OpenID Authentication Security

Erik Lagercrantz and Patrik Sternudd

Uppsala, May 17 2009

1 Abstract

This documents gives an introduction to OpenID, which is a system for centralised online authentication. In particular, we describe the protocol used for authentication. The description is followed by a discussion of its security, with focus on cryptographic soundness. Finally, we provide a short discussion on what we feel the the OpenID system is suitable for.

2 Scope

The central question of this paper is "What can we trust OpenID for? Before this question can be answered, we need to be aware that there are different trust issues that must be considered. For example:

- Can we trust the authentication protocol itself?
- Can we trust the security of the authentication server (the OpenID provider)?
- Can we trust the strength of the authentication methods used by the OpenID system?
- Do we trust the organisation in charge of the OpenID server?

While all of the above questions are important when considering whether to allow OpenID or not for a particular service, this document will only focus on the authentication protocol. This is perhaps the most fundamental question, because if the protocol is insecure, all the other questions lose meaning. However, if we simply do not trust a particular provider, we can always choose another one, or set up our own.

3 Introduction to OpenID

This section contain a brief introduction to OpenID. Readers who already are familiar with the concept may skip to the next chapter.

The main purpose of OpenID is to reduce the amount of usernames and passwords on the Internet (in particular on websites). Today, most sites where it is possible to publish information of any kind (e.g. blogs, blog comments, forums, etc.) or retrieve privileged services (member sites, bonus programs) requires an account. To authenticate, you have to choose a username and password combination (sometimes the username is merely an email address). The problem, of course, is that there is a multitude of such sites on the Internet. Thus, we have to remember a lot of different passwords (or more likely reuse them). The goal of OpenID is to allow individuals to have one username and password at an OpenID provider. This id is then given to the website, which delegates authentication to the OpenID server.

It is important to know that OpenID only handles authentication. It does not provide authorization of any kind – if that is wanted, the web site would have to match the authenticated user with an internal authorization profile.

Also, OpenID does not provide the web site with information beyond the user id and authentication results. That means that the user will still have to manually provide the required personal details (such as email- or postal address) when establishing an account with a web site. There are extensions to OpenID that do provide such services, but we will not cover those in this document.

A final important point is that the web site requiring authentication will *never* see the authentication credentials; only the OpenID server does.

4 Terminology

User Agent (UA)	*A client program (usually a web browser) used by the end user.*
Relying Party (RP)	*A service provider (usually a web site) that needs the end user to be authenticated.*
OpenID Provider (OP)	*The server performing authentication and storing the information required to do so.*
Assertion	*A message containing information about authentication results (positive or negative).*

5 How it works

The OpenID concept is based on URI:s. An OpenID-configured web site requiring a user to authenticate will ask the user to provide an URI instead of a traditional username. When the user asks to be logged in, the web site (or *Relying Party* in OpenID terminology) redirects the user to the OpenID provider of the supplied URI. The provider can authenticate the user in any way it wants (such as by asking for a password), and then return the user to the relying party while passing along the authentication result.

6 OpenID authentication

The latest version of the authentication protocol is 2.0, which can be found at:

http://openid.net/specs/openid-authentication-2_0.txt

All communication is sent over HTTP (or HTTPS), with UTF-8 encoding.

