Liberty ID-FF Bindings and Profiles Specification

Version: 1.2-errata-v2.0

Editors:
Scott Cantor, Internet2, The Ohio State University
John Kemp, IEEE-ISTO
Darryl Champagne, IEEE-ISTO

Contributors:
Robert Aarts, Nokia Corporation
Bronislav Kavsan, RSA Security Inc.
Thomas Wason, IEEE-ISTO

Abstract:

Specification of the Liberty Alliance Project core profiles and bindings. This specification defines the bindings and profiles of the Liberty protocols and messages to HTTP-based communication frameworks. This specification relies on the SAML core framework in SAML Core V1.1 and makes use of adaptations of the SAML profiles in SAML Bindings V1.1.

Filename: draft-liberty-idff-bindings-profiles-1.2-errata-v2.0.pdf

Notice

This document has been prepared by Sponsors of the Liberty Alliance. Permission is hereby granted to use the document solely for the purpose of implementing the Specification. No rights are granted to prepare derivative works of this Specification. Entities seeking permission to reproduce portions of this document for other uses must contact the Liberty Alliance to determine whether an appropriate license for such use is available.

Implementation of certain elements of this document may require licenses under third party intellectual property rights, including without limitation, patent rights. The Sponsors of and any other contributors to the Specification are not, and shall not be held responsible in any manner for identifying or failing to identify any or all such third party intellectual property rights. **This Specification is provided "AS IS", and no participant in the Liberty Alliance makes any warranty of any kind, express or implied, including any implied warranties of merchantability, non-infringement of third party intellectual property rights, and fitness for a particular purpose.** Implementors of this Specification are advised to review the Liberty Alliance Project's website (http://www.projectliberty.org/) for information concerning any Necessary Claims Disclosure Notices that have been received by the Liberty Alliance Management Board.

Liberty Alliance Project
Licensing Administrator
c/o IEEE-ISTO
445 Hoes Lane
Piscataway, NJ 08855-1331, USA
info@projectliberty.org

Contents

1. Introduction

This specification defines the bindings and profiles of the Liberty protocols and messages to HTTP-based communication frameworks. This specification relies on the SAML core framework in [SAMLCore11] and makes use of adaptations of the SAML profiles in [SAMLBind11]. A separate specification, [LibertyProtSchema], is used to define the Liberty protocols and messages used within the profiles. Definitions for Liberty-specific terms can be found in [LibertyGlossary].

1.1. Notation

The key words "MUST," "MUST NOT," "REQUIRED," "SHALL," "SHALL NOT," "SHOULD," "SHOULD NOT," "RECOMMENDED," "MAY," and "OPTIONAL" in this specification are to be interpreted as described in [RFC2119]: "they MUST only be used where it is actually required for interoperation or to limit behavior which has potential for causing harm (e.g., limiting retransmissions)."

These keywords are thus capitalized when used to unambiguously specify requirements over protocol and application features and behavior that affect the interoperability and security of implementations. When these words are not capitalized, they are meant in their natural-language sense.

```
Listings of productions or other normative code appear like this.
```

```
Example code listings appear like this.
```

Note:

Non-normative notes and explanations appear like this.

Conventional XML namespace prefixes are used throughout this specification to stand for their respective namespaces as follows, regardless of whether a namespace declaration is present in the example:

XML Namespace Conventions

- The prefix `lib:` stands for the Liberty namespace `urn:liberty:iff:2003-08`
- The prefix `saml:` stands for the SAML assertion namespace (see [SAMLCore]).
- The prefix `samlp:` stands for the SAML request-response protocol namespace (see [SAMLCore]).
- The prefix `ds:` stands for the W3C XML signature namespace, `http://www.w3.org/2000/09/xmldsig#`
- The prefix `xenc:` stands for the W3C XML encryption namespace, `http://www.w3.org/2001/04/xmlenc#`
- The prefix `SOAP-ENV:` stands for the SOAP 1.1 namespace, `http://schemas.xmlsoap.org/soap/envelope` (see [SOAP1.1]).

Terminology from [RFC2396] is used to describe components of an HTTP URL. An HTTP URL has the following form:

```
<scheme>://<authority><path>?<query>
```

Sections in this document specify certain portions of the `<query>` component of the URL. Ellipses (...) are used to indicate additional, but unspecified, portions of the `<query>` component.

2. Protocol Bindings

The Liberty protocol bindings are defined in this section.

2.1. SOAP Binding for Liberty

The Liberty SOAP binding defines how to use SOAP to send and receive Liberty protocol requests and responses using SOAP 1.1 messages.

Like Liberty, SOAP can be used over multiple underlying transports. This binding has protocol-independent aspects, but REQUIRES the use of SOAP over HTTP.

2.1.1. Protocol-Independent Aspects of the Liberty SOAP Binding

The following sections define aspects of the Liberty SOAP binding that are independent of the underlying protocol, such as HTTP, on which the SOAP messages are transported.

2.1.1.1. Basic Operation

SOAP messages consist of three elements: an envelope, header data, and a message body. Liberty request-response protocol elements MUST be enclosed within the SOAP message body.

SOAP 1.1 also defines an optional data encoding system. This system is not used within the Liberty SOAP binding. This means that SAML messages can be transported using SOAP without re-encoding from the "standard" Liberty schemas to one based on the SOAP encoding.

The specific profile determines the type of messages that can be sent or received. The system model used for Liberty conversations over SOAP may be a simple request-response model, or it may be a more complex interaction that includes HTML forms or other input mechanisms that interact with a Principal.

This Liberty specification defines constraints. Liberty protocol messages MUST be sent as the top level element in the SOAP body. The requester or responder MUST NOT include more than one Liberty protocol message in a single SOAP message. The requester or responder MUST NOT include any additional XML elements in the SOAP body. Additionally, if a SOAP fault code is returned, then no Liberty protocol message may appear in the SOAP body. SOAP faults MUST only be used for signaling non-Liberty-related errors.

[SOAPv1.1] references an early draft of the XML Schema specification including an obsolete namespace. Originators of Liberty SOAP messages SHOULD generate SOAP messages referencing only the final XML schema namespace. Receivers of Liberty SOAP messages MUST be able to process both the XML schema namespace used in [SOAPv1.1] and the final XML schema namespace.

2.1.1.2. SOAP Headers

A Liberty SOAP message MAY contain arbitrary headers added to the SOAP message. This binding does not define any additional SOAP headers.

Liberty SOAP messages MUST NOT require that any headers be understood for correct interpretation of the message.

2.1.1.3. Authentication

Authentication of Liberty messages is OPTIONAL and depends on the environment of use. Authentication protocols available from the underlying substrate protocol MAY be utilized to provide authentication. Section 2.1.2.1 describes authentication in the SOAP-over-HTTP environment.

2.1.1.4. Message Integrity

Message integrity of Liberty messages is OPTIONAL and depends on the environment of use. The security layer in the underlying substrate protocol MAY be used to ensure message integrity. Section 2.1.2.2 describes support for message integrity in the SOAP-over-HTTP environment.

2.1.1.5. Confidentiality

Confidentiality of Liberty messages is OPTIONAL and depends on the environment of use. The security layer in the underlying substrate protocol MAY be used to ensure message confidentiality. Section 2.1.2.3 describes support for confidentiality in the SOAP over HTTP environment.

2.1.2. Use of SOAP over HTTP

This section describes certain specifics of using SOAP over HTTP, including HTTP headers, error reporting, authentication, message integrity and confidentiality.

The HTTP binding for SOAP is described in [SOAPv1.1] §6.0. It requires the use of a SOAPAction header as part of a SOAP HTTP request. Processing of a Liberty message MUST NOT depend on the value of this header. A Liberty message MAY set the value of its SOAPAction header as follows:

```
urn:liberty:soap-action
```

2.1.2.1. Authentication

Liberty SOAP message endpoints MUST implement the following authentication methods:

1. No client or server authentication.
2. HTTP basic client authentication [RFC2617] with and without SSL 3.0 or TLS 1.0.
3. HTTP over SSL 3.0 or TLS 1.0 (see Section 6) server authentication with a server-side certificate.
4. HTTP over SSL 3.0 or TLS 1.0 client authentication with a client-side certificate.

If a message receiver uses SSL 3.0 or TLS 1.0, it MUST use a server-side certificate.

2.1.2.2. Message Integrity

When message integrity needs to be guaranteed, messages MUST be sent with HTTP over SSL 3.0 or TLS1.0 with a server-side certificate.

2.1.2.3. Message Confidentiality

When message confidentiality is required, messages MUST be sent with HTTP over SSL 3.0 or TLS 1.0 with a server-side certificate.

2.1.2.4. Security Considerations

Before deployment in a given profile, each combination of authentication, message integrity and confidentiality mechanisms SHOULD be analyzed for vulnerability in the context of the profile.

[RFC2617] describes possible attacks in the HTTP environment when basic or message-digest authentication schemes are used.

2.1.2.5. Error Reporting

A message receiver that refuses to perform a message exchange SHOULD return a "403 Forbidden" response. In this case, the content of the HTTP body is not significant.

As described in [SOAPv1.1] § 6.2, in the case of a SOAP error while processing a SOAP request, the SOAP HTTP server MUST return a "500 Internal Server Error" response and include a SOAP message in the response with a SOAP fault element. This type of error SHOULD be returned for SOAP-related errors detected before control is passed to the Liberty message processor, or when the SOAP processor reports an internal error (for example, the SOAP XML namespace is incorrect).

In the case of a Liberty processing error, the SOAP HTTP server MUST respond with "200 OK" and include a profile-specified response as the only child of the <SOAP-ENV:Body> element.

