	D. Recordon
	B. Fitzpatrick
	May 2006

OpenID Authentication 1.1

Abstract

OpenID Authentication provides a way to prove that an End User owns an Identity URL. It does this without passing around their password, email address, or anything they don't want it to.

OpenID is completely decentralized meaning that anyone can choose to be a Consumer or Identity Provider without having to register or be approved by any central authority. End User's can pick which Identity Provider they wish to use and preserve their Identity as they move between Providers.

While nothing in the protocol requires JavaScript or modern browsers, the authentication scheme plays nicely with "AJAX"-style setups, so an End User can prove their Identity to a Consumer without having to leave the page they are on.

The OpenID Authentication specification does not provide any mechanism to exchange profile information, though Consumers of an Identity can learn more about an End User from any public, semantically interesting documents linked thereunder (FOAF, RSS, Atom, vCARD, etc.). Extensions are being built on top of the foundation created by OpenID Authentication to provide mechanisms to exchange profile information.

Table of Contents

---

1. Requirements Notation

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].

---

2. Terminology

End User:
: The actual human user who wants to prove their Identity to a Consumer.

Identifier:
: An Identifier is just a URL. The whole flow of the OpenID Authentication protocol is about proving that an End User is, owns, a URL.

Claimed Identifier:
: An Identifier that the End User says they own, though that has not yet been verified by the Consumer.

Verified Identifier:
: An Identifier that the End User has proven to a Consumer that they own.

Consumer:
: A web service that wants proof that the End User owns the Claimed Identifier.

Identity Provider:
: Also called "IdP" or "Server". This is the OpenID Authentication server that a Consumer contacts for cryptographic proof that the End User owns the Claimed Identifier.
: How the End User authenticates to their Identity Provider is outside of the scope of OpenID Authenticaiton.

User-Agent:

The End User's web browser. No special plug-ins or JavaScript required.

3. Overview

3.1. Transforming a HTML Document Into an Identifier

In order for a Consumer to know the Identity Provider authoritative for an Identifier, the End User must add markup to the HEAD section of the HTML document located at their URL. The host of the HTML document is NOT REQUIRED to also be the End User's Identity Provider; the Identifier URL and Identity Provider can be fully decoupled services.

To use http://example.com/ as the End User's Identifier http://openid.example.com as their Identity Provider, the following tag would be added to the HEAD section of the HTML document returned when fetching their Identifier URL.

<link rel="openid.server" href="http://openid.example.com/">

3.1.1. Delegating Authentication

If the End User's host is not capable of running an Identity Provider, or the End User wishes to use one running on a different host, they will need to delegate their authentication. For example, if they want to use their website, http://www.example.com/, as their Identifier, but don't have the means, or desire, to run an Identity Provider.

If they have a LiveJournal account (say, user "exampleuser"), and know that LiveJournal provides an OpenID Identity Provider and that it'll assert that they control the Identifier http://exampleuser.livejournal.com/ they would be able to delegate their authentication to LiveJournal's Identity Provider..

So, to use www.example.com as their Identifier, but have Consumers actually verify http://exampleuser.livejournal.com/ with the Identity Provider located at http://www.livejournal.com/openid/server.bml, they'd add the following tags to the HEAD section of the HTML document returned when fetching their Identifier URL.

<link rel="openid.server" href="http://www.livejournal.com/openid/server.bml">

<link rel="openid.delegate" href="http://exampleuser.livejournal.com/">

Now, when a Consumer sees that, it'll talk to http://www.livejournal.com/openid/server.bml and ask if the End User is exampleuser.livejournal.com, never mentioning www.example.com anywhere on the wire.

The main advantage of this is that an End User can keep their Identifier over many years, even as services come and go; they'll just keep changing who they delegate to.

3.1.2. Important Notes

- The declared openid.server URL MAY contain existing query parameters and they MUST be properly preserved when appending extra query parameters. For example, not adding a second question mark if one already exists.
- The openid.server and openid.delegate URLs MUST be absolute URLs. Consumers MUST NOT attempt to resolve relative URLs.
- The openid.server and openid.delegate 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 %xx mechanism as described in [RFC2396] .

3.2. Submitting a Claimed Identifier

Continuing this example, the End User visits a Consumer site which supports OpenID Authentication. The Consumer presents the End User with a form field for them to enter their Identifier URL.