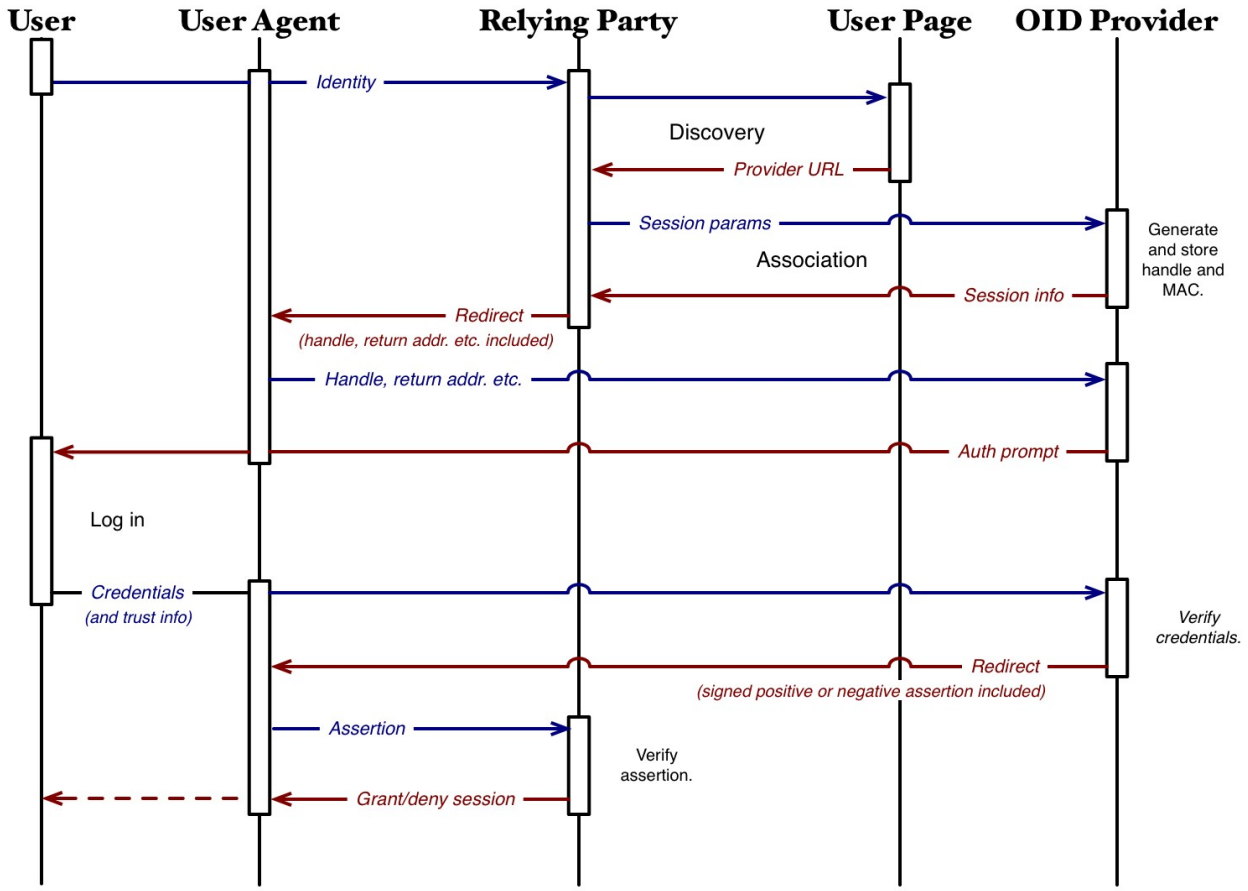
6.1 The protocol (slightly simplified)

Only the steps required for authentication are described. Thus, actions for normalising URI:s and handling extensions are omitted.

6.1.1 Authentication actions

1. 1.UA -> RP: Present URI
2. 2.RP <-> OP: [Optional] Association (negotiate a shared key for MAC)
3. 3.RP -> UA: Redirect UA to OP
4. 4.UA -> OP: Submit credentials for authentication
5. 5.OP -> UA: Return UA to RP (sending along signed authentication results including a nonce)
6. 6.UA -> RP: Provide the authentication assertion signed by the OP
7. 7.RP <-> OP: [Optional] Ask for verification of authentication results (if step 2 was skipped)
8. 8.RP -> UA: Allow or deny access to the protected resource

6.1.2 Authentication flowchart



The above diagram provides a slightly more detailed description of the authentication process. Note in particular the discovery phase in the beginning, in which the RP sends a request to get the resource pointed at by the URL that the user provided as its identity. The resource can be an HTML document containing a link to the actual OP service. Note that the process described in the diagram corresponds to the sequence that is used when the RP is stateful. In the stateless version, the association step is skipped, and the OP is contacted for a direct verification at the end instead.

6.1.3 Shared key negotiation

The shared key is used in the MAC algorithms. The purpose is to be able to sign and verify authentication assertions in subsequent messages. The specification recommends that shared key negotiation is performed if the participating parties are able to do so. This step is called association.

If the parties are unable to perform key agreement, for example because the RP cannot keep state between requests, they will have to do more communication in the last step (verifying authentication).

1. 1.RP -> OP: Preferred MAC and session protection types, and other parameters
    - Possible MAC types are *HMAC-SHA1* and *HMAC-SHA256.*
    - Possible session protection types are *no-encryption*, *DH-SHA1*, and *DH-SHA256*.
    - No-encryption *must not* be used unless transport security is available (e.g. TLS).
    - When using one of the DH (Diffie-Hellman) options, the parameters *modulus, generator,* and *public key* are included.

2. OP -> RP: Session response, including
    - Common response parameters:
        - Association handle
        - Session type
        - Association type
        - Expiration (lifetime)
    - **AND**
        - Unencrypted response parameters
            - The MAC key
        - OR DH response parameters
            - The public key of the OP
            - The MAC key, encrypted with Diffie-Hellman
        - OR Unsuccessful response parameters
            - Error messages and codes, indicating the problem (e.g. the OP does not support the preferred algorithms)

Notes:

1. If the MAC key is protected by Diffie-Hellman, it is prepared as follows, before being sent over the network:

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

    - `g`, `p`, `xa`, and `xb` are previously agreed DH parameters. `btwoc` is an integer encoding function (see Section 4.2 for more details).
    - The following condition must hold (Section 8.4.2):