2.1.2.6. Example of Message Exchange Using SOAP over HTTP

The following is an example of the SOAP exchange for the single sign-on browser artifact profile requesting an authentication assertion (the left margin white space added for legibility invalidates the signature).

```
POST /authn HTTP/1.1
Host: idp.example.com
Content-type: text/xml
Content-length: nnnn
<soap-env:Envelope
   xmlns:soap-env="http://schemas.xmlsoap.org/soap/envelope/">
   <soap-env:Header/>
   <soap-env:Body>
      <samlp:Request xmlns="urn:oasis:names:tc:SAML:1.0:protocol"
         xmlns:lib="urn:liberty:iff:2003-08"
         xmlns:saml="urn:oasis:names:tc:SAML:1.0:assertion"
         xmlns:samlp="urn:oasis:names:tc:SAML:1.0:protocol"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         IssueInstant="2002-12-12T10:08:56Z"
         MajorVersion="1"
         MinorVersion="1"
         RequestID="e4d71c43-c89a-426b-853e-a2b0c14a5ed8"
         id="ericssonb6dc3636-f2ad-42d1-9427-220f2cf70ec1">
         <ds:Signature xmlns:ds="http://www.w3.org/2000/09/xmldsig#">
            <ds:SignedInfo>
               <ds:CanonicalizationMethod
                  Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#">
               </ds:CanonicalizationMethod>
               <ds:SignatureMethod
                  Algorithm="http://www.w3.org/2000/09/xmldsig#rsa-sha1">
               </ds:SignatureMethod>
               <ds:Reference URI="#ericssonb6dc3636-f2ad-42d1-9427-220f2cf70ec1">
                  <ds:Transforms>
                     <ds:Transform
                        Algorithm="http://www.w3.org/2000/09/xmldsig#enveloped-signature">
                     </ds:Transform>
                     <ds:Transform
                        Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#">
                     </ds:Transform>
                  </ds:Transforms>
                  <ds:DigestMethod
                     Algorithm="http://www.w3.org/2000/09/xmldsig#sha1">
                  </ds:DigestMethod>
                  <ds:DigestValue>+k6TnolGkIPKZlpUQVyok8dwkuE=</ds:DigestValue>
               </ds:Reference>
            </ds:SignedInfo>
            <ds:SignatureValue>
               wXJMVoPO1V1jFnWJPyOWqP5Gqm8A1+/2b5gNzF4L4LMu4yEcRtttLdPPT3bvhwkwHXjL9NuOFumQ
               5YEyiVzlNcjAxX0LfgwutvEdJb748IU4L+8obXPXfqTZLiBK1RbHCRmRvjlPIu22oGCV6EwuiWRv
               OD6Ox9svtSgFJ+iXkZQ
            </ds:SignatureValue>
```

```
            <ds:KeyInfo>
               <ds:X509Data>
                  <ds:X509Certificate>
                     MIIDMTCCApqgAwIBAgIBHDANBgkqhkiG9w0BAQQFADCBlTEL MAkGA1UEBhMCVVMxCzAJBgNVBAcT
                     AlNGMRkwFwYDVQQKExBMaWJlcnR5IEFsb GlhbmNlMRQwEgYDVQQLEwtJT1AgVGV zdGVyczEiMCAG
                     A1UEAxMZTGliZXJ0eSBUZXN0ZXJzIENlcnRpZmllcjEkMCIG CSqGSIb3DQEJARYVcnJvZHJpZ3Vl
                     ekBuZW9zb2wubmV0MB4XDTAyMTIwNDE1N Tg0NFoXDTEyMTIwMTE1NTg0NFowgas xCzAJBgNVBAYT
                     AlVTMQswCQYDVQQHEwJTRjEkMCIGA1UEChMbTGliZXJ0eSBB bGxpYW5jZSBlcmljc3Nvbi1hMSYw
                     JAYDVQQLEx1JT1AgVGVzdGVycyBlcmljc 3Nvbi1hIHNpZ25lcjEXMBUGA1UEAxM OZXJpY3Nzb24t
                     YS5pb3AxKDAmBgkqhkiG9w0BCQEWGXJyb2RyaWd1ZXpAZXJp Y3Nzb24tYS5pb3AwgZ8wDQYJKoZI
                     hvcNAQEBBQADgY0AMIGJAoGBAPUoGYvJx Qc5jzDnJ14TV6TaTbB3fH95ju24Z0y 6HQxm6gXdJSAo
                     Wh7/AIes4UcV09DC2kKS6Vow2YoXt2LIyH9HWH2tEUt1jS/P UeBHEWcW3tFezM6jh5GG5rCuVPZa
                     W9eoGUbFPSzOPFKUAwdHUXSDWufY1KZ93 IxhOBeZgg6VAgMBAAGjeTB3MEoGCWC GSAGG+EIBDQQ9
                     FjtUaGlzIHNpZ25pbmcgY2VydCB3YXMgY3JlYXRlZCBmb3Ig dGVzdGluZy4gRG8gbm90IHRydXN0
                     IGl0LjAJBgNVHRMEAjAAMBEGCWCGSAGG+ EIBAQQEAwIEMDALBgNVHQ8EBAMCBsA wDQYJKoZIhvcN
                     AQEEBQADgYEAR/HSgBpAprQwQVyWDE9pCaiduKv4/W/+hrdp XlVKSr6TIlg4ouDCQJNos7tNuG9Z
                     AbfWtHvCss51N2cfAzfns/DKqxRqcsxzL 5ZUBksPpmsDoboopUv6Xm8RFsi7yB9 AGaVuqObeY/+m
                     70nOu03O+FlMN3U1k2E3rOKXlU1noC0
                  </ds:X509Certificate>
               </ds:X509Data>
            </ds:KeyInfo>
         </ds:Signature>
         <samlp:AssertionArtifact>
            AAM1uXw6+f+jyA/4XuFHqPl7QDvc/LIQL9+t7YQtG1Gwk9bph0Adl+o+
         </samlp:AssertionArtifact>
      </samlp:Request>
   </soap-env:Body>
</soap-env:Envelope>
```

The following is an example of a response, which supplies an assertion containing an authentication statement:

```
HTTP/1.1 200 OK
Content-Type: text/xml
Content-Length: nnnn
<soap-env:Envelope
   xmlns:soap-env="http://schemas.xmlsoap.org/soap/envelope/">
   <soap-env:Header/>
   <soap-env:Body>
      <samlp:Response
         xmlns:samlp="urn:oasis:names:tc:SAML:1.0:protocol"
         InResponseTo="RPCUk2ll+GVz+t1lLURp51oFvJXk"
         IssueInstant="2002-10-31T21:42:13Z" MajorVersion="1" MinorVersion="1"
         Recipient="http://localhost:8080/sp"
         ResponseID="LANWfL2xLybnc+BCwgY+p1/vIVAj">
         <samlp:Status>
            <samlp:StatusCode
               xmlns:qns="urn:oasis:names:tc:SAML:1.0:protocol"
               Value="qns:Success">
            </samlp:StatusCode>
         </samlp:Status>
         <saml:Assertion
            xmlns:saml="urn:oasis:names:tc:SAML:1.0:assertion"
            xmlns:ds="http://www.w3.org/2000/09/xmldsig#"
            xmlns:lib="urn:liberty:iff:2003-08"
            xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
            AssertionID="SqMC8Hs2vJ7Z+t4UiLSmhKOSUO0U"
            InResponseTo="RPCUk2ll+GVz+t1lLURp51oFvJXk"
            IssueInstant="2002-10-31T21:42:13Z" Issuer="http://localhost:8080/idp"
            MajorVersion="1" MinorVersion="2"
            xsi:type="lib:AssertionType">
            <saml:Conditions
               NotBefore="2002-10-31T21:42:12Z"
               NotOnOrAfter="2002-10-31T21:42:43Z">
               <saml:AudienceRestrictionCondition>
                  <saml:Audience>http://localhost:8080/sp</saml:Audience>
               </saml:AudienceRestrictionCondition>
```

```
            </saml:Conditions>
            <saml:AuthenticationStatement
               AuthenticationInstant="2002-10-31T21:42:13Z"
               AuthenticationMethod="urn:oasis:names:tc:SAML:1.0:am:password"
               xsi:type="lib:AuthenticationStatementType">
               <saml:Subject xsi:type="lib:SubjectType">
                  <saml:NameIdentifier Format="urn:liberty:iff:nameid:federated">
                    C9FfGouQdBJ7bpkismYgd8ygeVb3PlWK
                  </saml:NameIdentifier>
                  <saml:SubjectConfirmation>
                     <saml:ConfirmationMethod>
                        urn:oasis:names:tc:SAML:1.0:cm:artifact
                     </saml:ConfirmationMethod>
                  </saml:SubjectConfirmation>
                  <lib:IDPProvidedNameIdentifier>
                     C9FfGouQdBJ7bpkismYgd8ygeVb3PlWK
                  </lib:IDPProvidedNameIdentifier>
               </saml:Subject>
            </saml:AuthenticationStatement>
            <ds:Signature>
               <ds:SignedInfo>
                  <ds:CanonicalizationMethod
                     Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#">
                  </ds:CanonicalizationMethod>
                  <ds:SignatureMethod Algorithm="http://www.w3.org/2000/09/xmldsig#rsa-sha1">
                  </ds:SignatureMethod>
                  <ds:Reference URI="">
                     <ds:Transforms>
                        <ds:Transform
                           Algorithm="http://www.w3.org/2000/09/xmldsig#enveloped-signature">
                        </ds:Transform>
                        <ds:Transform Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#">
                        </ds:Transform>
                     </ds:Transforms>
                     <ds:DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1">
                     </ds:DigestMethod>
                     <ds:DigestValue>ZbscbqHTX9H8bBftRIWlG4Epv1A=</ds:DigestValue>
                  </ds:Reference>
               </ds:SignedInfo>
               <ds:SignatureValue>
                  H+q3nC3jUalj1uKUVkcC4iTFClxeZQIFF0nvHqPS5oZhtkBaDb9qITA7gIkotaB584wXqTXwsfsu
                  IrwT5uL3r85Rj7IF6NeCeiy3K0+z3uewxyeZPz8wna449VNm0qNHYkgNak9ViNCp0/ks5MAttoPo
                  2iLOfaKu3wWG6d1G+DM=
               </ds:SignatureValue>
            </ds:Signature>
         </saml:Assertion>
      </samlp:Response>
   </soap-env:Body>
</soap-env:Envelope>
```

2.1.2.7. Example of Message Exchange Using URL-encoding

```
http://127.0.0.1:8080/tfsidp/IDPSingleSignOnServiceV12?
   RequestID=ddd4aa20-24d4-4366-8f60-7bb0e068334a&MajorVersion=1
   &MinorVersion=2&IssueInstant=2003-07-26T05%3A44%3A00Z
   &ProviderID=http%3A%2F%2F127.0.0.1%3A8081%2Ftfssp
   &NameIDPolicy=none&IsPassive=0
   &RelayState=3f53f0934a63728de06f555493683e131ad131ff
   &SigAlg=http%3A%2F%2Fwww.w3.org%2F2000%2F09%2Fxmldsig%23rsa-sha1
   &Signature=
      kAfiiHlyI8OJG6IGV6Y17ToaxwF1E4wv%2FrONqAkkCllKyxvjcO3%2Bitx7Q44v
      RVDm%2BbAm38ofFSsb%0AbzYsWgTMAiDPsm2wQUuYP2oRB9AApS%2BdZwfGWhqV%
      2FU0malW7cRMDBA2ewyAQTdiALLesetQ6zAnJ%0A4KzHExQl%2BzelPehsWOE%3D
```

Example 1. URL-encoded `<AuthnRequest>`

```
http://127.0.0.1:8081/tfssp/SPAssertionConsumerServiceV12?
   SAMLart=AAMZf3sIji1R%2BgQe%2BMVfP6flshoGaPG1XrZG6E%2B2l2nHDCxFA7h%2BDcE%2B
   &RelayState=3f53f0934a63728de06f555493683e131ad131ff
```

Example 2. URL-encoded `<AuthnResponse>`

3. Profiles

This section defines the Liberty profiles for the use of request and response messages defined in [LibertyProtSchema] and [SAMLCore11]. The combination of message content specification and message transport mechanisms for a single client type (that is, user agent) is termed a *Liberty profile*. The profiles have been grouped into categories according to the protocol message intent.