For Example:

```
-----------------------------------
|[logo]example.com               | [Login Button]
-----------------------------------
```

3.2.1. Important Notes

- It is RECOMMENDED that every Consumer place the OpenID logo at the beginning of the form field where the End User enters their Identifier URL.
- The End User is NOT REQUIRED to prefix their Identifier URL with "http://" or postfix it with a trailing slash. Consumers MUST canonicalize the Identifier URL, following redirects, and note the final URL. The final, canonicalized URL is the End User's Identifier.
- It is RECOMMENDED that the form field be named "openid_url" so User-Agent's will auto-complete the End User's Identifier URL in the same way the eCommerce world tends to use conventions like "address1" and "address2".

3.3. Consumer Site Fetches the Identifier URL

Now the Consumer site fetchs the document located at the End User's Claimed Identifier. The Consumer then parses the HEAD section for the "openid.server" and the optional "openid.delegate" declarations.

3.3.1. Important Notes

- The End User could be malicious and try to make the Consumer connect to an internal network, tarpit, etc. It is RECOMMENDED that Consumers use a paranoid HTTP library like LWPx::ParanoidAgent that protects against these sorts of attacks.
- Consumers MUST implement support for Delegation (Delegating Authentication).

3.4. Smart vs Dumb Mode

OpenID Authentication supports both a "smart mode" and "dumb mode" to accomodate Consumers of differing capabilities. A smart Consumer does a little more work at the beginning to save itself work later, but requires local caching of state information. A dumb Consumer is completely stateless, but requires extra an HTTP request.

3.4.1. Important Notes for Smart Mode

- It's RECOMMENDED that a Consumer first submit an associate request (associate) to the End User's Identity Provider and request a shared secret if the Consumer does not already have one cached. This shared secret SHOULD be used as the HMAC-SHA1 key in future identity check requests until it expires.
- The shared secret can be exchanged either in plain-text or encrypted with a Diffie-Hellman-negotiated secret. Note that if Diffie-Hellman is used, it's only used in the associate mode. The checkid_immediate (checkid_immediate) and checkid_setup (checkid_setup) modes assume the Consumer already has a shared secret, regardless of how it got it.

3.5. Consumer Verifies the Identifier

The Consumer now constructs a URL to the Identity Provider's checkid_immediate (checkid_immediate) (or checkid_setup (checkid_setup)) URLs and sends the User-Agent to it.

By sending the User-Agent there, the End User's cookies and whatever other login credentials are sent back to their trusted Identity Provider. The Identity Provider does its work, appends its response onto the supplied openid.return_to URL, and sends the User-Agent back to the Consumer.

4. Modes

4.1. associate

- Description: Establish a shared secret between Consumer and Identity Provider.
- HTTP method: POST
- Flow: Consumer -> IdP -> Consumer

4.1.1. Request Parameters

- openid.mode

  Value: "associate"

- openid.assoc_type

  Value: Preferred association type

  Default: "HMAC-SHA1"

  Note: Optional; Currently only one value.

- openid.session_type

  Value: Blank or "DH-SHA1"

  Default: Blank. (cleartext)

  Note: It is RECOMMENDED that DH-SHA1 mode is used to encrypt the shared secret.

- openid.dh_modulus

  Value: base64(btwoc(p))

  Note: See Appendix A.1 (Diffie-Hellman P Value) for default p value.

- openid.dh_gen

  Value: base64(btwoc(g))

  Default: g = 2

Note: Only if using DH-SHA1 session_type. Should be specified if openid.dh_modulus is specified.

- openid.dh_consumer_public

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

  Note: REQUIRED if using DH-SHA1 session_type.

4.1.2. Response Parameters

Response format: Key-Value Pairs

- assoc_type

  Value: The association type for the returned handle.

  Note: The only current mode is HMAC-SHA1, and all Consumers MUST support it. When caching, the Consumer MUST map an assoc_handle to both its secret and its assoc_type.

- assoc_handle

  Value: The association handle to be provided in future transactions.

  Note: Consumers MUST NOT reuse this association handle after the corresponding expires_in value.

- expires_in

  Value: The number of seconds this association handle is good for in base10 ASCII.

- session_type

  Value: The encryption mode that the Provider chose. MAY be blank, absent, or "DH-SHA1".

- dh_server_public

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

  Description: The Provider's Diffie-Hellman public key (Rescorla, E., “Diffie-Hellman Key Agreement Method,” .) [RFC2631], if using DH-SHA1.

