encoded:

```
mode:error
error:This is an example message
```

x-www-urlencoded, as in a HTTP POST body or in a URL's query string ([RFC3986] (Berners-Lee, T., “Uniform Resource Identifiers (URI): Generic Syntax,” .) section 3):

```
openid.mode=error&openid.error=This%20is%20an%20example%20message
```

4.2. Integer Representations

Arbitrary precision integers MUST be encoded as big-endian signed two's complement binary strings. Henceforth, "btwoc" is a function that takes an arbitrary precision integer and returns its shortest big-endian two's complement representation. All integers that are used with Diffie-Hellman Key Exchange are positive. This means that the left-most bit of the two's complement representation MUST be zero. If it is not, implementations MUST add a zero byte at the front of the string.

Non-normative example:

```
Base 10 number | btwoc string representation
---------------+----------------------------
0              | "\x00"
127            | "\x7F"
128            | "\x00\x80"
255            | "\x00\xFF"
32768          | "\x00\x80\x00"
```

---

5. Communication Types

5.1. Direct Communication

Direct communication is initiated by a Relying Party to an OP endpoint URL. It is used for establishing associations (Establishing Associations) and verifying authentication assertions (Verifying Directly with the OpenID Provider).

5.1.1. Direct Request

The message MUST be encoded as a POST body, as specified by Section 4.1.2 (HTTP Encoding).

All direct requests are HTTP POSTs, and so contain the required fields listed in Section 4.1.2 (HTTP Encoding).

5.1.2. Direct Response

The body of a response to a Direct Request (Direct Request) consists of an HTTP Response body in Key-Value Form (Key-Value Form Encoding). The content-type of the response SHOULD be "text/plain".

All Key-Value form message MUST contain the following field:

- ns

  Value: "http://specs.openid.net/auth/2.0"

  This particular value MUST be present for the response to be a valid OpenID 2.0 response. Future versions of the specification may define different values in order to allow message recipients to properly interpret the request.

  If this value is absent or set to one of "http://openid.net/signon/1.1" or "http://openid.net/signon/1.0", then this message SHOULD be interpreted using OpenID Authentication 1.1 Compatibility mode (OpenID Authentication 1.1 Compatibility).

5.1.2.1. Successful Responses

A server receiving a valid request MUST send a response with an HTTP status code of 200.

5.1.2.2. Error Responses

If a request is malformed or contains invalid arguments, the server MUST send a response with a status code of 400. The response body MUST be a Key-Value Form (Key-Value Form Encoding) message with the following fields:

- ns

  As specified in Section 5.1.2 (Direct Response).

- error

  Value: A human-readable message indicating the cause of the error.

- contact

  Value: (optional) Contact address for the administrator of the sever. The contact address may take any form, as it is intended to be displayed to a person.

- reference

  Value: (optional) A reference token, such as a support ticket number or a URL to a news blog, etc.

The OP MAY add additional fields to this response.

5.2. Indirect Communication

In indirect communication, messages are passed through the User-Agent. This can be initiated by either the Relying Party or the OP. Indirect communication is used for authentication requests (Requesting Authentication) and authentication responses (Responding to Authentication Requests).

There are two methods for indirect communication: HTTP redirects and HTML form submission. Both form submission and redirection require that the sender know a recipient URL and that the recipient URL expect indirect messages, as specified in Section 4.1.2 (HTTP Encoding). The initiator of the communication chooses which method of indirect communication is appropriate depending on capabilities, message size, or other external factors.

All indirect messages arrive as HTTP requests, and so contain the required fields listed in Section 4.1.2 (HTTP Encoding).

5.2.1. HTTP Redirect

Data can be transferred by issuing a 302, 303, or 307 HTTP Redirect to the end user's User-Agent. The redirect URL is the URL of the receiver with the OpenID Authentication message appended to the query string, as specified in Section 4.1.2 (HTTP Encoding).

5.2.2. HTML FORM Redirection

A mapping of keys to values can be transferred by returning an HTML page to the User-Agent that contains an HTML form element. Form submission MAY be automated, for example by using JavaScript.

The <form> element's "action" attribute value MUST be the URL of the receiver. Each Key-Value pair MUST be included in the form as an <input> element. The key MUST be encoded as the "name" attribute and the value as the "value" attribute, such that the User-Agent will generate a message as specified in Section 4.1.2 (HTTP Encoding) when the form is submitted. The form MUST include a submit button.

5.2.3. Indirect Error Responses

In the case of a malformed request, or one that contains invalid arguments, the OpenID Provider MUST redirect the User-Agent to the "openid.return_to" URL value if the value is present and it is a valid URL.

- openid.ns

  As specified in Section 4.1.2 (HTTP Encoding).

- openid.mode

  Value: "error"

- openid.error

  Value: A human-readable message indicating the cause of the error.

- openid.contact

  Value: (optional) Contact address for the administrator of the sever. The contact address may take any form, as it is intended to be displayed to a person.

- openid.reference

Value: (optional) A reference token, such as a support ticket number or a URL to a news blog, etc.

The server MAY add additional keys to this response.

If the malformed or invalid message is received by the Relying Party, or "openid.return_to" is not present or its value is not a valid URL, the server SHOULD return a response to the end user indicating the error and that it is unable to continue.

---

6. Generating Signatures

The most common usage of an association is as a Message Authentication Code (MAC) key used to sign OpenID Authentication messages.

When generating MAC keys, the recommendations in [RFC1750] (Eastlake, D., Crocker, S., and J. Schiller, “Randomness Recommendations for Security,” .) SHOULD be followed.

6.1. Procedure

To generate a message signature:

1. Determine the list of keys to be signed, according to the message to be signed (See Section 10.1 (Positive Assertions)). The list of keys to be signed MUST be part of the message. The list is stored with the key "openid.signed". The value is a comma-separated list of keys, with the "openid." prefix stripped. This algorithm is only capable of signing keys that start with "openid."
2. Iterate through the list of keys to be signed in the order they appear in "openid.signed" list. For each key, find the value in the message whose key is equal to the signed list key prefixed with "openid."
3. Convert the list of key/value pairs to be signed to an octet string by encoding with Key-Value Form Encoding (Key-Value Form Encoding).
4. Determine the signature algorithm from the association type (Establishing Associations). Apply the signature algorithm (Signature Algorithms) to the octet string.

6.2. Signature Algorithms

OpenID Authentication supports two signature algorithms:

- HMAC-SHA1 - 160 bit key length ([RFC2104] (Krawczyk, H., Bellare, M., and R. Canetti, “HMAC: Keyed-Hashing for Message Authentication,” .) and [RFC3174] (Eastlake, D. and P. Jones, “US Secure Hash Algorithm 1 (SHA1),” .))
- HMAC-SHA256 - 256 bit key length ([RFC2104] (Krawczyk, H., Bellare, M., and R. Canetti, “HMAC: Keyed-Hashing for Message Authentication,” .) and [FIPS180-2] (U.S. Department of Commerce and National Institute of Standards and Technology, “Secure Hash Signature Standard,” .)

If supported, the use of HMAC-SHA256 is RECOMMENDED.

7. Initiation and Discovery

7.1. Initiation

To initiate OpenID Authentication, the Relying Party SHOULD present the end user with a form that has a field for entering a User-Supplied Identifier.

The form field's "name" attribute SHOULD have the value "openid_identifier", so that User-Agents can automatically determine that this is an OpenID form. Browser extensions or other software that support OpenID Authentication may not detect a Relying Party's support if this attribute is not set appropriately.