> *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 [..]*

6.2 Security measures

6.2.1 Nonce usage

An important part for protection against replay attacks is the nonce. It is described in section 10.1 of the specification. In short, it is an unique string less than 255 characters, starting with current time (formatted as per section 5.6 in RFC3339, and in UTC with no fractal seconds), followed (if necessary for uniqueness) by printable, non-whitespace characters. An example would be *2009-05-16T11:47:11ZFOO.*

To quote from Section 11.3 (a *positive assertion* is when the OP informs the RP that the authentication attempt was successful):

> *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.*

6.2.2 Signed messages

As previously stated, using MAC for message authentication is recommended. Furthermore, HMAC-SHA256 is preferred over HMAC-SHA1. The specification notes the importance of choosing a random key,

and refers to a RFC that provide guidance for secure random numbers (RFC 1750). When MAC is used, the following fields in the positive assertion must be signed:

- *op_endpoint:* - Authentication endpoint
- *return_to:* - The return address (to the RP)
- *response_nonce:* - Nonce (previously described)
- *assoc_handle:* - Session handle
- *claimed_id* and i*dentity* (if available): - The authenticated identity

6.2.3 Transport security

As previously stated, if no session encryption with Diffie-Hellman is performed while negotiating the MAC key, the specification instead require that a transport encryption such as TLS or SSL is used.

7 Discussion

7.1 Possible attacks

Man in the middle attacks are a real threat when the OpenID protocol is improperly used. In particular, it is trivial to control the outcome of any authentication attempt if one is able to modify the communication between the relying party and the provider without being detected. Fooling the RP into connecting to a malicious OP can be done by manipulating DNS information, the information in the discovery document, or by simply impersonating the original OP after intercepting the traffic from the RP. There is nothing in the OpenID protocol itself that prevents the RP from accepting information from such a malicious OP. The shared key that is exchanged in the association step serves only to verify that the signed authentication assertions are delivered by the OP that was first contacted, it does not enable the RP to verify that it was in fact talking to the right OP in the first place.

The implications of the above is that real security can only be achieved if the RP uses some means outside of the OpenID protocol to verify the identity of the OP and the authenticity of critical messages. This can be achieved by requiring that all direct communication between the RP and the OP is protected with TLS, and that the OP has a certificate signed by a trusted party. It is sufficient to protect the direct communication (discovery, association and direct verification messages) in this manner. Other messages (e.g. authentication assertions) are sent indirectly between the OP and the RP via the UA, but critical parts of these messages are signed by the shared key to detect modifications done by the UA, and thus need not be protected with TLS.

Even when not using TLS, the protocol may be considered secure enough in some cases. Once the RP has negotiated an associated with an OP, they can both store the shared key and reuse it in the future. This means that in future authentication attempts, all critical messages are signed with the MAC that was determined during association, meaning that a man in the middle attack is only possible if the MAC can be broken. It is however unclear if any existing OpenID implementations actually enforce reuse of the shared key, or if a malicious OP could get an old shared key to be discarded and generate a new one. In any case, TLS is the appropriate way to protect against man in the middle attacks, since it also protects on first contact.

It goes without saying that communication between the UA and the OP should be encrypted if authentication is performed by transmitting a password. How to perform this is outside the scope of the OpenID specification, but the obvious solution is to use TLS also for this communication. If trusted certificates are used and validated this also helps to prevent phishing attacks, where a user is tricked into supplying their authentication credentials on a malicious web site.

7.2 Conclusion – can we trust it?

The OpenID protocol itself can be trusted provided that the MAC can be trusted, and that the communication channel can be trusted such as when using TLS as described in the previous section. However, the security ultimately depends on the authentication methods used by the provider. Some providers employ passwords while others have more advanced solutions. This is not covered by the OpenID specification.

A user must also be able to trust the general security and integrity of its OpenID provider. If a provider is compromised a malicious person will able to impersonate any user of that provider on any OpenID-enabled service.

For this reason, OpenID or any similar system is unlikely to be widely adopted in security-critical areas such as online banking, unless these institutions can find a way to only admit users of secure providers. One solution could be to have only a few officially trusted providers, much like the way that it works for the Swedish e-legitimation. Until then however, OpenID will remain a tool that is used when convenience is considered more important than security.

7.3 Other considerations

Since the Relying Party need to communicate to OpenID providers through HTTP/HTTPS, outbound firewall rules must be opened. This is a security risk in itself, especially when considering that there have been a lot of worms that specifically target web servers.

Another issue that is not directly technical, but rather social, is the question “who would you like to host your identity?”. That company or organisation would be able to put together a pretty good picture of your online habits (which sites are you member of, how often do you log in there, at what times, etc.). This exposure could be limited by using several OpenID providers, but that sort of defeats the whole idea.

8 References

"OpenID Authentication 2.0 – Final", 2007, http://openid.net/specs/openid-authentication-2_0.html (Accessed 2009-06-17)

[RFC 1750] Eastlake, D., Crocker, S. and J. Schiller, "Randomness Requirements for Security", RFC 1750, December 1994.

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

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

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