The following profile categories are defined in this document:

- **Single Sign-On and Federation:** The profiles by which a service provider obtains an authentication assertion from an identity provider facilitating single sign-on and identity federation.
- **Name Registration:** The profiles by which service providers and identity providers specify the name identifier to be used when communicating with each other about the Principal.
- **Federation Termination Notification:** The profiles by which service providers and identity providers are notified of federation termination.
- **Single Logout:** The profiles by which service providers and identity providers are notified of authenticated session termination.
- **Identity Provider Introduction:** The profile by which a service provider discovers which identity providers a Principal may be using.
- **Name Identifier Mapping:** The profile by which a service provider may obtain a NameIdentifier with which to refer to a Principal at a SAML Authority.
- **Name Identifier Encryption:** The profile by which one provider may encrypt a NameIdentifier to permit it to pass through a third-party without revealing the actual value until received by the intended provider.

3.1. Common Requirements

The following rules apply to all profiles in this specification, unless otherwise noted by the individual profile.

1. All HTTP requests and responses MUST be drawn from either HTTP 1.1 (see [RFC2616]) or HTTP 1.0 (see [RFC1945]). When an HTTP redirect is specified, the HTTP response MUST have a status code of "302.". According to HTTP 1.1 and HTTP 1.0, the use of status code 302 is recommended to indicate "the requested resource resides temporarily under a different URI." The response may also include additional headers and an optional message.
2. When `https` is specified as the `<scheme>` for a URL, the HTTP connection MUST be made over either SSL 3.0 (see [SSL]) or TLS 1.0 (see [RFC2246]) or any subsequent protocols that are backwards compatible with SSL 3.0 and/or TLS 1.0. Other security protocols MAY be used as long as they implement equivalent security measures.
3. Messages between providers MUST have their integrity protected, confidentiality MUST be ensured and the recipient MUST authenticate the sender.
4. Providers MUST use secure transport (`https`) to achieve confidentiality and integrity protection. The initiator of the secure connection MUST authenticate the server using server-side X.509 certificates.

5. The authenticated identity of an identity provider MUST be securely available to a Principal before the Principal presents his/her personal authentication data to that identity provider.

6. Certificates and private keys MUST be suitable for long-term signatures. See [LibertyProtSchema] for guidelines on signature verification. For signing and verification of protocol messages, identity and service providers SHOULD use certificates and private keys that are distinct from the certificates and private keys applied for SSL or TLS channel protection.

7. In transactions between service providers and identity providers, requests MUST be protected against replay, and received responses MUST be checked for correct correspondence with issued requests. (**Note:** Other steps may intervene between the issuance of a request and its eventual response within a multistep transaction involving redirections.) Additionally, time-based assurance of freshness MAY be provided.

8. Each service provider within a circle of trust MUST be configured to enable identification of the identity providers whose authentications it will accept. Each identity provider MUST be configured to enable identification of the service providers it intends to serve.
   Note:
   The format of this configuration is a local matter and could, for example, be represented as lists of names or as sets of X.509 certificates of other circle of trust members.

9. Circle of trust bilateral agreements on selecting certificate authorities, obtaining X.509 credentials, establishing and managing trusted public keys, and tracking lifecycles of corresponding credentials are assumed and not in scope for this specification.

10. The `<scheme>` of the URL for SOAP endpoints MUST be `https`.

11. All SOAP message exchanges MUST adhere to the SOAP protocol binding for Liberty (see Section 2.1).

3.1.1. User Agent

Unless otherwise noted in the specific profile, a user agent MUST support the following features to be interoperable with the protocols in [LibertyProtSchema] and Liberty profiles in this document:

- HTTP 1.0 (see [RFC1945]) or HTTP 1.1 (see [RFC2616]).
- SSL 3.0 (see [SSL]) or TLS 1.0 (see [RFC2246]) or any subsequent protocols which are backwards compatible with SSL 3.0 and/or TLS 1.0 either directly or via a proxy (for example, a WAP gateway).
- Minimum maximum URL length of 256 bytes. See [LibertyGlossary] for definition.
- A WAP browser user agent MUST support WML 1.0,1.1, 1.2 or 1.3 [WML] in addition to the above requirements.

Additionally, to support the optional identity provider introduction profile, either the user agent or a proxy must support session cookies (see [RFC2965]). The issue of using persistent cookies or session-length cookies is discussed in [LibertyImplGuide].

3.1.2. Formatting and Encoding of Protocol Messages

All protocol messages that are indicated by the profile as being communicated in the `<query>` component of the URL MUST adhere to the formatting and encoding rules in Section 3.1.2.1.

3.1.2.1. Encoding URL-embedded Messages

URL-embedded messages are encoded using the `application/x-www-form-urlencoded` MIME type as if they were generated from HTML forms with the GET method as defined in [HTML4].

The original XML protocol message MUST be encoded as follows:

- The `<query>` component parameter value MUST be the value of the XML protocol message element or attribute value.
- The value of the `<query>` component parameter MUST be a space-delimited list when the original message element has multiple values.
- Some of the referenced protocol message elements and attributes are optional. If an optional element or attribute does not appear in the original XML protocol message, then the corresponding data item MUST be omitted from the URL encoded message.
- URLs appearing in the URL-encoded message SHOULD NOT exceed 80 bytes in length (including %-escaping overhead). Likewise, the `<lib:RelayState>` data value SHOULD NOT exceed 80 bytes in length.
- The URL-encoding of status codes in the responses `RegisterNameIdentifierResponse` and `LogoutResponse` may be taken from several sources. The top level codes MUST be from SAML. Other codes (including Liberty-defined values) MAY be used at the second or lower levels. The URL parameter value should be interpreted as a QName with the "`lib`", "`saml`", and "`samlp`" namespaces pre-defined to their respective namespace URIs. Query parameters with the name "`xmlns:prefix`" can be used to map additional namespace prefixes for the purpose of QName resolution, so long as the `xmlns:prefix` URL parameter appears before the URL parameter containing the QName which needs the prefix definition.

  As `<samlp:StatusCode>` elements may be nested hierarchically (see [[SAMLCore11]), there may exist multiple values for `<samlp:StatusCode>` in the response messages. These multiple values MUST be encoded by producing a URL-encoded space-separated string as the value of this query parameter. An example is as follows:

  ```
  Value=samlp%3AResponder%20lib%3AFederationDoesNotExist
  ```

- Certain XML protocol messages support extensibility via an `<Extension>` element. Messages that are to be URL-encoded MUST adhere to the following restrictions when including extension content:
  - Only attribute values and elements with simple content models are permitted.
  - All attributes and elements MUST have an empty namespace and MUST have unique local names.
  - Each value included SHOULD NOT exceed 80 bytes in length (including encoding overhead).

XML digital signatures are not directly URL-encoded due to space concerns. If the Liberty XML protocol message is signed with an XML signature, the encoded URL form of the message MUST be signed as follows:

- Include the signature algorithm identifier as a new `<query>` component parameter named SigAlg, but omit the signature.

- Sign the string containing the URL-encoded message. The string to be signed MUST include only the `<query>` part of the URL (that is, everything after `?` and before `&Signature=`). Any required URL-escaping MUST be done before signing.

- Encode the signature using base64 (see [RFC2045]).

- Add the base64-encoded signature to the encoded message as a new data item named `Signature`.

Note that some characters in the base64-encoded signature value may require URL escaping before insertion into the URL `<query>` part, as is the case for any other data item value.

Any items added after the `Signature <query>` component parameter are implicitly unsigned.

The service URL provided by the provider (the URL to which `<query>` parameters are added) MUST NOT contain any pre-existing `<query>` parameter values.

The following signature algorithms (i.e., `DSAwithSHA1, RSAwithSHA1`) and their identifiers (the URIs) MUST be supported:

- `DSAwithSHA1 - http://www.w3.org/2000/09/xmldsig#dsa-sha1`

- `RSAwithSHA1 - http://www.w3.org/2000/09/xmldsig#rsa-sha1`

3.1.2.1.1. Size Limitations

When the request initiator knows or suspects that the user agent cannot process the full URL-encoded message in the URL due to size considerations, the requestor MAY send the Liberty XML protocol message using a form POST. The form MUST be constructed with contents that contain the field `LAREQ` or `LARES` with the respective value being the Liberty XML protocol request or response message (e.g., `<lib:AuthnRequest>` or `<lib:AuthnResponse>`) as defined in [LibertyProtSchema]. The Liberty XML protocol message MUST be encoded by applying a base64 transformation (refer to [RFC2045]) to the XML message and all its elements.

3.1.2.1.2. URL-encoded <lib:AuthnRequest>

The original `<lib:AuthnRequest>` message:

```
<lib:AuthnRequest RequestID="[RequestID]"
   MajorVersion="[MajorVersion]"
   MinorVersion="[MinorVersion]"
   IssueInstant="[IssueInstant]"
   consent="[consent]">
   <lib:ProviderID>[ProviderID]</lib:ProviderID>
   <lib:AffiliationID>[AffiliationID]</lib:AffiliationID>
   <lib:ForceAuthn>[ForceAuthn]</lib:ForceAuthn>
   <lib:IsPassive>[IsPassive]</lib:IsPassive>
   <lib:NameIDPolicy>[NameIDPolicy]</lib:NameIDPolicy>
   <lib:ProtocolProfile>[ProtocolProfile]</lib:ProtocolProfile>
   <lib:AssertionConsumerServiceID>[AssertionConsumerServiceID]
   </lib:AssertionConsumerServiceID>
   <lib:AuthnContext>
      <lib:AuthnContextStatementRef>[AuthnContextStatementRef]
      </lib:AuthnContextStatementRef>
   </lib:AuthnContext>
   <lib:RelayState>[RelayState]</lib:RelayState>
   <lib:AuthnContextComparison>[AuthnContextComparison]</lib:AuthnContextComparison>
```

```
            <lib:Scoping>
               <lib:ProxyCount>[ProxyCount]</lib:ProxyCount>
               <lib:IDPList>
                  <lib:IDPEntries>[IDPEntries]</lib:IDPEntries>
                  <lib:GetComplete>[GetComplete]</lib:GetComplete>
               </lib:IDPList>
            </lib:Scoping>
         </lib:AuthnRequest>
```

- Data elements that MUST be included in the encoded data with their values as indicated in brackets above if present in the original message:

  `RequestID, MajorVersion, MinorVersion, IssueInstant, ProviderID, AffiliationID, ForceAuthn, IsPassive, NameIDPolicy, ProtocolProfile, AuthnContextStatementRef, AuthnContextClassRef, AuthnContextComparison, RelayState, ProxyCount, IDPEntries, GetComplete, consent.`

- The `<IDPEntries>` element may contain multiple `<IDPEntry>` elements, each of which may contain multiple pieces of data (`<ProviderID>`, `<ProviderName>` and `<Loc>`). The `<IDPEntries>` element MUST be URL-encoded by taking only the `<ProviderID>` element from each individual `<IDPEntry>` element, and concatenating them in a space-separated string, as in the following example:

  ```
  ... &IDPEntries=http%3A%2F%2Fidp1.com%2Fliberty%2F%20http%3A%2F%2Fidp2.com%2Fliberty%2F ...
  ```

  The recipient of such a URL-encoded list of `<ProviderID>` elements may obtain the remainder of the information present in the original `<IDPEntry>` by accessing metadata for the individual providers referenced in the URL-encoded list.