7.2. Normalization

The end user's input MUST be normalized into an Identifier, as follows:

1. If the user's input starts with the "xri://" prefix, it MUST be stripped off, so that XRIs are used in the canonical form.
2. If the first character of the resulting string is an XRI Global Context Symbol ("=", "@", "+", "$", "!") or "(", as defined in Section 2.2.1 of [XRI_Syntax_2.0] (Reed, D. and D. McAlpin, “Extensible Resource Identifier (XRI) Syntax V2.0,” .), then the input SHOULD be treated as an XRI.
3. Otherwise, the input SHOULD be treated as an http URL; if it does not include a "http" or "https" scheme, the Identifier MUST be prefixed with the string "http://". If the URL contains a fragment part, it MUST be stripped off together with the fragment delimiter character "#". See Section 11.5.2 (HTTP and HTTPS URL Identifiers) for more information.
4. URL Identifiers MUST then be further normalized by both following redirects when retrieving their content and finally applying the rules in Section 6 of [RFC3986] (Berners-Lee, T., “Uniform Resource Identifiers (URI): Generic Syntax,” .) to the final destination URL. This final URL MUST be noted by the Relying Party as the Claimed Identifier and be used when requesting authentication (Requesting Authentication).

See normalization example (Normalization).

7.3. Discovery

Discovery is the process where the Relying Party uses the Identifier to look up ("discover") the necessary information for initiating requests. OpenID Authentication has three paths through which to do discovery:

1. If the identifier is an XRI, [XRI_Resolution_2.0] (Wachob, G., Reed, D., Chasen, L., Tan, W., and S. Churchill, “Extensible Resource Identifier (XRI) Resolution V2.0 - Committee Draft 02,” .) will yield an XRDS document that contains the necessary information. It should also be noted that Relying Parties can take advantage of XRI Proxy Resolvers, such as the one provided by XDI.org at http://www.xri.net. This will remove the need for the RPs to perform XRI Resolution locally.
2. If it is a URL, the Yadis protocol (Miller, J., “Yadis Specification 1.0,” .) [Yadis] SHALL be first attempted. If it succeeds, the result is again an XRDS document.
3. If the Yadis protocol fails and no valid XRDS document is retrieved, or no Service Elements (OpenID Service Elements) are found in the XRDS document, the URL is retrieved and HTML-Based discovery (HTML-Based Discovery) SHALL be attempted.

7.3.1. Discovered Information

Upon successful completion of discovery, the Relying Party will have one or more sets of the following information (see the Terminology section (Terminology) for definitions). If more than one set of the following information has been discovered, the precedence rules defined in [XRI_Resolution_2.0] (Wachob, G., Reed, D., Chasen, L., Tan, W., and S. Churchill, “Extensible Resource Identifier (XRI) Resolution V2.0 - Committee Draft 02,” .) are to be applied.

- OP Endpoint URL
- Protocol Version

If the end user did not enter an OP Identifier, the following information will also be present:

- Claimed Identifier
- OP-Local Identifier

If the end user entered an OP Identifier, there is no Claimed Identifier. For the purposes of making OpenID Authentication requests, the value "http://specs.openid.net/auth/2.0/identifier_select" MUST be used as both the Claimed Identifier and the OP-Local Identifier when an OP Identifier is entered.

7.3.2. XRDS-Based Discovery

If XRI or Yadis discovery was used, the result will be an XRDS Document. This is an XML document with entries for services that are related to the Identifier. It is defined in Section 3 of (Wachob, G., Reed, D., Chasen, L., Tan, W., and S. Churchill, “Extensible Resource Identifier (XRI) Resolution V2.0 - Committee Draft 02,” .) [XRI_Resolution_2.0]. See Appendix A.3 (XRDS) for an example XRDS document.

7.3.2.1. OpenID Service Elements

7.3.2.1.1. OP Identifier Element

An OP Identifier Element is an <xrd:Service> element with the following information:

An <xrd:Type> tag whose text content is "http://specs.openid.net/auth/2.0/server".
An <xrd:URI> tag whose text content is the OP Endpoint URL

7.3.2.1.2. Claimed Identifier Element

A Claimed Identifier Element is an <xrd:Service> element with the following information:

An <xrd:Type> tag whose text content is "http://specs.openid.net/auth/2.0/signon".
An <xrd:URI> tag whose text content is the OP Endpoint URL.
An <xrd:LocalID> tag (optional) whose text content is the OP-Local Identifier.

7.3.2.2. Extracting Authentication Data

Once the Relying Party has obtained an XRDS document, it MUST first search the document (following the rules described in [XRI_Resolution_2.0] (Wachob, G., Reed, D., Chasen, L., Tan, W., and S. Churchill, “Extensible Resource Identifier (XRI) Resolution V2.0 - Committee Draft 02,” .)) for an OP Identifier Element. If none is found, the RP will search for a Claimed Identifier Element.

7.3.2.3. XRI and the CanonicalID Element

When the Identifier is an XRI, the <xrd:XRD> element that contains the OpenID Authentication <xrd:Service> element MUST also contain a <CanonicalID> element. The content of this element MUST be used as the Claimed Identifier (see Section 11.5 (Identifying the end user)). This is a vital security consideration because a primary purpose of the <CanonicalID> element is to assert a persistent identifier that will never be reassigned, thus preventing the possibility of an XRI being ("taken over") by a new registrant.

The Relying Party MUST confirm that the provider of the XRD that contains the <CanonicalID> element is authoritative for that Canonical ID and that this XRDS

document is authoritative for the OpenID Service Element. Relying Parties should either do this manually or ensure that their resolver does this.

When using XRI resolution, the Canonical ID MUST be used as the Claimed Identifier. For an XRI to be a valid Identifier, both the <ProviderID> and <CanonicalID> MUST be present in the discovered XRDS document.

When using URL Identifiers, the CanonicalID element MUST be ignored if present.

7.3.2.4. Additional Information

The "openid" namespace is no longer used as of OpenID Authentication 2.0. The "xrd" namespace is "xri://$xrd*($v*2.0)".

For compatibility with deployed code, it is RECOMMENDED that Relying Parties also accept "http://openid.net/signon/1.0" or "http://openid.net/signon/1.1" for the value of <xrd:Type>, as described in the OpenID Authentication 1.1 Compatibility mode (OpenID Authentication 1.1 Compatibility) section. It is RECOMMENDED that Relying Parties supporting OpenID Authentication 2.0 choose to use, if available, endpoints with the type "http://specs.openid.net/auth/2.0/server" and "http://specs.openid.net/auth/2.0/signon", in this order, as specified in Section 7.3.2.2 (Extracting Authentication Data)

If an OP supports extensions (Section 12 (Extensions)), the extensions SHOULD be listed as additional <xrd:Type> child elements of the <xrd:Service> element.

7.3.3. HTML-Based Discovery

HTML-Based discovery MUST be supported by Relying Parties. HTML-Based discovery is only usable for discovery of Claimed Identifiers. OP Identifiers must be XRIs or URLs that support XRDS discovery.

To use HTML-Based discovery, an HTML document MUST be available at the URL of the Claimed Identifier. Within the HEAD element of the document:

A LINK element MUST be included with attributes "rel" set to "openid2.provider" and "href" set to an OP Endpoint URL

A LINK element MAY be included with attributes "rel" set to "openid2.local_id" and "href" set to the end user's OP-Local Identifier

The protocol version when HTML discovery is performed is "http://specs.openid.net/auth/2.0/signon".

The host of the HTML document MAY be different from the end user's OP's host.

The "openid2.provider" and "openid2.local_id" URLs MUST NOT include entities other than "&", "<", ">", and """. Other characters that would not be valid in the HTML document or that cannot be represented in the document's character encoding MUST be escaped using the percent-encoding (%xx) mechanism described in [RFC3986] (Berners-Lee, T., “Uniform Resource Identifiers (URI): Generic Syntax,” .).

As discussed in the OpenID Authentication 1.1 Compatibility mode (OpenID Authentication 1.1 Compatibility) section, these discovery tags are not the same as in previous versions of the protocol. While the same data is conveyed, the names have changed which allows a Relying Party to determine the protocol version being used. A Relying Party MAY encounter a Claimed Identifier which uses HTML-Based Discovery to advertise both version 1.1 and 2.0 Providers.

---

8. Establishing Associations

An association between the Relying Party and the OpenID Provider establishes a shared secret between them, which is used to verify subsequent protocol messages and reduce round trips.

It is RECOMMENDED that a Relying Party form associations if it is possible for it to do so. If a Relying Party is incapable of creating or storing associations, Section 11.4.2 (Verifying Directly with the OpenID Provider) provides an alternate verification mechanism referred to as Stateless Mode.