- enc_mac_key

Value: base64(SHA1(btwoc(g ^ (xy) mod p)) XOR secret(assoc_handle))

Description: The encrypted shared secret, if using DH-SHA1.

- mac_key

  Value: base64(secret(assoc_handle))

  Description: The plaintext shared secret, if not using DH-SHA1.

4.1.3. Extra Notes

- A Consumer can ask a server for DH-SHA1 encryption and get back a plaintext secret. If this troubles you, don't use the handle and instead use dumb mode with that Identity Provider.

  If somebody sniffed the plaintext secret, it won't matter, since you'll never accept queries using that association handle. If the Identity Provider can't do DH-SHA1, it's probably limited in some way, but using dumb mode is still safe, if not a little slower.
- If the Identity Provider chooses the server private key 1 <= y < p-1. The shared DH-SHA1 secret is thus g ^ xy mod p = (g ^ x) ^ y mod p = (g ^ y) ^ x mod p. For more information, read the Crypt::DH docs.
- The underlying mac_key MUST be the same length as the output of H, the hash function - in this instance, 160 bits (20 bytes) for DH-SHA1.
- If the Provider does not support DH-SHA1, they WILL ignore the DH-SHA1 fields in the request and reply exactly as to a non-DH-SHA1 request.
- When using DH-SHA1, the resulting key SHOULD be treated as a binary string.
- Most integers are represented in big-endian signed two's complement, Base64 encoded. In other words, btwoc is a function that takes a bigint and returns its shortest big-endian two's complement notation

---

4.2. checkid_immediate

- Description: Ask an Identity Provider if a End User owns the Claimed Identifier, getting back an immediate "yes" or "can't say" answer.
- HTTP method: GET
- Flow: Consumer -> User-Agent -> IdP -> User-Agent -> Consumer

---

4.2.1. Request Parameters

- openid.mode

Value: "checkid_immediate"

- openid.identity

  Value: Claimed Identifier

- openid.assoc_handle

  Value: The assoc_handle from the associate request.

  Note: Optional; Consumer MUST use check_authentication if an association handle isn't provided or the Identity Provider feels it is invalid.

- openid.return_to

  Value: URL where the Provider SHOULD return the User-Agent back to.

- openid.trust_root

  Value: URL the Provider SHALL ask the End User to trust.

  Default: return_to URL

  Optional; the URL which the End User SHALL actually see to approve.

4.2.2. Response Parameters

Response format: query string arguments

4.2.2.1. Always Sent

- openid.mode

  Value: "id_res"

4.2.2.2. Sent on Failed Assertion

- openid.user_setup_url

  Value: URL to redirect User-Agent to so the End User can do whatever's necessary to fulfill the assertion.

4.2.2.3. Sent on Positive Assertion

- openid.identity

  Value: Verified Identifier

- openid.assoc_handle

  Value: Opaque association handle being used to find the HMAC key for the signature.

- openid.return_to

  Value: Verbatim copy of the return_to URL parameter sent in the request, before the Provider modified it.

- openid.signed

  Value: Comma-seperated list of signed fields.

  Note: Fields without the "openid." prefix that the signature covers. For example, "mode,identity,return_to".

- openid.sig

  Value: base64(HMAC(secret(assoc_handle), token_contents)

  Note: Where token_contents is a key-value format string of all the signed keys and values in this response. They MUST be in the same order as listed in the openid.signed field. Consumer SHALL recreate the token_contents string prior to checking the signature. See Appendix D (Limits).

- openid.invalidate_handle

  Value: Optional; The association handle sent in the request if the Provider did not accept or recognize it.

4.2.3. Extra Notes

- This mode is commonly used for "AJAX"-style setups. The more classic mode to check a Claimed Identifier is checkid_setup (checkid_setup).
- An Identity Provider SHOULD only assert URLs that it manages/produces directly. If a End User wants to assert other URLs outside of that Identity Provider's realm, they MUST use delegation (Delegating Authentication).

- The openid.return_to URL provided MAY contain an existing query string, and the Provider MUST preserve it when appending the response parameters. OpenID Consumer's SHOULD add a self-signed nonce with Consumer-local timestamp in the openid.return_to URL parameters to prevent replay attacks. Details of that are left up to the Consumer.