- Example of `<lib:AuthnRequest>` message URL-encoded and signed:

  ```
  http://idp.example.com/authn?RequestID=RMvY34pg%2FV9aGJ5yw0HL0AcjcqQF
  &MajorVersion=1&MinorVersion=2&IssueInstant=2002-05-15T00%3A58%3A19
  &consent=urn%3Aliberty%3Aconsent%3Aobtained&ProviderID=http%3A%2F%2Fsp.example.com%2Fliberty%2F

  &ForceAuthn=true&IsPassive=false&NameIDPolicy=federated
  &ProtocolProfile=http%3A%2F%2Fprojectliberty.org%2Fprofiles%2Fbrws-post
  http%3A%2F%2Fwww.projectliberty.org%2Fschemas%2Fauthctx%2Fclasses%2FPasswordProtectedTransport
  &RelayState=03mhakSms5tMQ0WRDCEzpF7BNcywZa75FwIcSSEPvbkoFxaQHCuNnc5yChId
  DlWc7JBV9Xbw3avRBK7VFsPl2X
  &SigAlg=http%3A%2F%2Fwww.w3.org%2F2000%2F09%2Fxmldsig%23rsa-sha1
  &Signature=EoD8bNr2jEOe%2Fumon6oU%2FZGIIF7gbJAe4MLUUMrD%2BPP7P8Yf3gfdZG2qPJdNAJkzVHGfO8W8DzpQ
  %0D%0AsDTTd5VP9MLPcvxbFQoF0CJJmvL26cPsuc54q7ourcH0jJ%2F2UkDq4DAlYlZ5kPIg%2BtrykgLz0U%2BS%0D%
  0ANqpNHkjh6W3YkGv7RBs%3D
  ```

3.1.2.1.3. URL-Encoded <lib:FederationTerminationNotification>

The original `<lib:FederationTerminationNotification>` message:

```
<lib:FederationTerminationNotification ...
   RequestID="[RequestID]"
   MajorVersion="[MajorVersion]"
   MinorVersion="[MinorVersion]"
   IssueInstant="[IssueInstant]"
   consent="[consent]">
   <lib:ProviderID>[ProviderID]</lib:ProviderID>
   <saml:NameIdentifier
      NameQualifier="[NameQualifier]"
 Format="[NameFormat]">[NameIdentifier]</saml:NameIdentifier>
</lib:FederationTerminationNotification>
```

- Data elements that MUST be included in the encoded data with their values as indicated in brackets above if present in the original message:

  `RequestID, MajorVersion, MinorVersion, IssueInstant, ProviderID, NameQualifier, NameFormat, NameIdentifier, consent.`

3.1.2.1.4. URL-Encoded <lib:LogoutRequest>

The original `<lib:LogoutRequest>` message:

```
<lib:LogoutRequest ...
    RequestID="[RequestID]"
    MajorVersion="[MajorVersion]"
    MinorVersion="[MinorVersion]"
    IssueInstant="[IssueInstant]"
    consent="[consent]">
      <lib:ProviderID>[ProviderID]</lib:ProviderID>
      <saml:NameIdentifier
         NameQualifier="[NameQualifier]"
         Format="[NameFormat]">
           [NameIdentifier]
      </saml:NameIdentifier>
      <lib:SessionIndex>[SessionIndex]</lib:SessionIndex>
      <lib:RelayState>[RelayState]</lib:RelayState>
   </lib:LogoutRequest>
```

- Data elements that MUST be included in the encoded data with their values as indicated in brackets above if present in the original message:

  `RequestID, MajorVersion, MinorVersion, IssueInstant, ProviderID, NameQualifier, NameFormat, NameIdentifier, SessionIndex, RelayState, consent.`

3.1.2.1.5. URL-Encoded <lib:LogoutResponse>

The `<lib:LogoutResponse>` response message:

```
<lib:LogoutResponse
   ResponseID="[ResponseID]"
   InResponseTo="[InResponseTo]"
   MajorVersion="[MajorVersion]"
   MinorVersion="[MinorVersion]"
   IssueInstant="[IssueInstant]"
   Recipient="[Recipient]">
 <lib:ProviderID>[ProviderID]</lib:ProviderID>
 <samlp:Status>
 <samlp:StatusCode Value="[Value]"/>
 </samlp:Status>
 <lib:RelayState>[RelayState]</lib:RelayState>
</lib:LogoutResponse>
```

- Data elements that MUST be included in the encoded data with their values as indicated in brackets above if present in the original message:

  `ResponseID, InResponseTo, MajorVersion, MinorVersion, IssueInstant, Recipient, ProviderID, Value, RelayState.`

- The <lib:LogoutResponse> message may contain nested status code information. Multiple values MUST be URL-encoded by creating a space-separated list (see general requirements at top of Section 3.1.2.1.5).

3.1.2.1.6. URL-Encoded <lib:RegisterNameIdentifierRequest>

The original <libRegisterNameIdentifierRequest> message:

```
<lib:RegisterNameIdentifierRequest
  RequestID="[RequestID]"
  MajorVersion="[MajorVersion]"
  MinorVersion="[MinorVersion]"
  IssueInstant="[IssueInstant]">
 <lib:ProviderID>[ProviderID]</lib:ProviderID>
 <lib:IDPProvidedNameIdentifier
   NameQualifier="[IDPNameQualifier]"
   Format="[IDPNameFormat]">[IDPProvidedNameIdentifier]
 </lib:IDPProvidedNameIdentifier>
 <lib:SPProvidedNameIdentifier
   NameQualifier="[SPNameQualifier]"
   Format="[SPNameFormat]">[SPProvidedNameNameIdentifier]
 </lib:SPProvidedNameIdentifier>
 <lib:OldProvidedNameIdentifier
   NameQualifier="[OldNameQualifier]"
   Format="[OldNameFormat]">[OldProvidedNameIdentifier]
 </lib:OldProvidedNameIdentifier>
 <lib:RelayState>[RelayState]</lib:RelayState>
</lib:RegisterNameIdentifierRequest>
```

- Data elements that MUST be included in the encoded data with their values as indicated in brackets above if present in the original message:

  ```
  RequestID, MajorVersion, MinorVersion, IssueInstant,
  ProviderID, IDPNameQualifier, IDPNameFormat, IDPProvidedNameIdentifier,
  SPNameQualifier, SPNameFormat, SPProvidedNameIdentifier,
  OldNameQualifier, OldNameFormat, OldProvidedNameIdentifier,
  RelayState
  ```

3.1.2.1.7. URL-Encoded <lib:RegisterNameIdentifierResponse>

The <lib:RegisterNameIdentifierResponse> response message:

```
<lib:RegisterNameIdentifierResponse
  ResponseID="[ResponseID]"
  InResponseTo="[InResponseTo]"
  MajorVersion="[MajorVersion]"
  MinorVersion="[MinorVersion]"
  IssueInstant="[IssueInstant]"
  Recipient="[Recipient]">
 <lib:ProviderID>[ProviderID]</lib:ProviderID>
 <samlp:Status>
 <samlp:StatusCode Value="[Value]"/>
 </samlp:Status>
 <lib:RelayState>[RelayState]</lib:RelayState>
</lib:RegisterNameIdentifierResponse>
```

- Data elements that MUST be included in the encoded data with their values as indicated in brackets above if present in the original message:

```
ResponseID, InResponseTo, MajorVersion, MinorVersion,
IssueInstant,  Recipient, ProviderID, Value, RelayState
```

- The `<lib:RegisterNameIdentifierResponse>` message may contain nested status code information. Multiple values MUST be URL-encoded by creating a space-separated list (see general requirements at top of Section 3.1.2.1.

3.1.3. Provider Metadata

The majority of the Liberty profiles defined in this document rely on metadata that specify the policies that govern the behavior of the service provider or identity provider. These provider metadata may be shared out of band between an identity provider and a service provider prior to the exchange of Liberty protocol messages or with the protocols described in [LibertyMetadata]. The provider metadata relevant to each profile are listed in this document at the beginning of the profile category. Refer to [LibertyMetadata] for a complete enumeration of the Liberty provider metadata elements and their associated schema.

3.2. Single Sign-On and Federation Profiles

This section defines the profiles by which a service provider obtains an authentication assertion of a user agent from an identity provider to facilitate single sign-on. Additionally, the single sign-on profiles can be used as a means of federating an identity from a service provider to an identity provider through the use of the `<NameIDPolicy>` element in the `<lib:AuthnRequest>` protocol message as specified in [LibertyProtSchema].

The single sign-on profiles make use of the following metadata elements, as defined in [LibertyProtSchema]:

- `ProviderID` Used to uniquely identify the service provider to the identity provider and is documented in these profiles as "service provider ID."
- `AffiliationID` Used to uniquely identify an affiliation group to the identity provider and is documented in these profiles as "affiliation ID."
- `SingleSignOnServiceURL` The URL at the identity provider that the service provider should use when sending single sign-on and federation requests. It is documented in these profiles as "single sign-on service URL."
- `AssertionConsumerServiceURL` The URL(s) at the service provider that an identity provider should use when sending single sign-on or federation responses. It is documented in these profiles as "assertion consumer service URL."
- `SOAPEndpoint` The SOAP endpoint location at the service provider or identity provider to which Liberty SOAP messages are sent.

3.2.1. Common Interactions and Processing Rules

This section defines the set of interactions and process rules that are common to all single sign-on profiles.

All single sign-on profiles can be described by one interaction diagram, provided that different messages are optional in different profiles and that the actual content of the messages may differ slightly. Where interactions and messages differ or are optional, they are designated and detailed within the specific single sign-on profiles. Figure 1 represents the basic template of interactions for achieving single sign-on. This should be used as the baseline for all single sign-on profiles.

In the figure below, steps 1 through 5 can be considered typical but optional. An identity provider MAY initiate a SSO profile by unilaterally creating a `<lib:AuthnResponse>` or artifact, and proceeding with step 6, as discussed in [LibertyProtSchema].

It should be noted that multiple identity providers may be involved in the authentication of the Principal. Although a single identity provider is depicted in the profiles below (Figure 1, that identity provider MAY interact with other identity providers to authenticate the Principal using the proxying method described in [LibertyProtSchema] and the profiles as noted below. In such situations these profiles would be used by the identity provider originally contacted by the requesting service provider to communicate with additional identity providers.