8.1. Association Session Request

An association session is initiated by a direct request (Direct Communication) from a Relying Party to an OP Endpoint URL with the "openid.mode" key having the value of "associate".

8.1.1. Common Request Parameters

These parameters are common to all association requests:

- openid.ns

  As specified in Section 4.1.2 (HTTP Encoding).

- openid.mode

  Value: "associate"

- openid.assoc_type

  The preferred association type. The association type defines the algorithm to be used to sign subsequent messages.

  Value: A valid association type from Section 8.3 (Association Types)

- openid.session_type

  The preferred association session type. This defines the method used to encrypt the association's MAC key in transit.

  Value: A valid association session type from Section 8.4 (Association Session Types).

Note: Unless using transport layer encryption, "no-encryption" MUST NOT be used.

8.1.2. Diffie-Hellman Request Parameters

The following parameters are common to requests whose requested association session type is "DH-SHA1" or "DH-SHA256":

- openid.dh_modulus

  Value: base64(btwoc(p))

  Default: See Appendix B (Diffie-Hellman Key Exchange Default Value)

- openid.dh_gen

  Value: base64(btwoc(g))

  Default: g = 2

- openid.dh_consumer_public

  Value: base64(btwoc(g ^ xa mod p))

See Section 8.4.2 (Diffie-Hellman Association Sessions) for more information on these parameters.

NOTE: The 'btwoc' function is defined in Section 4.2 (Integer Representations).

8.2. Association Session Response

An association session response is a direct response from the OP to the Relying Party in Key-Value Form (Key-Value Form Encoding).

8.2.1. Common Response Parameters

- ns

  As specified in Section 5.1.2 (Direct Response).

- assoc_handle

The association handle is used as a key to refer to this association in subsequent messages.

Value: A string 255 characters or less in length. It MUST consist only of ASCII characters in the range 33-126 inclusive (printable non-whitespace characters).

- session_type

  The value of the "openid.session_type" parameter from the request. If the OP is unwilling or unable to support this association type, it MUST return an unsuccessful response (Unsuccessful Response Parameters).

- assoc_type

  The value of the "openid.assoc_type" parameter from the request. If the OP is unwilling or unable to support this association type, it MUST return an unsuccessful response (Unsuccessful Response Parameters).

- expires_in

  The lifetime, in seconds, of this association. The Relying Party MUST NOT use the association after this time has passed.

  Value: An integer, represented in base 10 ASCII.

8.2.2. Unencrypted Response Parameters

- mac_key

  The MAC key (shared secret) for this association, Base 64 (Josefsson, S., “The Base16, Base32, and Base64 Data Encodings,” .) [RFC3548] encoded.

8.2.3. Diffie-Hellman Response Parameters

- dh_server_public

  Value: base64(btwoc(g ^ xb mod p))

  Description: The OP's Diffie-Hellman public key.

- enc_mac_key

  Value: base64(H(btwoc(g ^ (xa * xb) mod p)) XOR MAC key)

Description: The MAC key (shared secret), encrypted with the secret Diffie-Hellman value. H is either "SHA1" or "SHA256" depending on the session type.

NOTE: The 'btwoc' function is defined in Section 4.2 (Integer Representations)

8.2.4. Unsuccessful Response Parameters

If the OP does not support a session type or association type, it MUST respond with a direct error message indicating that the association request failed. If there is another association session type or association type that is supported, the OP SHOULD include that information in the response.

- ns

  As specified in Section 5.1.2 (Direct Response).

- error

  Value: A human-readable message indicating why the association request failed.

- error_code

  Value: "unsupported-type"

- session_type

  Value: (optional) A valid association session type from Section 8.4 (Association Session Types) that the OP supports.

- assoc_type

  Value: (optional) An association type supported by the OP from Section 8.3 (Association Types).

Upon receipt of an "unsupported-type" response, the Relying Party MAY make another request with the specified association session type and association type. If no association is established, the Relying Party MAY continue the authentication process in Direct Verification (Verifying Directly with the OpenID Provider).

8.3. Association Types

8.3.1. HMAC-SHA1

An association of type "HMAC-SHA1" uses the HMAC-SHA1 (Signature Algorithms) signature algorithm.

8.3.2. HMAC-SHA256

An association of type "HMAC-SHA256" uses the HMAC-SHA256 (Signature Algorithms) signature algorithm.

8.4. Association Session Types

OpenID Authentication defines three valid association session types: "no-encryption", "DH-SHA1", and "DH-SHA256".

8.4.1. No-Encryption Association Sessions

In a "no-encryption" association session, the OP sends the association MAC key in plain-text to the Relying Party. This makes it possible for an eavesdropper to intercept the key, and forge messages to this Relying Party when not using transport layer encryption. Therefore, "no-encryption" association sessions MUST NOT be used unless the messages are using transport layer encryption. See Section 15.1.1 (Eavesdropping Attacks) for more information.

The MAC key sent by the OP MUST be the length specified for the requested association type, as specified in Section 6.2 (Signature Algorithms).

8.4.2. Diffie-Hellman Association Sessions

The "DH-SHA1" and "DH-SHA256" association types use Diffie-Hellman Key Exchange to securely transmit the shared secret.

The MAC key MUST be the same length as the output of H, the hash function - 160 bits (20 bytes) for DH-SHA1 or 256 bits (32 bytes) for DH-SHA256, as well as the output of the signature algorithm of this association.

The Relying Party specifies a modulus, p, and a generator, g. The Relying Party chooses a random private key xa and OpenID Provider chooses a random private key xb, both in the range [1 .. p-1]. The shared secret used to encrypt the MAC key is thus g ^ (xa * xb) mod p = (g ^ xa) ^ xb mod p = (g ^ xb) ^ xa mod p. For more information, see [RFC2631] (Rescorla, E., “Diffie-Hellman Key Agreement Method,” .). For information

on the selection of random values, see [RFC1750] (Eastlake, D., Crocker, S., and J. Schiller, “Randomness Recommendations for Security,” .).

---

9. Requesting Authentication

Once the Relying Party has successfully performed discovery and (optionally) created an association with the discovered OP Endpoint URL, it can send an authentication request to the OP to obtain an assertion. An authentication request is an indirect request (Indirect Communication).

9.1. Request Parameters

- openid.ns

  As specified in Section 4.1.2 (HTTP Encoding).

- openid.mode

  Value: "checkid_immediate" or "checkid_setup"

  Note: If the Relying Party wishes the end user to be able to interact with the OP, "checkid_setup" should be used. An example of a situation where interaction between the end user and the OP is not desired is when the authentication request is happening asynchronously in JavaScript.

- openid.claimed_id

  Value: (optional) The Claimed Identifier.

  "openid.claimed_id" and "openid.identity" SHALL be either both present or both absent. If neither value is present, the assertion is not about an identifier, and will contain other information in its payload, using extensions (Extensions).

  It is RECOMMENDED that OPs accept XRI identifiers with or without the "xri://" prefix, as specified in the Normalization (Normalization) section.

- openid.identity

  Value: (optional) The OP-Local Identifier.

  If a different OP-Local Identifier is not specified, the claimed identifier MUST be used as the value for openid.identity.

  Note: If this is set to the special value "http://specs.openid.net/auth/2.0/identifier_select" then the OP SHOULD choose an Identifier that belongs to the end user. This parameter MAY be omitted if the request is not about an identifier (for instance if an extension is in use that makes the request meaningful without it; see openid.claimed_id above).

- openid.assoc_handle

  Value: (optional) A handle for an association between the Relying Party and the OP that SHOULD be used to sign the response.

  Note: If no association handle is sent, the transaction will take place in Stateless Mode (Verifying Directly with the OpenID Provider).

- openid.return_to

  Value: (optional) URL to which the OP SHOULD return the User-Agent with the response indicating the status of the request.

  Note: If this value is not sent in the request it signifies that the Relying Party does not wish for the end user to be returned.