  However, because the openid.return_to URL is signed by the Idenity Provide, a Consumer can make sure outside parties haven't sent id_res responses with mismatching openid.return_to URLs and signatures.
- If the Identity Provider didn't accept/recognize the provided assoc_handle for whatever reason, it'll choose its own to use, and copy the one provided back into openid.invalidate_handle, to tell the Consumer to stop using it. The Consumer SHOULD then send it along in a check_authentication (check_authentication) request to verify it actually is no longer valid.
- If the Identifier assertion fails, the Identity Provider provides the openid.user_setup_url for where the End User can do whatever's necessary to fulfill the assertion, be it login, setup permissions, etc. The server SHOULD return a URL which doesn't imply anything about what's needed, so the Consumer is left in the dark about why the assertion failed.

  The Identity Provider handling SHOULD eventually return the End User to the openid.return_to URL, acting like a checkid_setup response, with either a "id_res" or "cancel" mode.
- The openid.return_to URL MUST descend from the openid.trust_root, or the Identity Provider SHOULD return an error. Namely, the URL scheme and port MUST match. The path, if present, MUST be equal to or below the value of openid.trust_root, and the domains on both MUST match, or, the openid.trust_root value contain a wildcard like http://*.example.com. The wildcard SHALL only be at the beginning. It is RECOMMENDED Identity Provider's protect their End Users from requests for things like http://*.com/ or http://*.co.uk/.
- In the response, the Identity Provider's signature MUST cover openid.identity and openid.return_to.

4.3. checkid_setup

- Description: Ask an Identity Provider if a End User owns the Claimed Identifier, but be willing to wait for the reply. The Consumer will pass the User-Agent to the Identity Provider for a short period of time which will return either a "yes" or "cancel" answer.
- HTTP method: GET
- Flow: Consumer -> User-Agent -> [IdP -> User-Agent ->]+ Consumer

---

4.3.1. Request Parameters

- openid.mode

  Value: "checkid_setup"

- openid.identity

  Value: Claimed Identifier

- openid.assoc_handle

  Value: The assoc_handle from the associate request.

  Note: Optional; Consumer MUST use check_authentication if an association handle isn't provided or the Identity Provider feels it is invalid.

- openid.return_to

  Value: URL where the Provider SHOULD return the User-Agent back to.

- openid.trust_root

  Value: URL the Provider SHALL ask the End User to trust.

  Default: return_to URL

  Optional; the URL which the End User SHALL actually see to approve.

4.3.2. Respone Parameters

Response format: query string arguments

4.3.2.1. Always Sent

- openid.mode

  Value: "id_res" or "cancel"

4.3.2.2. Sent on Positive Assertion

- openid.identity

  Value: Verified Identifier

- openid.assoc_handle

Value: Opaque association handle being used to fine the HMAC key for the signature.

- openid.return_to

  Value: Verbatim copy of the return_to URL parameter sent in the request, before the Provider modified it.

- openid.signed

  Value: Comma-seperated list of signed fields.

  Note: Fields without the "openid." prefix that the signature covers. For example, "mode,identity,return_to".

- openid.sig

  Value: base64(HMAC(secret(assoc_handle), token_contents)

  Note: Where token_contents is a key-value format string of all the signed keys and values in this response. They MUST be in the same order as listed in the openid.signed field. Consumer SHALL recreate the token_contents string prior to checking the signature. See Appendix D (Limits).

- openid.invalidate_handle

  Value: Optional; The association handle sent in the request if the Provider did not accept or recognize it.

4.3.3. Extra Notes

- In the response, the Identity Provider's signature MUST cover openid.identity and openid.return_to.
- In a lot of cases, the Consumer won't get a cancel mode; the End User will just quit or press back within their User-Agent. But if it is returned, the Consumer SHOULD return to what it was doing. In the case of a cancel mode, the rest of the response parameters will be absent.

4.4. check_authentication

- Description: Ask an Identity Provider if a message is valid. For dumb, stateless Consumers or when verifying an invalidate_handle response.

  WARNING: Only validates signatures with stateless association handles. Identity Providers MUST NOT ever validate a signature for an association handle whose secret

has been shared with anybody. They MUST differentiate its stateless vs. associated association handles, and only offer check_authentication service on the stateless handles.
- HTTP method: POST
- Flow: Consumer -> IdP -> Consumer

4.4.1. Request Parameters

- openid.mode

  Value: "check_authentication"

- openid.assoc_handle

  Value: The association handle from checkid_setup or checkid_immediate response.

- openid.sig

  Value: The signature from the checkid_setup or checkid_immediate request the Consumer wishes to verify.