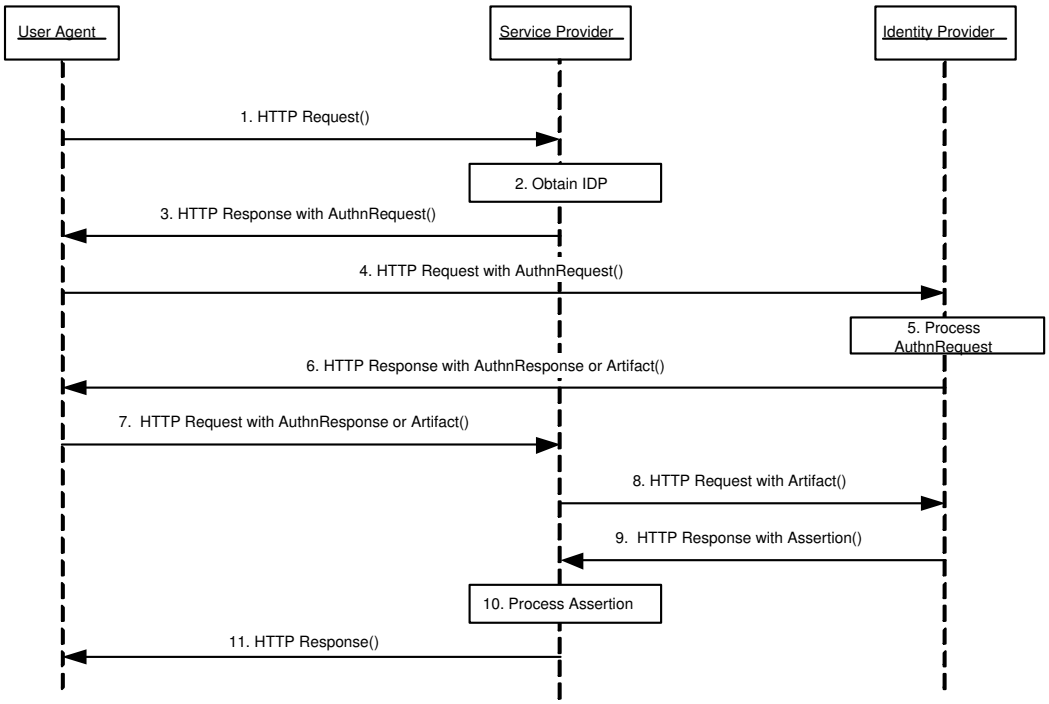


Figure 1. Basic single sign-on profile.

3.2.1.1. Step 1: HTTP Request

In step 1, the user agent accesses the intersite transfer service at the service provider with information about the desired target attached to the URL. Typically, access to the intersite transfer service occurs via a redirection by the service provider in response to a user agent request for a restricted resource.

It is RECOMMENDED that the HTTP request be made over either SSL 3.0 (see [SSL]) or TLS 1.0 (see [RFC2246]) to maintain confidentiality and message integrity in step 1.

3.2.1.2. Step 2: Obtain Identity Provider

In step 2, the service provider obtains the address of the appropriate identity provider to redirect the user agent to in step 3. The means by which the identity provider address is obtained is implementation-dependent and up to the service provider. The service provider MAY use the Liberty identity provider introduction profile in this step.

3.2.1.3. Step 3: HTTP Response with <AuthnRequest>

In step 3, the service provider's intersite transfer service responds and sends the user agent to the single sign-on service URL at the identity provider. The form and contents of the HTTP response in this step are profile-dependent.

3.2.1.4. Step 4: HTTP Request with <AuthnRequest>

In step 4, the user agent accesses the identity provider's single sign-on service URL with the `<lib:AuthnRequest>` information. This request may be a GET or POST request; providers MUST support both methods. As described later, such a POST MUST contain an `LAREQ` form element containing the XML protocol request in base64-encoded format.

3.2.1.5. Step 5: Processing <AuthnRequest>

In step 5, the identity provider MUST process the `<lib:AuthnRequest>` message according to the rules specified in [LibertyProtSchema].

If the Principal has not yet been authenticated with the identity provider, authentication at the identity provider MAY occur in this step. The identity provider MAY obtain consent from the Principal for federation, or otherwise consult the Principal. To this end the identify provider MAY return to the HTTP request any HTTP response; including but not limited to HTTP Authentication, HTTP redirect, or content. The identity provider SHOULD respect the HTTP User-Agent and Accept headers and SHOULD avoid responding with content-types that the User-Agent may not be able to accept. Authentication of the Principal by the identity provider is dependent upon the `<lib:AuthnRequest>` message content.

In case the identity provider responds to the user agent with a form, it is RECOMMENDED that the `<input>` parameters of the form be named according to [RFC3106] whenever possible.

3.2.1.6. Step 6: HTTP Response with <AuthnResponse> or Artifact

In step 6, the identity provider MUST respond to the user agent with a `<lib:AuthnResponse>`, a SAML artifact, or an error. The form and contents of the HTTP response in this step are profile-dependent.

3.2.1.7. Step 7: HTTP Request with <AuthnResponse> or Artifact

In step 7, the user agent accesses the assertion consumer service URL at the service provider with a `<lib:AuthnResponse>` or a SAML artifact. This request may be a GET or POST request; providers MUST support both methods. As described later, such a POST MUST contain an `LARES` form element containing the XML protocol request or artifact in base64-encoded format.

3.2.1.8. Step 8: HTTP Request with Artifact

Step 8 is required only for single sign-on profiles that use a SAML artifact.

In this step the service provider, in effect, dereferences the single SAML artifact in its possession to acquire the authentication assertion that corresponds to the artifact.

The service provider MUST send a `<samlp:Request>` SOAP message to the identity provider's SOAP endpoint, requesting the assertion by supplying the SAML assertion artifact in the `<samlp:AssertionArtifact>` element as specified in [SAMLBind11].

The service provider MUST provide a mechanism for the identity provider to authenticate the service provider.

3.2.1.9. Step 9: HTTP Response with Assertion

Step 9 is required only for single sign-on profiles that use a SAML artifact.

In this step if the identity provider is able to find or construct the requested assertion, it responds with a `<samlp:Response>` SOAP message with the requested `<saml:Assertion>`. Otherwise, it returns an appropriate status code, as defined within the “SOAP binding for SAML” (see[SAMLBind11]) and the [LibertyProtSchema].

3.2.1.10. Step 10: Process Assertion

In step 10, the service provider processes the `<saml:Assertion>` returned in the `<samlp:Response>` or `<lib:AuthnResponse>` protocol message to determine its validity and how to respond to the Principal’s original request. The signature on the `<saml:Assertion>` must be verified.

The service provider processing of the assertion MUST adhere to the rules defined in [SAMLCore11] for things such as assertion `<saml:Conditions>` and `<saml:Advice>`.

The service provider MAY obtain authentication context information for the Principal’s current session from the `<lib:AuthnContext>` element contained in `<saml:Advice>`. Similarly, the information in the `<lib:RelayState>` element MAY be obtained and used in further processing by the service provider.

3.2.1.11. Step 11: HTTP Response

In step 11, the user agent is sent an HTTP response that either allows or denies access to the originally requested resource.

3.2.2. Liberty Artifact Profile

The Liberty artifact profile relies on a reference to the needed assertion traveling in a SAML artifact, which the service provider must dereference from the identity provider to determine whether the Principal is authenticated. This profile is an adaptation of the "Browser/artifact profile" for SAML as documented in [SAMLBind11]. See Figure 3.

The following URI-based identifier MUST be used when referencing this specific profile (for example, `<lib:ProtocolProfile>` element of the `<lib:AuthnRequest>` message):

URI: `http://projectliberty.org/profiles/brws-art`

The Liberty artifact profile consists of a single interaction among three parties: a user agent, an identity provider, and a service provider, with a nested subinteraction between the identity provider and the service provider.

3.2.2.1. Interactions

Figure 2 illustrates the Liberty artifact profile for single sign-on.

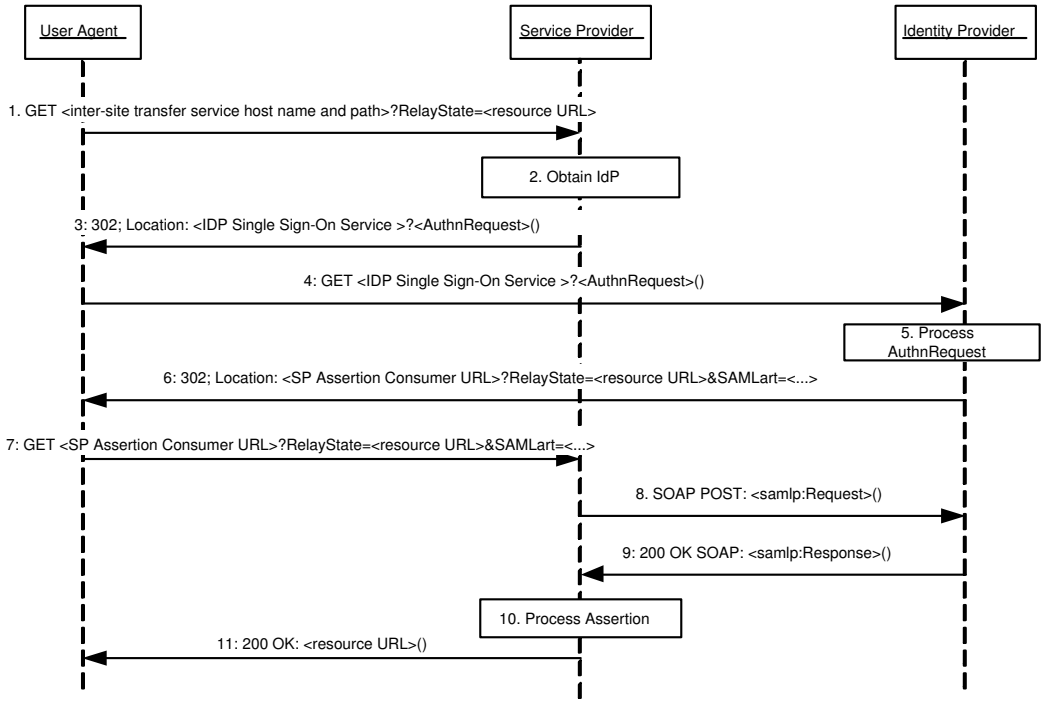


Figure 2. Liberty artifact profile for single sign-on

This profile description assumes that the user agent has already authenticated at the identity provider prior to step 1. Thus, a valid session exists for the user agent at the identity provider. When implementing this profile, all processing rules defined in Section 3.2.1 for the single sign-on profiles MUST be followed. Additionally, the following rules MUST be observed as they relate to steps 3, 6 and 7:

3.2.2.1.1. Step 3: Single sign on Service with <AuthnRequest>

In step 3, the service provider's intersite transfer service responds and instructs the user agent to access the single sign-on service URL at the identity provider.

This step may take place via an HTTP 302 redirect, a WML redirect deck or any other method that results in the user agent being instructed to make an HTTP GET or POST request to the identity provider's single signon service.

This response MUST adhere to the following rules:

- The response MUST contain the identity provider's single sign-on service URL (for example, as the Location header of an HTTP 302 redirect, the `action` attribute of an HTML`form` or the `href` attribute of a `<go>` element in a WML redirect deck).

- The identity provider's single sign-on service URL MUST specify `https` as the URL scheme.
  Note:
  Future protocols may be adopted and enabled to work within this framework. Therefore, implementers are encouraged to not hardcode a reliance on `https`.

- The response MUST include one of the following:

- A <query> component containing the <lib:AuthnRequest> protocol message as defined in [LibertyProtSchema] with formatting as specified in Section 3.1.2.
  Note:
  The <lib:RelayState> element of the <lib:AuthnRequest> message can be used by the service provider to help maintain state information during the single sign-on and federation process. For example, the originally requested resource (i.e., RelayState in step 1) could be stored as the value for the <lib:RelayState> element, which would then be returned to the service provider in the <lib:AuthnResponse> in step 7. The service provider could then use this information to formulate the HTTP response to the user agent in step 11.