  Note: The return_to URL MAY be used as a mechanism for the Relying Party to attach context about the authentication request to the authentication response. This document does not define a mechanism by which the RP can ensure that query parameters are not modified by outside parties; such a mechanism can be defined by the RP itself.

- openid.realm

  Value: (optional) URL pattern the OP SHOULD ask the end user to trust. See Section 9.2 (Realms). This value MUST be sent if openid.return_to is omitted.

  Default: return_to URL

9.2. Realms

A "realm" is a pattern that represents the part of URL-space for which an OpenID Authentication request is valid. A realm is designed to give the end user an indication of the scope of the authentication request. OPs SHOULD present the realm when requesting the end user's approval for an authentication request. The realm SHOULD be used by OPs to uniquely identify Relying Parties. For example, OPs MAY use the realm to allow the end user to automate approval of authentication requests.

A realm pattern is a URL, with the following changes:

- A realm MUST NOT contain a URI fragment
- A realm MAY contain a wild-card at the beginning of the URL authority section. A wild-card consists of the characters "*." prepended to the DNS name in the authority section of the URL.

A URL matches a realm if:

- The URL scheme and port of the URL are identical to those in the realm. See RFC 3986 (Berners-Lee, T., “Uniform Resource Identifiers (URI): Generic Syntax,” .) [RFC3986], section 3.1 for rules about URI matching.
- The URL's path is equal to or a sub-directory of the realm's path.
- Either:
    1. The realm's domain contains the wild-card characters "*.", and the trailing part of the URL's domain is identical to the part of the realm following the "*." wildcard, or
    2. The URL's domain is identical to the realm's domain

When present, the "openid.return_to" URL MUST match the "openid.realm", or the OP MUST return an indirect error response (Indirect Error Responses).

It is RECOMMENDED that OPs protect their users from making assertions with overly-general realms, like http://*.com/ or http://*.co.uk/. Overly general realms can be dangerous when the realm is used for identifying a particular Relying Party. Whether a realm is overly-general is at the discretion of the OP.

9.2.1. Using the Realm for Return URL Verification

OpenID providers SHOULD verify that the return_to URL specified in the request is an OpenID relying party endpoint. To verify a return_to URL, obtain the relying party endpoints for the realm by performing discovery on the relying party (Discovering OpenID Relying Parties). As always when performing discovery, the discovered URL is the URL of the last HTTP response, following redirects. If any redirects are followed when performing discovery on the realm, verification has failed. If discovery has successfuly completed, check to make sure that the return_to URL matches one of the relying party endpoints.

A realm may contain a wildcard, and so may not be a valid URL. In that case, perform discovery on the URL obtained by substituting "www" for the wildcard in the realm.

To match a return_to URL against a relying party endpoint, use the same rules as for matching the return_to URL against the realm, treating the relying party's endpoint URL as the realm. Relying party endpoint URLs MUST NOT contain a domain wildcard, and SHOULD be as specific as possible.

If verification is attempted and fails, the provider SHOULD NOT send a positive assertion to that return_to URL.

Providers MAY cache verified return_to URLs.

9.3. Immediate Requests

When requesting authentication, the Relying Party MAY request that the OP not interact with the end user. In this case the OP MUST respond immediately with either an assertion that authentication is successful, or a response indicating that the request cannot be completed without further user interaction. This is accomplished by an authentication request with "openid.mode" set to "checkid_immediate".

---

10. Responding to Authentication Requests

When an authentication request comes from the User-Agent via indirect communication (Indirect Communication), the OP SHOULD determine that an authorized end user wishes to complete the authentication. If an authorized end user wishes to complete the authentication, the OP SHOULD send a positive assertion (Positive Assertions) to the Relying Party.

Methods of identifying authorized end users and obtaining approval to return an OpenID Authentication assertion are beyond the scope of this specification. See Section 15.1.2.1 (Rogue Relying Party Proxying) for OpenID Provider security considerations.

If the relying party requested OP-driven identifier selection by setting "openid.identity" to "http://specs.openid.net/auth/2.0/identifier_select" and there are Identifiers for which the end user is authorized to issue authentication responses, the OP SHOULD allow the end user to choose which Identifier to use.

If the Relying Party supplied an association handle with the authentication request, the OP SHOULD attempt to look up an association based on that handle. If the association is missing or expired, the OP SHOULD send the "openid.invalidate_handle" parameter as part of the response with the value of the request's "openid.assoc_handle" parameter, and SHOULD proceed as if no association handle was specified.

If no association handle is specified, the OP SHOULD use a private association for signing the response. The OP MUST store this association and MUST respond to later requests to check the signature of the response via Direct Verification (Verifying Directly with the OpenID Provider).

Relying Parties SHOULD accept and verify assertions about Identifiers for which they have not requested authentication. OPs SHOULD use private associations for signing unsolicited positive assertions.

If the "openid.return_to" value is omitted in the request, the Relying Party does not wish to receive an authentication assertion from the OP. This can be useful when using extensions to transfer data from the Relying Party to the OP.

10.1. Positive Assertions

Positive assertions are indirect responses (Indirect Communication) with the following fields:

- openid.ns

  As specified in Section 4.1.2 (HTTP Encoding).

- openid.mode

  Value: "id_res"

- openid.op_endpoint

  The OP Endpoint URL.

- openid.claimed_id

  Value: (optional) The Claimed Identifier. "openid.claimed_id" and "openid.identity" SHALL be either both present or both absent.

  Note: The end user MAY choose to use an OP-Local Identifier as a Claimed Identifier.

  Note: If neither Identifier is present in the assertion, it is not about an identifier, and will contain other information in its payload, using extensions (Extensions).

- openid.identity

  Value: (optional) The OP-Local Identifier

  Note: OpenID Providers MAY assist the end user in selecting the Claimed and OP-Local Identifiers about which the assertion is made. The openid.identity field MAY be omitted if an extension is in use that makes the response meaningful without it (see openid.claimed_id above).

- openid.return_to

  Value: Verbatim copy of the return_to URL parameter sent in the request.

- openid.response_nonce

  Value: A string 255 characters or less in length, that MUST be unique to this particular successful authentication response. The nonce MUST start with the current time on the server, and MAY contain additional ASCII characters in the range 33-126 inclusive (printable non-whitespace characters), as necessary to make each response unique. The date and time MUST be formatted as specified in section 5.6 of [RFC3339] (Klyne, G. and C. Newman, “Date and Time on the Internet: Timestamps,” .), with the following restrictions:

    - All times must be in the UTC timezone, indicated with a "Z".
    - No fractional seconds are allowed

  For example: 2005-05-15T17:11:51ZUNIQUE

- openid.invalidate_handle

  Value: (optional) If the Relying Party sent an invalid association handle with the request, it SHOULD be included here.

- openid.assoc_handle

  Value: The handle for the association that was used to sign this assertion.

- openid.signed

  Value: Comma-separated list of signed fields.

  Note: This entry consists of the fields without the "openid." prefix that the signature covers. This list MUST contain at least "op_endpoint", "return_to" "response_nonce" and "assoc_handle", and if present in the response, "claimed_id" and "identity". Additional keys MAY be signed as part of the message. See Generating Signatures (Generating Signatures).

  For example, "op_endpoint,identity,claimed_id,return_to,assoc_handle,response_nonce".

- openid.sig

  Value: Base 64 encoded signature calculated as specified in Section 6 (Generating Signatures).

10.2. Negative Assertions

If the OP is unable to identify the end user or the end user does not or cannot approve the authentication request, the OP SHOULD send a negative assertion to the Relying Party as an indirect response (Indirect Communication).

When receiving a negative assertion in response to a "checkid_immediate" mode request, Relying Parties SHOULD construct a new authentication request using "checkid_setup" mode. Details about how this differs from OpenID Authentication 1.1 can be found in Section 14 (OpenID Authentication 1.1 Compatibility).

10.2.1. In Response to Immediate Requests

If the request was an immediate request, there is no chance for the end user to interact with pages on the OP to provide identifying credentials or approval of a request. A negative assertion of an immediate request takes the following form:

- openid.ns

  As specified in Section 4.1.2 (HTTP Encoding).