- openid.signed

  Value: The list of signed fields from the checkid_setup or checkid_immediate request the Consumer wishes to verify the signature of.

- openid.*

  Value: The Consumer MUST send all the openid.* response parameters from the openid.signed list which they'd previously gotten back from a checkid_setup or checkid_immediate request, with their values being exactly what were returned from the Provider.

- openid.invalidate_handle

  Value: Optional; association handle returned via invalidate_handle.

4.4.2. Response Parameters

Response format: Key-Value Pairs

- openid.mode

  Value: "id_res"

- is_valid

  Value: "true" or "false"

  Description: Boolean; whether the signature is valid.

- invalidate_handle

  Value: opaque association handle

  Description: If present, the Consumer SHOULD uncache the returned association handle.

4.4.3. Extra Notes

- Identity Providers MUST implement this mode for error recovery and dumb Consumers, which can't keep state locally, but it's RECOMMENDED that it is used as little as possible, as it shouldn't be necessary most the time. It's good for debugging, though, as you develop your Consumer library.
- If you got an invalidate_handle response during a checkid_setup or checkid_immediate request, that means the Identity Provider didn't recognize the association handle, maybe it lost it, and had to pick its own.

  This means the Consumer will have to fallback to dumb mode, since you don't have the shared secret which the Identity Provider is using. While doing this check_authentication request, also send along the invalidate_handle response from the Identity Provider and it'll be checked to see if it actually is missing/bogus.
- When verifying the signature using openid.* query values, the openid.mode value must be changed to "id_res".

---

5. Security Considerations

- While the OpenID Authentication protocol often refers to using HTTP, HTTPS can be used for additional security. It is RECOMMENDED it is used during the associate mode (associate) and helps to protect against man in the middle, DNS, and some phishing attacks.
- Consumers SHOULD NOT use IFrames or popup's when requesting an End User login via OpenID.

Appendix A. Default Values

Appendix A.1. Diffie-Hellman P Value

155172898181473697471232257763715 5\ 39915724801966915404479707795314 05\
76293785419175806512274236981889 93\ 72781615264663143856159582568818 88\
89951272158842675419950341258706 55\ 65498035801048705376814767265132 55\
74704076585747929129157233451064 32\ 45094715007229621094194349783925 98\
47603755949858482533593058543963 8443

Appendix B. Error Responses

This section pertains to protocol/run-time errors, not authentication errors. Authentication errors are defined in the protocol.

- No error codes have been defined; just unstructured natural language error text.
- If it's a GET request with bad arguments, but a valid openid.return_to URL, the Identity Provider SHALL redirect the User-Agent with openid.mode=error and openid.error=Error+Text set.
- If it's a GET request with bad arguments, and no valid openid.return_to URL, the Identity Provider SHALL return a "400 Bad Request" with any content-type and error message it wants.
- If it's a GET request with no arguments, the Identity Provider SHALL show a 200 text/html error saying "This is an OpenID server endpoint. For more information, see http://openid.net/".
- If it's a POST request with bad/no arguments, the Identity Provider SHALL return a 400 Bad Eequest with the Key-Value response format containing a single key "error" with the natural language text. The Identity Provider can add any additonal keys it wishes in this case.

Appendix C. Key-Value Format

Lines of:

- some_key:some value
- There MUST NOT be a space before or after the colon.
- Newline characters MUST be Unix-style, just ASCII character 10 ("\n").
- Newlines MUST BE at end of each line as well as between lines.
- MIME type is unspecified, but text/plain is RECOMMENDED.
- Character encoding MUST BE UTF-8.

Appendix D. Limits

- Identifier URL: 255 max bytes

- Identity Provider URL: 2047 max bytes, after Consumer-added URL arguments. The raw endpoint URL SHOULD be kept well below this.
- return_to URL: 2047 max bytes, after Identity Provider added URL arguments. The raw return_to URL SHOULD be kept well below this.
- assoc_handle: 255 characters or less, and consist only of ASCII characters in the range 33-126 inclusive (ie printable non-whitespace characters).

Appendix E. Misc

- Timestamps must be in w3c format, and must be in the UTC timezone, indicated with a "Z". For example: 2005-05-15T17:11:51Z

---

6. Normative References

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

[RFC2396] Berners-Lee, T., “Uniform Resource Identifiers (URI): Generic Syntax.”

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

Authors' Addresses

David Recordon
Email: drecordon@verisign.com

Brad Fitzpatrick
Email: brad@danga.com