- An HTTP form containing the field LAREQ with the value of the <lib:AuthnRequest> protocol message as defined in [LibertyProtSchema]. The <lib:AuthnRequest> MUST be encoded by applying a base64 transformation (see [RFC2045].

Implementation examples:

- HTTP 302 Redirect

```
<HTTP-Version> 302 <Reason Phrase>
<other headers>
Location: https://<Identity Provider Single Sign-On Service host name and path>?<query>
<other HTTP 1.0 or 1.1 components>
```

- HTML Form POST

```
<html>
   <body onLoad="document.forms[0].submit()">
      <form action="https://<Identity Provider Single Sign-On Service host name and path>"
method="POST">
         <input type="hidden" name="LAREQ" value="<base64 encoded AuthnRequest>" >
      </form>
   </body>
</html>
```

- WML Redirect with POST

```
            ...
<wml>
<card id="redirect" title="Log In">
   <onenterforward>
      <go method="post" href="<Identity Provider Single Sign-On service host name and path>" >
         <postfield name="LAREQ" Value="<base64-encoded AuthnRequest>" />
      </go>
   </onenterforward>
   <onenterbackward>
      <prev/>
   </onenterbackward>
   <p>
      Contacting IdP. Please wait...
   </p>
...
</card>
...
</wml>
```

- WML Redirect with GET

```
                ...
<wml>
<card id="redirect" title="Log In">
   <onenterforward>
      <go href="<Identity Provider Single Sign-On service host name and path>?<query>" />
   </onenterforward>
   <onenterbackward>
      <prev/>
   </onenterbackward>
   <p>
      Contacting IdP. Please wait...
   </p>
...
</card>
...
</wml>
```

where:

`<Identity Provider Single Sign-On service host name and path>`

This element provides the host name, port number, and path components of the single sign-on service URL at the identity provider.

`<query>= ...<URL-encoded AuthnRequest> ...`

A `<query>` component MUST contain a single authentication request:

`<base64-encoded AuthnRequest>`

A `<base64-encoded AuthnRequest>` component MUST contain a single authentication request message in base64-encoded form.

3.2.2.1.2. Step 6: Redirecting to the Service Provider

In step 6, the identity provider instructs the user agent to access the service provider's assertion consumer service URL, and provides a SAML artifact for de-referencing by the service provider.

This step may take place via an HTTP 302 redirect, a WML redirect deck or any other method that results in the user agent being instructed to make an HTTP GET or POST request to the service provider's assertion consumer service.

This response MUST adhere to the following rules:

- The response MUST contain the service provider's assertion consumer service URL (for example, as the Location header of an HTTP 302 redirect, the `action` attribute of an HTML`form` or the `href` attribute of a `<go>` element in a WML redirect deck).

- The service provider's assertion consumer service URL MUST specify `https` as the URL scheme.
  Note:
  Future protocols may be adopted and enabled to work within this framework. Therefore, implementers are encouraged to not hardcode a reliance on `https`.

- The response MUST include one of the following:

- A <query> component containing a parameter SAMLart, the value of which is the SAML artifact on success or on failure. In the case of failure, the status will be conveyed in the <saml:Response> returned in Step 9. Additionally, if the <lib:AuthnRequest> processed in Step 5 included a value for the <lib:RelayState> element, then a parameter named RelayState with a value set to that of the <lib:RelayState> element MUST be included in the <query> component.

- An HTTP form containing the field LARES with the value of the SAML Artifact as defined in Section 3.2.2.2 If a value for <RelayState> was supplied in the <lib:AuthnRequest>, then the form MUST contain a field RelayState, with a value obtained from that element in the <lib:AuthnRequest>.

- All SAML artifacts returned MUST contain the same identity provider ID.

Implementation examples:

- HTTP 302 Redirect

```
<HTTP-Version> 302 <Reason Phrase>
<other headers>
Location: https://<Service Provider Assertion Consumer Service host name and path>?<query>
<other HTTP 1.0 or 1.1 components>
```

- HTML Form POST

```
<html>
   <body onLoad="document.forms[0].submit()">
      <form action="https://<Service Provider Assertion Consumer Service host name and path>"
method="POST">
         <input type="hidden" name="LAREQ" value="<SAML Artifact>" >
         <input type="hidden" name="RelayState" value="<RelayState>" >
      </form>
   </body>
</html>
```

- WML Redirect with POST

```
            ...
<wml>
<card id="redirect" title="Artifact">
   <onenterforward>
      <go method="post" href="<Service Provider Assertion Consumer Service host name and path>" >
         <postfield name="LARES" Value="<SAML Artifact>" />
         <postfield name="RelayState" Value="<RelayState>" />
      </go>
   </onenterforward>
   <onenterbackward>
      <prev/>
   </onenterbackward>
   <p>
      Contacting IdP. Please wait...
   </p>
...
</card>
...
</wml>
```

- WML Redirect with GET

```
                ...
  <wml>
  <card id="redirect" title="Artifact">
     <onenterforward>
        <go href="<Service Provider Assertion Consumer Service host name and path>?<query>" />
     </onenterforward>
     <onenterbackward>
        <prev/>
     </onenterbackward>
     <p>
        Contacting IdP. Please wait...
     </p>
  ...
  </card>
  ...
  </wml>
```

where:

`<Service Provider Assertion Consumer Service host name and path>`

This element provides the host name, port number, and path components of the assertion consumer service URL at the service provider.

`<query>= ...SAMLArt=<SAML Artifact> ...RelayState=<resource URL>`

A `<query>` component MUST contain at least one SAML Artifact. A single RelayState MUST be included if a value for the <RelayState> was provided in the `<lib:AuthnRequest>`. All SAML Artifacts included MUST contain the same identity provider ID (see Section 3.2.2.2).

`<SAML Artifact>`

A `<SAML Artifact>` component MUST contain at least one SAML Artifact.

`<RelayState>`

A form field named RelayState, with the value of that element from the `<lib:AuthnRequest>` MUST be included if a value for the `<RelayState>` was provided in the `<lib:AuthnRequest>` and the HTTP request is made using a POST.

3.2.2.1.3. Step 7: Accessing the Assertion Consumer Service

In step 7, the user agent accesses the assertion consumer service URL at the service provider, with a SAML artifact representing the Principal's authentication information attached to the URL.

3.2.2.2. Artifact Format

The artifact format includes a mandatory two-byte artifact type code, as follows:

```
SAML_artifact    := B64(TypeCode RemainingArtifact)
TypeCode        := Byte1Byte2
```

The notation `B64(TypeCode RemainingArtifact)` represents the application of the base64 transformation to the catenation of the `TypeCode` and `RemainingArtifact`. This profile defines an artifact type of type code `0x0003`,

which is REQUIRED (mandatory to implement) for any implementation of the Liberty browser artifact profile. This artifact type is defined as follows:

```
TypeCode       := 0x0003
RemainingArtifact := IdentityProviderSuccinctID AssertionHandle
IdentityProviderSuccinctID:= 20-byte_sequence
AssertionHandle  := 20-byte_sequence
```

`IdentityProviderSuccinctID` is a 20-byte sequence used by the service provider to determine identity provider identity and location. It is assumed that the service provider will maintain a table of `IdentityProviderSuccinctID` values as well as the URL (or address) for the corresponding SAML responder at the identity provider. This information is communicated between the identity provider and service provider out of band. On receiving the SAML artifact, the service provider determines whether the `IdentityProviderSuccinctID` belongs to a known identity provider and, if so, obtains the location before sending a SAML request.

Any two identity providers with a common service provider MUST use distinct `IdentityProviderSuccinctID` values. Construction of `AssertionHandle` values is governed by the principles that the values SHOULD have no predictable relationship to the contents of the referenced assertion at the identity provider, and that constructing or guessing the value of a valid, outstanding assertion handle MUST be infeasible.

The following rules MUST be followed for the creation of SAML artifacts at identity providers:

- Each identity provider selects a single identification URL, corresponding to the provider metadata element ProviderID specified in [LibertyMetadata].
- The identity provider constructs the `IdentityProviderSuccinctID` component of the artifact by taking the SHA-1 hash of the identification URL as a 20-byte binary value. Note that the `IdentityProviderSuccinctID` value, used to construct the artifact, is not encoded in hexadecimal. The `AssertionHandle` value is constructed from a cryptographically strong random or pseudo-random number sequence (see [RFC1750]) generated by the identity provider. The sequence consists of a value of at least eight bytes. The value should be padded to a total length of 20 bytes.

3.2.3. Liberty Browser POST Profile

The Liberty browser POST profile allows authentication information to be supplied to an identity provider without the use of an artifact. Figure 3 diagrams the interactions between parties in the Liberty POST profile. This profile is an adaptation of the "Browser/post profile" for SAML as documented in [SAMLBind11].

The following URI-based identifier MUST be used when referencing this specific profile (for example, `<lib:ProtocolProfile>` element of the `<lib:AuthnRequest>` message):

URI: `http://projectliberty.org/profiles/brws-post`

The Liberty POST profile consists of a series of two interactions, the first between a user agent and an identity provider, and the second directly between the user agent and the service provider.

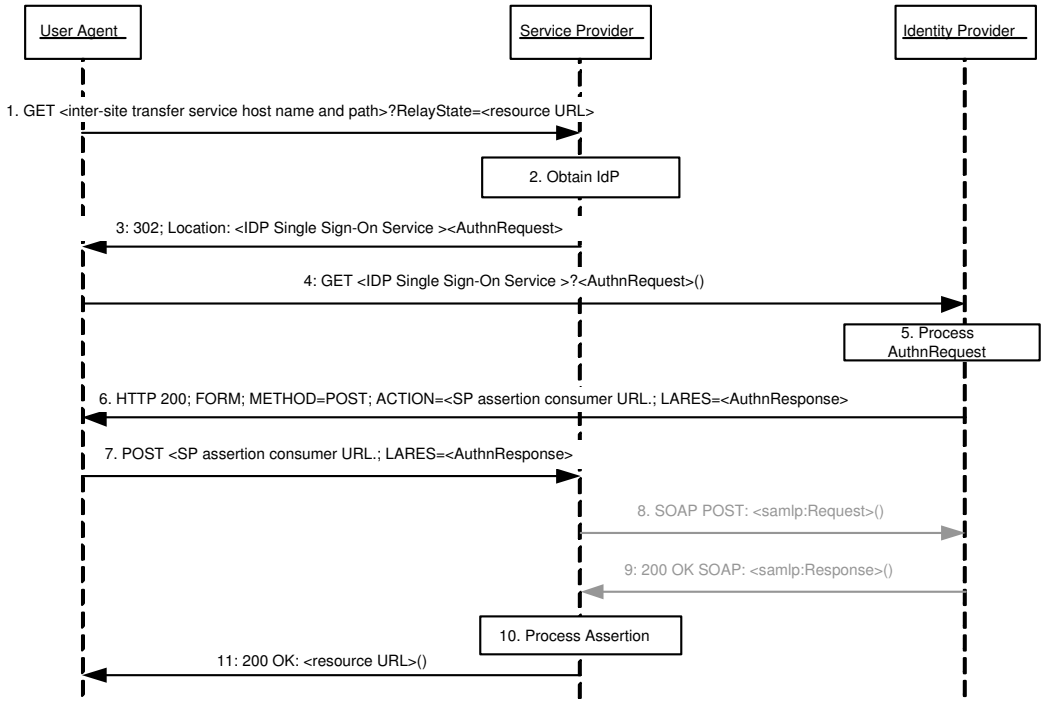


Figure 3. Liberty browser POST profile for single sign-on

This profile description assumes that the user agent has already authenticated at the identity provider prior to step 1. Thus, a valid session exists for the user agent at the identity provider.