- openid.mode

  Value: "setup_needed"

10.2.2. In Response to Non-Immediate Requests

Since the OP may display pages to the end user and request credentials from the end user, a negative response to a request that is not immediate is definitive. It takes the following form:

- openid.ns

  As specified in Section 4.1.2 (HTTP Encoding).

- openid.mode

  Value: "cancel"

Often, if the user does not wish to or cannot complete the authentication request, the OpenID authentication process will be aborted and the Relying Party will not get a cancel mode response (the end user may quit or press the back button in their User-Agent instead of continuing). If a RP receives the "cancel" response, authentication was unsuccessful and the RP MUST treat the end user as non-authenticated.

11. Verifying Assertions

When the Relying Party receives a positive assertion, it MUST verify the following before accepting the assertion:

- The value of "openid.return_to" matches the URL of the current request (Section 11.1 (Verifying the Return URL))
- Discovered information matches the information in the assertion (Section 11.2 (Verifying Discovered Information))
- An assertion has not yet been accepted from this OP with the same value for "openid.response_nonce" (Section 11.3 (Checking the Nonce))
- The signature on the assertion is valid and all fields that are required to be signed are signed (Section 11.4 (Verifying Signatures))

If all four of these conditions are met, assertion is now verified. If the assertion contained a Claimed Identifier, the user is now authenticated with that identifier.

11.1. Verifying the Return URL

To verify that the "openid.return_to" URL matches the URL that is processing this assertion:

- The URL scheme, authority, and path MUST be the same between the two URLs.
- Any query parameters that are present in the "openid.return_to" URL MUST also be present with the same values in the URL of the HTTP request the RP received.

11.2. Verifying Discovered Information

If the Claimed Identifier in the assertion is a URL and contains a fragment, the fragment part and the fragment delimiter character "#" MUST NOT be used for the purposes of verifying the discovered information.

If the Claimed Identifier is included in the assertion, it MUST have been discovered (Discovery) by the Relying Party and the information in the assertion MUST be present in the discovered information. The Claimed Identifier MUST NOT be an OP Identifier.

If the Claimed Identifier was not previously discovered by the Relying Party (the "openid.identity" in the request was "http://specs.openid.net/auth/2.0/identifier_select" or a different Identifier, or if the OP is sending an unsolicited positive assertion), the Relying Party MUST perform discovery on the Claimed Identifier in the response to make sure that the OP is authorized to make assertions about the Claimed Identifier.

If no Claimed Identifier is present in the response, the assertion is not about an identifier and the RP MUST NOT use the User-supplied Identifier associated with the current OpenID authentication transaction to identify the user. Extension information in the assertion MAY still be used.

Discovered Value	Response Field
Claimed Identifier	openid.claimed_id
OP-Local Identifier	openid.identity
OP Endpoint URL	openid.op_endpoint
Protocol Version	openid.ns

This table shows the mapping of discovered information (Discovered Information) into fields in the OpenID Authentication 2.0 "id_res" response (Positive Assertions)

Discovered Information to Authentication Response Mapping

If using a discovery mechanism that yields an XRDS document, the protocol version, OP Endpoint URL and the OP-Local Identifier (if different than the Claimed Identifier) MUST be present in one <xrd:Service> element. There MAY be unused fields in that <xrd:Service> element.

Non-normative example:

```
<Service xmlns="xri://$xrd*($v*2.0)">
  <Type>http://specs.openid.net/auth/2.0/signon</Type>
  <URI>http://provider.example.com/openid</URI>
  <URI>https://provider.example.com/openid</URI>
</Service>
```

In this example XRDS snippet, the <xrd:Service> element has two <xrd:URI> elements, which map to OP Endpoint URLs as per Section 7.3.1 (Discovered Information). If an assertion has either value for "openid.op_endpoint", then that field matches this <xrd:Service> element. The other <xrd:URI> element is unused.

11.3. Checking the Nonce

To prevent replay attacks, the agent checking the signature keeps track of the nonce values included in positive assertions and never accepts the same value more than once for the same OP Endpoint URL.

- When using "check_authentication", the OP MUST NOT issue more than one successful response to a request with the same value for "openid.response_nonce".
- When the Relying Party checks the signature on an assertion, the Relying Party SHOULD ensure that an assertion has not yet been accepted with the same value for "openid.response_nonce" from the same OP Endpoint URL.

The time-stamp MAY be used to reject responses that are too far away from the current time, limiting the amount of time that nonces must be stored to prevent attacks. The acceptable range is out of the scope of this specification. A larger range requires storing more nonces for a longer time. A shorter range increases the chance that clock-skew and transaction time will cause a spurious rejection.

11.4. Verifying Signatures

If the Relying Party has stored an association with the association handle specified in the assertion, it MUST check the signature on the assertion itself. If it does not have an association stored, it MUST request that the OP verify the signature via Direct Verification (Verifying Directly with the OpenID Provider).

11.4.1. Verifying with an Association

The Relying Party follows the same procedure that the OP followed in generating the signature (Generating Signatures), and then compares the signature in the response to the signature it generated. If the signatures do not match, the assertion is invalid.

If an authentication request included an association handle for an association between the OP and the Relying party, and the OP no longer wishes to use that handle (because it has expired or the secret has been compromised, for instance), the OP will send a response that must be verified directly with the OP, as specified in Section 11.4.2 (Verifying Directly with the OpenID Provider). In that instance, the OP will include the field "openid.invalidate_handle" set to the association handle that the Relying Party included with the original request.

11.4.2. Verifying Directly with the OpenID Provider

To have the signature verification performed by the OP, the Relying Party sends a direct request (Direct Request) to the OP. To verify the signature, the OP uses a private association that was generated when it issued the positive assertion (Positive Assertions).

11.4.2.1. Request Parameters

- openid.mode

  Value: "check_authentication"

- Exact copies of all fields from the authentication response, except for "openid.mode".

For verifying signatures an OP MUST only use private associations and MUST NOT use associations that have shared keys. If the verification request contains a handle for a shared association, it means the Relying Party no longer knows the shared secret, or an entity other than the RP (e.g. an attacker) has established this association with the OP.

To prevent replay attacks, the OP MUST NOT issue more than one verification response for each authentication response it had previously issued. An authentication response and its matching verification request may be identified by their "openid.response_nonce" values.

11.4.2.2. Response Parameters

- ns

  As specified in Section 5.1.2 (Direct Response).

- is_valid

  Value: "true" or "false"; asserts whether the signature of the verification request is valid.

- invalidate_handle

  Value: (optional) The "invalidate_handle" value sent in the verification request, if the OP confirms it is invalid.

  Description: If present in a verification response with "is_valid" set to "true", the Relying Party SHOULD remove the corresponding association from its store and SHOULD NOT send further authentication requests with this handle.

  Note: This two-step process for invalidating associations is necessary to prevent an attacker from invalidating an association at will by adding "invalidate_handle" parameters to an authentication response.

11.5. Identifying the end user

The Claimed Identifier in a successful authentication response SHOULD be used by the Relying Party as a key for local storage of information about the user. The Claimed Identifier MAY be used as a user-visible Identifier. When displaying URL Identifiers, the fragment MAY be omitted.

11.5.1. Identifier Recycling

OpenID Providers with large user bases can use fragments to recycle URL Identifiers if it is so desired. When reassigning a URL Identifier to a new end user OPs should generate a new, unique fragment part.

The full URL with the fragment part constitutes the Claimed Identifier in positive assertions, therefore Relying Parties will distinguish between the current and previous owners of the fragment-less URL.

This mechanism allows the (presumably short, memorable) recycled URL Identifiers without the fragment to be used by end users at login time and by Relying Parties for display purposes.

11.5.2. HTTP and HTTPS URL Identifiers

Relying Parties MUST differentiate between URL Identifiers that have different schemes. When end user input is processed into a URL, it is processed into a HTTP URL. If the same end user controls the same URL, differing only by scheme, and it is desired that the Identifier be the HTTPS URL, it is RECOMMENDED that a redirect be issued from the HTTP URL to the HTTPS URL. Because the HTTP and HTTPS URLs are not equivalent and the Identifier that is used is the URL after following redirects, there is no foreseen reduction in security when using this scheme. If an attacker could gain control of the HTTP URL, it would have no effect on the HTTPS URL, since the HTTP URL is not ever used as an Identifier except to initiate the discovery process.