When implementing this profile, all processing rules defined in Section 3.2.1 for single sign-on profiles MUST be followed with the exception that steps 8 and 9 MUST be omitted. Additionally, the following rules MUST be observed as they relate to steps 3, 6 and 7:

3.2.3.1. Step 3: Single Sign-On Service with <AuthnRequest>

In step 3, the service provider's intersite transfer service responds and sends the user agent to the single sign-on service URL at the identity provider.

This step may take place via an HTTP 302 redirect, a WML redirect deck or any other method that results in the user agent being instructed to make an HTTP GET or POST request to the identity provider's single sign-on service.

This response MUST adhere to the following rules:

- The response MUST contain the identity provider's single sign-on service URL (for example, as the Location header of an HTTP 302 redirect, the `action` attribute of an HTML`form` or the `href` attribute of a `<go>` element in a WML redirect deck).

- The identity provider's single sign-on service URL MUST specify `https` as the URL scheme.
  Note:
  Future protocols may be adopted and enabled to work within this framework. Therefore, implementers are encouraged to not hardcode a reliance on `https`.

- The response MUST include one of the following:

- A <query> component containing the <lib:AuthnRequest> protocol message as defined in [LibertyProtSchema] with formatting as specified in Section 3.1.2
  Note:
  The <lib:RelayState> element of the <lib:AuthnRequest> message can be used by the service provider to help maintain state information during the single sign-on and federation process. For example, the originally requested resource (that is, RelayState in step 1) could be stored as the value for the <lib:RelayState> element, which would then be returned to the service provider in the <lib:AuthnResponse> in step 7. The service provider could then use this information to formulate the HTTP response to the user agent in step 11.

- An HTTP form containing the field LAREQ with the value of the <lib:AuthnRequest> protocol message as defined in [LibertyProtSchema]. The <lib:AuthnRequest> MUST be encoded by applying a base64 transformation (see [RFC2045]).

See the discussion of this step in the artifact profile for implementation examples.

3.2.3.2. Step 6: Generating and Supplying the <AuthnResponse>

In step 6 the identity provider generates an HTML form containing an authentication assertion that MUST be sent in an HTTP 200 response to the user agent.

The form MUST be constructed such that it requests a POST to the service provider's assertion consumer URL with form contents that contain the field LARES with the value being the <lib:AuthnResponse> protocol message as defined in [LibertyProtSchema]. The <lib:AuthnResponse> MUST be encoded by applying a base64 transformation (refer to [RFC2045]) to the <lib:AuthnResponse> and all of its elements. The service provider's assertion consumer service URL used as the target of the form POST MUST specify https as the URL scheme; if another scheme is specified, it MUST be treated as an error by the identity provider.

Multiple <saml:Assertion> elements MAY be included in the response. The identity provider MUST digitally sign each of the assertions included in the response.

The <saml:ConfirmationMethod> element of the assertion MUST be set to the value specified in [SAMLCore11] for "Assertion Bearer."

3.2.3.3. Step 7: Posting the Form Containing the <AuthnResponse>

In step 7 the user agent issues the HTTP POST request containing the <lib:AuthnResponse> to the service provider.

3.2.4. Liberty-Enabled Client and Proxy Profile

The Liberty-enabled client and proxy profile specifies interactions between Liberty-enabled clients and/or proxies, service providers, and identity providers. See Figure 5. A Liberty-enabled client is a client that has, or knows how to obtain, knowledge about the identity provider that the Principal wishes to use with the service provider. In addition a Liberty-enabled client receives and sends Liberty messages in the body of HTTP requests and responses. Therefore, Liberty-enabled clients have no restrictions on the size of the Liberty protocol messages.

A Liberty-enabled proxy is an HTTP proxy (typically a WAP gateway) that emulates a Liberty-enabled client. Unless stated otherwise, all statements referring to "LECP" are to be understood as statements about both Liberty-enabled clients and Liberty-enabled proxies.

In some environments the successful deployment of a Liberty-Enabled proxy may require that service providers in those environments perform operations in addition to those described below. Such cases, and specific guidance for them, are covered in [LibertyImplGuide].

The following URI-based identifier must be used when referencing this specific profile (for example, <lib:ProtocolProfile> element of the <lib:AuthnRequest> message):

URI: `http://projectliberty.org/profiles/lecp`

A LECP, in addition to meeting the common requirements for profiles in Section 3.1, MUST indicate that it is a LECP by including a Liberty-Enabled header or entry in the value of the HTTP User-Agent header for each HTTP request it makes. The preferred method is the Liberty-Enabled header. The formats of the Liberty-Enabled header and User-Agent header entry are defined in Section 3.2.4.1.

3.2.4.1. Liberty-Enabled Indications

A LECP SHOULD add the Liberty-Enabled header to each HTTP request. The Liberty-Enabled header MUST be named `Liberty-Enabled` and be defined as using Augmented BNF as specified in section 2 of [RFC2616].

```
Liberty-Enabled = "Liberty-Enabled" ":" LIB_Version ["," 1#Extension]
LIB_Version = "LIBV" "=" 1*absoluteURI
; any spaces or commas in the absoluteURI MUST be escaped as defined in section 2.4 of [RFC 2396]
Extension = ExtName "=" ExtValue
ExtName = (["." host] | <any field-value but ".", "," or "="">) <any field-value but "=" or ",">
ExtValue = <any field-value but ",">)
```

The comment, field-value, and product productions are defined in [RFC2616]. `LIB_Version` identifies the versions of the Liberty specifications that are supported by this LECP. Each version is identified by a URI. Service providers or identity providers receiving a Liberty-Enabled header MUST ignore any URIs listed in the `LIB_Version` production that they do not recognize. All LECPs compliant with this specification MUST send out, at minimum, the URI `http://projectliberty.org/specs/v1` as a value in the `LIB_Version` production. It SHOULD precede this with the URI `urn:liberty:iff:2003-08` if it supports version 1.2 requests and knows that the identity providers available to it also support version 1.2 requests and responses. It MUST NOT include this URI if it knows that the identity providers available to it cannot process version 1.2 messages. The ordering of the URIs in the `LIB_Version` header is meaningful; therefore, service providers and identity providers are encouraged to use the first version in the list that they support. Supported Liberty versions are not negotiated between the LECP and the service provider. The LECP advertises what version it supports; the service provider MUST return the response for the corresponding version as defined in step 3 below.

Optional extensions MAY be added to the Liberty-Enabled header to indicate new information. The value of the `ExtName` production MUST use the "`host`" "`;`" prefixed form if the new extension name has not been standardized and registered with Liberty or its designated registration authorities. The value of the host production MUST be an IP or DNS address that is owned by the issuer of the new name. By using the DNS/IP prefix, effectively namespace collisions can be prevented without the need of introducing another centralized registration agency.

A LECP MAY include the Liberty-Agent header in its requests. This header provides information about the software implementing the LECP functionality and is similar to the User-Agent and Server headers in HTTP.

```
Liberty-Agent = "Liberty-Agent" ":" 1*( product | comment)
```

Note:

The reason for introducing the new header (that is, Liberty-Enabled) rather than using User-Agent is that a LECP may be a Liberty-enabled proxy. In such a case the information about the Liberty-enabled proxy would not be in the User-Agent header. In theory the information could be in the VIA header. However, for security reasons, values in the VIA header can be collapsed, and comments (where software information would be recorded) can always be removed. As such, the VIA header is not suitable. Using the User-Agent header for a Liberty-enabled client and the Liberty-Agent header for a Liberty-enabled proxy was also discussed. However, this approach seemed too complex.

Originally the Liberty-Agent header was going to be part of the Liberty-Enabled header. However, header lengths in HTTP implementations are limited; therefore, putting this information in its own header was considered the preferred approach.

A LECP MAY add a Liberty-Enabled entry in the HTTP User-Agent request header. The HTTP User-Agent header is specified in [RFC2616]. A LECP MAY include in the value of this header the `Liberty-Enabled` string as defined above for the Liberty-Enabled header.

Note:

The reason for adding information to the User-Agent header is to allow for Liberty-enabled client products that must rely on a platform that cannot be instructed to insert new headers in each HTTP request.

The User-Agent header is often overloaded; therefore, the Liberty-Enabled header should be the first choice for any implementation of a LECP. The entry in the User-Agent header then remains as a last resort.

3.2.4.2. Interactions

Figure 5 illustrates the Liberty-enabled client and proxy profile for single sign-on.

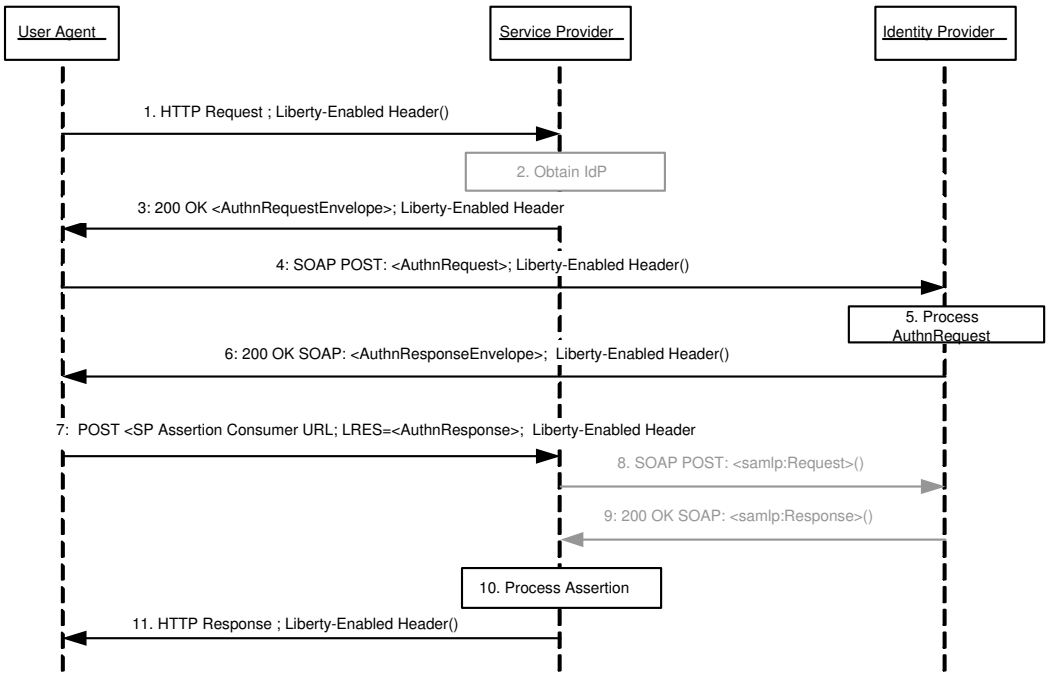


Figure 5. Liberty-enabled client and proxy profile for single sign-on

This profile description assumes that the user agent has already authenticated at the identity provider prior to step 1. Thus, a valid session exists for the user agent at the identity provider.