12. Extensions

An Extension to OpenID Authentication is a protocol that "piggybacks" on the authentication request and response. Extensions are useful for providing extra information about an authentication request or response as well as providing extra information about the subject of the authentication response.

OpenID extensions are identified by a Type URI. The Type URI MAY be used as the value of an <xrd:Type> element of an OpenID <xrd:Service> element in an XRDS document associated with a Claimed Identifier. The Type URI is also used to associate key-value pairs in messages with the extension.

To associate keys and values in a message with an extension, the key MUST be associated with the Type URI. To associate keys with a Type URI, establish an alias by adding a key prefixed with "openid.ns." and ending with the alias text whose value is the Type URI. Once an alias has been established, all pairs in the message whose keys start with "openid." followed by the alias text, followed by a period or the end of the key are associated with that extension. This mechanism is similar to the XML namespaces.

A namespace alias MUST NOT contain a period and MUST NOT be the same as another namespace alias in the same message. A namespace alias also MUST NOT be in the following list of disallowed aliases:

- assoc_handle
- assoc_type
- claimed_id
- contact
- delegate
- dh_consumer_public
- dh_gen
- dh_modulus
- error
- identity
- invalidate_handle
- mode
- ns
- op_endpoint
- openid
- realm
- reference
- response_nonce
- return_to
- server
- session_type
- sig
- signed

- trust_root

A namespace MUST NOT be assigned more than one alias in the same message. If a message is a response to another message, the response MAY use a different alias to refer to the same namespace.

Non-normative example:

An extension's type URI is "<http://example.com/ext/1.0>".

openid.ns.x=http://example.com/ext/1.0

openid.x=example

openid.x.foo=bar

openid.xx=notx

In this example, the keys "openid.x" and "openid.x.foo" are associated with the extension; the "openid.xx" key is not.

Extensions MUST NOT define multiple parameters with the same name. Extensions that need to send multiple values for the same parameter name must define their own conventions for doing so.

---

13. Discovering OpenID Relying Parties

Relying Party discovery allows for software agents to discover sites that support OpenID. It also allows OpenID providers to automatically verify that a return_to URL in an OpenID request is an OpenID relying party endpoint for the specified realm.

Relying Parties SHOULD use the Yadis protocol to publish their valid return_to URLs. The relying party MAY publish this information at any URL, and SHOULD publish it under the realm so that providers can verify return_to URLs.

A Relying Party discovery XRDS document MUST contain one or more <xrd:Service> elements:

- Containing at least one <xrd:URI> element.
- Where all <xrd:URI> tags contain a URL that accepts OpenID 2.0 Authentication responses.
- Containing a <xrd:Type> tag whose content is "http://specs.openid.net/auth/2.0/return_to".

Non-normative example:

```
<Service xmlns="xri://$xrd*($v*2.0)">
  <Type>http://specs.openid.net/auth/2.0/return_to</Type>
  <URI>http://consumer.example.com/return</URI>
</Service>
```

14. OpenID Authentication 1.1 Compatibility

This section describes how to interact with OpenID Authentication 1.1 Relying Parties and OPs. OpenID Authentication 2.0 implementations SHOULD support OpenID Authentication 1.1 compatibility, unless security considerations make it undesirable.

14.1. Changes from OpenID Authentication 1.1

(non-normative)

This specification is based on the original specification for OpenID Authentication as written by Brad Fitzpatrick. That specification did not have a version number, but was called OpenID 1.0, and then OpenID 1.1 when it was revised. The protocol outlined in this specification is intended to be backwards-compatible with the revised OpenID protocol. The changes to the specification are outlined in this section.

14.1.1. Updated Initiation and Discovery

- Supports OP Identifiers. This new variation of the protocol flow is initiated by an end user entering an OP Identifier instead of their own Identifier. This allows the OP to assist the end user in selecting an Identifier.
- Supports the use of XRIs as Identifiers. XRIs may be used as Identifiers for both end users and OPs, and provide automatic mapping from one or more reassignable i-names to a synonymous persistent Canonical ID that will never be reassigned.
- When URLs are used as Identifiers, they are normalized according to the guidelines in [RFC3986] (Berners-Lee, T., “Uniform Resource Identifiers (URI): Generic Syntax,” .), for better compatibility with the existing Web infrastructure.
- Uses the Yadis protocol for discovery. This allows for using multiple OPs for a single Identifier, for load-balancing and fallback in the case of OP failure. Additionally, it allows for discovery of supported extensions and other associated services.

14.1.2. Security improvements

A nonce is now part of the protocol for built-in protection against replay attacks, which was previously implemented out-of-band by each library or application.

A new association type, HMAC-SHA256, and a new association session type, DH-SHA256, allow for stronger signatures on authentication assertions.

An actual Security Considerations section (Security Considerations) which looks at protecting the protocol from end-to-end.

14.1.3. Extensions

Extensions are now an officially supported mechanism to support data exchange and other Relying Party-OP communication along with the authentication exchange. Extensions allow for the exchange of arbitrary attributes, as well as for protocol extensions, such as the inclusion of additional information about the Relying Party in the authentication request.

Because extensions can transfer arbitrary data, the Identifier is now optional in authentication messages.

14.2. Implementing OpenID Authentication 1.1 Compatibility

All messages in OpenID Authentication 1.1 omit the "openid.ns" parameter, which is an easy way for an RP to determine if the message is from an OpenID Authentication 1.1 endpoint. OpenID Authentication 1.1 supports only HMAC-SHA1 associations.

Error responses in OpenID Authentication 1.1 did not define "contact" or "reference". OpenID Authentication 1.1 did allow for the addition of extra fields in error responses. It is RECOMMENDED for contact and reference to be sent even when using OpenID Authentication 1.1, since they may be useful for debugging and do not affect compatibility.

14.2.1. Relying Parties

- When HTML discovery is performed, the OP endpoint URL is marked by the link relationship "openid.server" rather than "openid2.provider". The end user's OP-Local Identifier is marked by the link relationship "openid.delegate" rather than "openid2.local_id". The protocol version is in this case "http://openid.net/signon/1.1". HTML allows multiple link relationships to be specified for a single link, so if an OP provides both OpenID Authentication 1.1 and OpenID Authentication 2.0, "openid2.provider" and "openid.server" may appear in the same "rel" attribute.
- When XRDS-based discovery is performed, the end user's OP-Local Identifier appears in the <openid:Delegate> tag of the OpenID <xrd:Service> element rather than in the <xrd:LocalID> tag. In order to support currently-deployed discovery code, both tags MAY appear in the <xrd:Service> element.
- Relying Parties SHOULD extract and use OpenID Authentication 1.x service elements from XRDS documents, if Yadis succeeds on an URL Identifier. Such service elements are identified by <xrd:Type> tags whose text contents are "http://openid.net/server/1.0" or "http://openid.net/server/1.1". Although this is not specified in the previous version of the protocol, it is a generally accepted practice of advertising OpenID Authentication 1.x services through Yadis.

- "openid.claimed_id" is not defined by OpenID Authentication 1.1. Relying Parties MAY send the value when making requests, but MUST NOT depend on the value being present in authentication responses. When the OP-Local Identifier ("openid.identity") is different from the Claimed Identifier, the Relying Party MUST keep track of what Claimed Identifier was used to discover the OP-Local Identifier, for example by keeping it in session state. Although the Claimed Identifier will not be present in the response, it MUST be used as the identifier for the user.
- "openid.identity" MUST be sent in a authentication request (Responding to Authentication Requests).
- Relying Parties MUST send a blank session_type parameter in "no-encryption" association requests.
- In OpenID Authentication 1.1, the "no-encryption" association session type is represented by a blank or missing "openid.session_type" parameter. Relying Parties MUST NOT send requests with "openid.session_type" set to "no-encryption".
- In authentication requests (Requesting Authentication), the "openid.identity" parameter SHOULD NOT be the special value "http://specs.openid.net/auth/2.0/identifier_select", because OpenID Authentication 1.1 does not support the use of OP Identifiers.
- The "openid.realm" parameter in authentication requests was known as "openid.trust_root". The syntax and meaning are identical.
- When responding with a negative assertion to a "checkid_immediate" mode authentication request, the "user_setup_url" parameter MUST be returned. This is a URL that the end user may visit to complete the request. The OP MAY redirect the end user to this URL, or provide the end user with a link that points to this URL.
- The Relying Party MUST accept an authentication response (Positive Assertions) that is missing the "openid.response_nonce" parameter. It SHOULD implement a method for preventing replay attacks.
- Relying Parties MUST accept authentication responses (Positive Assertions) that are missing the "openid.op_endpoint" parameter.

14.2.2. OpenID Providers

- "openid.identity" MUST be sent in a positive authentication assertion (Positive Assertions).
- In OpenID Authentication 1.1, the "no-encryption" association session type is represented by a blank or missing "openid.session_type" parameter. OPs MUST NOT send responses with "openid.session_type" set to "no-encryption".
- OPs MAY choose to return a successful "no-encryption" response to any association request. As above, the "openid.session_type" parameter MUST be blank or omitted from the response.
- OPs MUST accept association requests with no assoc_type parameter, and assume them to be of type HMAC-SHA1.

- Unsuccessful association attempts MAY be responded with direct error messages or with "no-encryption" positive association responses.
- The "openid.realm" parameter in authentication requests was known as "openid.trust_root". The syntax and meaning are identical.
- When responding with a negative assertion to a "checkid_immediate" mode authentication request, the "user_setup_url" parameter MUST be returned. This is a URL that the end user may visit to complete the request. The Relying Party may redirect the end user to this URL, or provide the end user with a link that points to this URL.
- OPs MUST NOT send the "openid.op_endpoint" parameter in authentication responses (Positive Assertions), since it is not part of the OpenID Authentication 1.1 protocol.

---

15. Security Considerations

15.1. Preventing Attacks

15.1.1. Eavesdropping Attacks

There is one place in this protocol that is vulnerable to eavesdropping attacks.

- If the nonce were not checked, an eavesdropper could also intercept a successful authentication assertion and re-use it.

This attack can be prevented by using transport layer encryption for these connections to prevent eavesdropping. In addition, if not using TLS this attack can still be prevented by checking the nonce while performing message verification. When doing so, the positive authentication assertion cannot be re-used.

15.1.2. Man-in-the-Middle Attacks

Associations prevent tampering of signed fields by a man in the middle except during discovery, association sessions and Direct Verification (Verifying Directly with the OpenID Provider). Altering signed fields without the shared secret requires breaking the MAC. Currently no tractable attack is known on the MACs used in this protocol. The quality of the protection provided by the MAC depends on the randomness of the shared MAC key, so it is important that an unguessable value be used.

If DNS resolution or the transport layer is compromised signatures on messages are not adequate, since the attacker can impersonate the OP and issue its own associations, or its own decisions in Stateless Mode. If an attacker can tamper with the discovery process they can specify any OP, and so does not have to impersonate the OP. Additionally, if an attacker can compromise the integrity of the information returned during the discovery process, by altering the XRDS document, the need for a man in the middle is removed. One method to prevent this sort of attack is by digitally signing the XRDS file as per XMLDSIG (Eastlake 3rd, D., Reagle Jr., J., and D. Solo, “(Extensible Markup Language) XML-Signature Syntax and Processing,” .) [RFC3275]. The keying material is not specified, since the RP ultimately needs to make its own decision whether to trust keys used for such signature.

Using SSL with certificates signed by a trusted authority prevents these kinds of attacks by verifying the results of the DNS look-up against the certificate. Once the validity of the certificate has been established, tampering is not possible. Impersonating an SSL server requires forging or stealing a certificate, which is significantly harder than the network based attacks.

In order to get protection from SSL, SSL must be used for all parts of the interaction, including interaction with the end user through the User-Agent. While the protocol does not require SSL be used, its use is strongly RECOMMENDED. Current best practices dictate that an OP SHOULD use SSL, with a certificate signed by a trusted authority, to secure its Endpoint URL as well as the interactions with the end user's User-Agent. In addition, SSL, with a certificate signed by a trusted authority, SHOULD be used so that a Relying Party can fetch the end user's URL in a secure manner. Following its own security policies, a Relying Party MAY choose to not complete, or even begin, a transaction if SSL is not being correctly used at these various endpoints.

15.1.2.1. Rogue Relying Party Proxying

A special type of man-in-the-middle attack is one where the Relying Party is a rogue party acting as a MITM. The RP would perform discovery on the End User's Claimed Identifier and instead of redirecting the User Agent to the OP, would instead proxy the OP through itself. This would thus allow the RP to capture credentials the End User provides to the OP. While there are multiple ways to prevent this sort of attack, the specifics are outside the scope of this document. Each method of prevention requires that the OP establish a secure channel with the End User.

15.2. User-Agents

Since this protocol is intended to be used interactively, User-Agents will primarily be common Web browsers. Web browsers or their hosts may be infected with spyware or other malware, which limits the strength of the authentication assertion, since untrusted software makes it impossible to know whether the authentication decision has been made with the end user's approval. With that said, many web applications and protocols today rely on the security of the Web browser and their hosts.

Cross-site-scripting attacks against OPs may be used to the same effect. For the best security, OPs should not depend on scripting. This enables User-Agents that do not support scripting, or have scripting disabled, to still employ the protocol.

15.3. User Interface Considerations

The Relying Party SHOULD redirect the end user to the OP Endpoint URL in a top-level browser window with all controls visible. This allows better protection for the end user against OP look-alike sites (phishing).

OpenID Providers SHOULD educate their end users about the potential for OpenID phishing attacks and SHOULD equip their end users with the tools to defeat such attacks, for example browser plug-ins that verify the authenticity of the OP's Authentication Service Endpoint URL.

15.4. HTTP and HTTPS URL Identifiers

While these types of Identifiers have been previously discussed (HTTP and HTTPS URL Identifiers), they are worth mentioning again. As previously stated, the RECOMMENDED method of an End User expressing control over a URL differing only be scheme is to setup a redirect from the HTTP URL to the HTTPS URL. Relying Parties will never store the HTTP URL as during the discovery and initiation phase will follow the redirect and use the HTTPS URL as the Claimed Identifier.

End users with concerns over this recommendation should directly enter their HTTPS URL at each Relying Party. This thus removes the step where the Relying Party follows the redirect to the HTTPS URL. The single security consideration currently seen is if an attacker were to compromise the integrity of the HTTP URL by removing the redirect and pointing the Identifier at a rogue OP. This however will alter the user experience, is detectable by anti-phishing technologies, and the security of the Identifier itself is a fundamental principle within OpenID.

15.5. Denial of Service Attacks

Within the protocol there are places where a rogue RP could launch a denial of service attack against an OP since there is nothing in OpenID protocol messages that allows the OP to quickly check that it is a genuine request. This can be done by the RP repeatedly requesting associations, authentication, or verification of a signature.

The potentially most severe attack is during the association phase as each message requires the OP to execute a discrete exponentiation. Since the RP has the ability to specify modulus and generator per message, an attacker can even force the OP to perform this exponentiation in real time prior to responding for each message.

While this could be particularly harmful, OpenID Providers can easily use generic IP based rate-limiting and banning techniques to help combat these sorts of attacks. OPs can also look at banning requests based on the "openid.realm" and "openid.return_to" values.

15.6. Protocol Variants

The following are known variations in the protocol which may or may not directly affect the security of the use of the protocol. It is imagined that these values could be used in the creation of security profiles for this protocol. The following list of variants are from the perspective of an OpenID Provider.

Number	Variant	Values

1.	Are wildcards allowed in realms?	One of Yes/No
2.	Require prior association? Does the OP require the RP first create an association before requesting authentication?	One of Yes/No
3.	Types of claimed identifiers accepted.	Set of HTTP/HTTPS/XRI
4.	Are self-issued certificates allowed for authentication? This applies to all SSL traffic. If 'no' here, then OP *probably* requires all HTTPS identifiers to chain up to known trust roots, but that's intentionally not implied.	One of Yes/No
5.	Must the XRDS file be signed? Signature on the XRDS as per XMLDSIG. Keying material not specified, since the RP ultimately needs to make own decision whether to trust keys used for such signature.	One of Yes/No
6.	Must the XRDS file be retrieved over secure channel? This does not imply SSL?	One of Yes/No
7.	What types of session types can be used when creating associations?	Set of no-encryption/DH-SHA1/DH-SHA256
8.	Must the RP have an XRDS document?	One of Yes/No
9.	What association types the OP agrees to use for signatures?	Set of HMAC-SHA1/HMAC-SHA256
10.	Must the association request take place over secure channel?	One of Yes/No

Identified security variants.

Appendix A. Examples

Non-normative

Appendix A.1. Normalization

See section 6 of [RFC3986] (Berners-Lee, T., “Uniform Resource Identifiers (URI): Generic Syntax,” .) for textual URL normalization details and more examples.

User's Input	Identifier	Type	Discussion
example.com	http://example.com/	URL	A URI with a missing scheme is normalized to a http URI
http://example.com	http://example.com/	URL	An empty path component is normalized to a slash
https://example.com/	https://example.com/	URL	https URIs remain https URIs
http://example.com/user	http://example.com/user	URL	No trailing slash is added to non-empty path components
http://example.com/user/	http://example.com/user/	URL	Trailing slashes are preserved on non-empty path components
http://example.com/	http://example.com/	URL	Trailing slashes are preserved when the path is empty
=example	=example	XRI	Normalized XRIs start with a global context symbol
xri://=example	=example	XRI	Normalized XRIs start with a global context symbol

User's Input to Identifier Normalization

Appendix A.2. OP-Local Identifiers

An end user wants to use "http://www.example.com/" as their Claimed Identifier. The end user has an account with Example Provider, which functions as an OpenID Provider. The end user's OP-Local Identifier is "https://exampleuser.exampleprovider.com/".

In this scenario, with the proper configuration of Yadis or HTML-Based Discovery (see Section 7.3 (Discovery) and Appendix A.3 (XRDS) below), a Relying Party will discover the following information about the end user:

Claimed Identifier
: http://www.example.com/

OP-Local Identifier
: https://exampleuser.exampleprovider.com/

Appendix A.3. XRDS

For an end user to use "http://www.example.com/" as their Identifier, but have Relying Parties actually verify "https://exampleuser.exampleprovider.com/" with the OP Endpoint URL "https://www.exampleprovider.com/endpoint/", the following XML snippet should be present in the final XRD element in the XRDS file when discovery is preformed on "http://www.example.com/":

```
<Service xmlns="xri://$xrd*($v*2.0)">
  <Type>http://specs.openid.net/auth/2.0/signon</Type>
  <URI>https://www.exampleprovider.com/endpoint/</URI>
  <LocalID>https://exampleuser.exampleprovider.com/</LocalID>
</Service>
```

Appendix A.4. HTML Identifier Markup

To use "http://www.example.com/" as their Identifier, but have Relying Parties actually verify "http://exampleuser.livejournal.com/" with the OpenID Provider located at "http://www.livejournal.com/openid/server.bml", the following markup should be present in the <head> of the HTML document located by the identifier URL:

```
<link rel="openid2.provider openid.server"
    href="http://www.livejournal.com/openid/server.bml"/>
<link rel="openid2.local_id openid.delegate"
    href="http://exampleuser.livejournal.com/"/>
```

Appendix A.5. XRI CanonicalID

For example, if the XRI i-names =example and =exmpl both yield an XRDS document with the CanonicalID xri://(example)!1234 then those Identifiers should be treated as equivalent. For applications with user accounts, the persistent Canonical ID xri://(example)!1234 should be used the primary key for the account. Although the i-names =example and =exmpl may also be stored for reference as display names, they are reassignable identifiers and should not be used as persistent keys.

Appendix B. Diffie-Hellman Key Exchange Default Value

This is a confirmed-prime number, used as the default modulus for Diffie-Hellman Key Exchange. In hexadecimal:

```
DCF93A0B883972EC0E19989AC5A2CE310E1D37717E8D9571BB7623731866E61E
F75A2E27898B057F9891C2E27A639C3F29B60814581CD3B2CA3986D268370557
7D45C2E7E52DC81C7A171876E5CEA74B1448BFDFAF18828EFD2519F14E45E382
6634AF1949E5B535CC829A483B8A76223E5D490A257F05BDFF16F2FB22C583AB
```

Appendix C. Acknowledgements

The OpenID Community would like to thank the following people for the work they've done in the drafting and editing of this specification. If you want to know the nitty gritty of who actually wrote what, feel free to look at our SVN repository or even use "svn blame". :) http://openid.net/svn/specifications/authentication/2.0/

Barry Ferg (barry@sxip.com)

Brad Fitzpatrick (brad@danga.com) <author>

Carl Howells (chowells@janrain.com)

David Recordon (david@sixapart.com) <author/editor>

Dick Hardt (dick@sxip.com) <author>

Drummond Reed (drummond.reed@cordance.net)

Hans Granqvist (hgranqvist@verisign.com)

Johannes Ernst (jernst@netmesh.us)

Johnny Bufu (johnny@sxip.com) <editor>

Josh Hoyt (josh@janrain.com) <author/editor>

Kevin Turner (kevin@janrain.com)

Marius Scurtescu (marius@sxip.com)

Martin Atkins (mart@degeneration.co.uk)

Mike Glover (mpg4@janrain.com)

16. Normative References

[FIPS180-2] U.S. Department of Commerce and National Institute of Standards and Technology, “Secure Hash Signature Standard,” FIPS 180-2.

Defines Secure Hash Algorithm 256 (SHA256)

[HTML401] W3C, “HTML 4.01 Specification.”

[RFC1750] Eastlake, D., Crocker, S., and J. Schiller, “Randomness Recommendations for Security,” RFC 1750.

[RFC2104] Krawczyk, H., Bellare, M., and R. Canetti, “HMAC: Keyed-Hashing for Message Authentication,” RFC 2104.

[RFC2119] Bradner, B., “Key words for use in RFCs to Indicate Requirement Levels,” RFC 2119.

[RFC2616] Fielding, R., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P., and T. Berners-Lee, “Hypertext Transfer Protocol -- HTTP/1.1,” RFC 2616.

[RFC2631] Rescorla, E., “Diffie-Hellman Key Agreement Method,” RFC 2631.

[RFC3174] Eastlake, D. and P. Jones, “US Secure Hash Algorithm 1 (SHA1),” RFC 3174.

[RFC3275] Eastlake 3rd, D., Reagle Jr., J., and D. Solo, “(Extensible Markup Language) XML-Signature Syntax and Processing,” RFC 3275.

[RFC3339] Klyne, G. and C. Newman, “Date and Time on the Internet: Timestamps,” RFC 3339.

[RFC3548] Josefsson, S., “The Base16, Base32, and Base64 Data Encodings,” RFC 3548.

[RFC3629] Yergeau, F., “UTF-8, a transformation format of Unicode and ISO 10646,” RFC 3629.

[RFC3986] Berners-Lee, T., “Uniform Resource Identifiers (URI): Generic Syntax,” RFC 3986.

[XRI_Resolution_2.0] Wachob, G., Reed, D., Chasen, L., Tan, W., and S. Churchill, “Extensible Resource Identifier (XRI) Resolution V2.0 - Committee Draft 02” (HTML, PDF).

[XRI_Syntax_2.0] Reed, D. and D. McAlpin, “Extensible Resource Identifier (XRI) Syntax V2.0” (HTML, PDF).

[Yadis] Miller, J., “Yadis Specification 1.0” (PDF, ODT).

Author's Address

specs@openid.net