The LECP receives authentication requests from the service provider in the body of the HTTP response. The LECP submits this authentication request as a SOAP request to the identity provider. Because this SOAP request is between the LECP and the identity provider, TLS authentication cannot be performed between service provider and identity provider; therefore, service providers and identity providers MUST rely on the signature of the `<lib:AuthnRequest>` and the returned `<saml:Assertion>`, respectively, for mutual authentication.

When implementing this profile, processing rules for steps 5, 10, and 11 defined in Section 3.2.1 for single sign-on profiles MUST be followed, while steps 2, 8, and 9 MUST be omitted. Additionally, the following rules MUST be observed as they relate to steps 1, 3, 4, 6, and 7:

3.2.4.2.1. Step 1: Accessing the Service Provider

In step 1, the user agent accesses the service provider with the Liberty-Enabled header (or with the Liberty-Enabled entry in the User-Agent header) included in the HTTP request.

The HTTP request MUST contain only one Liberty-Enabled header. Hence if a proxy receives an HTTP request that contains a Liberty-Enabled header, it MUST NOT add another Liberty-Enabled header. However, a proxy MAY replace the Liberty-Enabled header. A proxy that replaces or adds a Liberty-Enabled header MUST process `<lib:AuthnRequest>` messages as defined in steps 3 and 4 as well as `<lib:AuthnResponse>` messages as specified in steps 6 and 7.

It is RECOMMENDED that a LECP add "`application/vnd.liberty-request+xml`" as one of its supported content types to the Accept header.

3.2.4.2.2. Step 3: HTTP Response with <AuthnRequest>

In step 3, the service provider's intersite transfer service issues an HTTP 200 OK response to the user agent. The response MUST contain a single `<lib:AuthnRequestEnvelope>` with content as defined in [LibertyProtSchema]. If a service provider receives a Liberty-Enabled header, or a User-Agent header with the Liberty-Enabled entry, the service provider MUST respond according to the Liberty-enabled client and proxy profile and include a Liberty-Enabled header in its response. Hence service providers MUST support the Liberty-enabled client and proxy profile.

The processing rules and default values for the Liberty-Enabled indications are as defined in Section 3.2.4.1. The service provider MAY advertise any Liberty version supported in this header, not only the version used for the specific response.

The HTTP response MUST contain a Content-Type header with the value `application/vnd.liberty-request+xml` unless the LECP and service provider have negotiated a different format.

A service provider MAY provide a list of identity providers it recognizes by including the `<lib:IDPList>` element in the `<lib:AuthnRequestEnvelope>`. The format and processing rules for the identity provider list MUST be as defined in [LibertyProtSchema].

Note:

In cases where a value for the `<lib:GetComplete>` element is provided within `<lib:IDPList>`, the URI value for this element MUST specify `https` as the URL `<scheme>`.

The service provider MUST specify a URL for receiving `<AuthnResponse>` elements, locally generated by the intermediary, by including the `<lib:AssertionConsumerServiceURL>` element in the `<lib:AuthnRequestEnvelope>`.

The following example demonstrates the usage of the `<lib:AuthnRequestEnvelope>`:

```
<?xml version="1.0" ?>
<lib:AuthnRequestEnvelope xmlns:lib="urn:liberty:iff:2003-08">
   <lib:AuthnRequest >
      . . . AuthnRequest goes here . . .
   </lib:AuthnRequest>
   <lib:AssertionConsumerServiceURL>
      https://service-provider.com/LibertyLogin
   </lib:AssertionConsumerServiceURL>
   <lib:IDPList >
      . . . IdP list goes here . . .
   </lib:IDPList>
</lib:AuthnRequestEnvelope>
```

If the service provider does not support the LECP-advertised Liberty version, the service provider MUST return to the LECP an HTTP 501 response with the reason phrase "Unsupported Liberty Version."

The responses in step 3 and step 6 SHOULD NOT be cached. To this end service providers and identity providers SHOULD place both "`Cache-Control: no-cache`" and "`Pragma: no-cache`" on their responses to ensure that the LECP and any intervening proxies will not cache the response.

3.2.4.2.3. Step 4: HTTP Request with <AuthnRequest>

In step 4, the LECP determines the appropriate identity provider to use and then issues an HTTP POST of the `<lib:AuthnRequest>` in the body of a SOAP message to the identity provider's single sign-on service URL. The request MUST contain the same `<lib:AuthnRequest>` as was received in the `<lib:AuthnRequestEnvelope>` from the service provider in step 3.

Note:

The identity provider list can be used by the LECP to create a user identifier to be presented to the Principal. For example, the LECP could compare the list of the Principal's known identities (and the identities of the identity provider that provides those identities) against the list provided by the service provider and then only display the intersection.

If the LECP discovers a syntax error due to the service provider or cannot proceed any further for other reasons (for example, cannot resolve identity provider, cannot reach the identity provider, etc.), the LECP MUST return to the service provider a `<lib:AuthnResponse>` with a `<samlp:Status>` indicating the desired error element as defined in [LibertyProtSchema]. The `<lib:AuthnResponse>` containing the error status MUST be sent using a POST to the service provider's assertion consumer service URL obtained from the `<lib:AssertionConsumerServiceURL>` element of the `<lib:AuthnRequestEnvelope>`. The POST MUST be a form that contains the field LARES with the value being the `<lib:AuthnResponse>` protocol message as defined in [LibertyProtSchema], containing the `<samlp:Status>`. The `<lib:AuthnResponse>` MUST be encoded by applying a base64 transformation (refer to [RFC2045]) to the `<lib:AuthnResponse>` and all its elements.

3.2.4.2.4. Step 6: HTTP Response with <AuthnResponse>

In step 6, the identity provider responds to the `<lib:AuthnRequest>` by issuing an HTTP 200 OK response. The response MUST contain a single `<lib:AuthnResponseEnvelope>` in the body of a SOAP message with content as defined in [LibertyProtSchema].

The identity provider MUST include the Liberty-Enabled HTTP header following the same processing rules as defined in 3.2.5.1.

The Content-Type MUST be set to application/vnd.liberty-response+xml.

If the identity provider discovers a syntax error due to the service provider or LECP or cannot proceed any further for other reasons (for example, an unsupported Liberty version), the identity provider MUST return to the LECP a `<lib:AuthnResponseEnvelope>` containing a `<lib:AuthnResponse>` with a `<samlp:Status>` indicating the desired error element as defined in [LibertyProtSchema].

3.2.4.2.5. Step 7: Posting the Form Containing the <AuthnResponse>

In step 7, the LECP issues an HTTP POST of the `<lib:AuthnResponse>` that was received in the `<lib:AuthnResponseEnvelope>` SOAP response in step 6. The `<lib:AuthnResponse>` MUST be sent using a POST to the service provider's assertion consumer service URL identified by the `<lib:AssertionConsumerServiceURL>` element within the `<lib:AuthnResponseEnvelope>` *obtained from the identity provider in step 6*. The POST MUST be a form that contains the field `LARES` with the value being the `<lib:AuthnResponse>` protocol message as defined in [LibertyProtSchema]. The `<lib:AuthnResponse>` MUST be encoded by applying a base64 transformation (refer to [RFC2045]) to the `<lib:AuthnResponse>` and

all its elements. The service provider's assertion consumer service URL used as the target of the form POST MUST specify `https` as the URL scheme; if another scheme is specified, it MUST be treated as an error by the identity provider.

If the LECP discovers an error (for example, syntax error in identity provider response), the LECP MUST return to the service provider a `<lib:AuthnResponse>` with a `<samlp:Status>` indicating the appropriate error element as defined in [LibertyProtSchema]. The `<ProviderID>` in the `<lib:AuthnResponse>` MUST be set to *urn:liberty:iff:lecp*. The `<lib:AuthnResponse>` containing the error status MUST be sent using a POST to the service provider's assertion consumer service URL. The POST MUST be a form that contains the field named LARES with its value being the `<lib:AuthnResponse>` protocol message as defined in [LibertyProtSchema] with formatting as specified Section 3.1.2. Any `<lib:AuthnResponse>` messages created by the identity provider MUST NOT be sent to the service provider.

3.3. Register Name Identifier Profiles

This section defines the profiles by which a provider may register or change a name identifier for a Principal. This message exchange is optional. During federation, the identity provider supplies an opaque handle identifying the Principle. This is the `<lib:IDPProvidedNameIdentifier>`. If neither provider involved in the federation opts to register any other name identifier, then this initial `<lib:IDPProvidedNameIdentifier>` is to be used by both providers.

An identity provider may choose to register a new `<lib:IDPProvidedNameIdentifier>` at any time subsequent to federation, using this protocol. Additionally, a service provider may choose to register a `<lib:SPProvidedNameIdentifier>`, which it expects the identity provider to use (instead of the `<lib:IDPProvidedNameIdentifier>`) when communicating with it about the Principal.

Two profiles are specified: HTTP-Redirect-Based and SOAP/HTTP-based.

Either the identity or service provider may initiate the register name identifier protocol. The available profiles are defined in Section 3.3.1 and Section 3.3.2, and vary slightly based on whether the protocol was initiated by the identity or service provider:

- Register Name Identifier Initiated at Identity Provider
  - HTTP-Redirect-Based: Relies on an HTTP 302 redirect to communicate between the identity provider and the service provider.
  - SOAP/HTTP-Based: Relies on a SOAP call from the identity provider to the service provider.
- Register Name Identifier Initiated at Service Provider
  - HTTP-Redirect-Based: Relies on an HTTP 302 redirect to communicate between the service provider and the identity provider.
  - SOAP/HTTP-Based: Relies on a SOAP call from the service provider to the identity provider.

The interactions and processing rules for the SOAP/HTTP-based and HTTP-redirect-based profiles are essentially the same regardless of whether the profile was initiated at the service provider or at the identity provider, but the message flow directions are reversed.

The register name identifier profiles make use of the following metadata elements, as defined in [LibertyMetadata]:

- `RegisterNameIdentifierProtocolProfile`: The service provider's preferred register name identifier profile, which should be used by the identity provider when registering a new identifier. This would specify the URI based identifier for one of the IDP Initiated register name identifier profiles.

- `RegisterNameIdentifierServiceURL`: The URL used for user-agent-based Register Name Identifier Protocol profiles.

- `RegisterNameIdentifierServiceReturnURL`: The provider's redirecting URL for use after HTTP name registration has taken place.

- `SOAPEndpoint`: The SOAP endpoint location at the service provider or identity provider to which Liberty SOAP messages are sent.

3.3.1. Register Name Identifier Initiated at Identity Provider

An identity provider MAY change the `<lib:IDPProvidedNameIdentifier>` it has assigned a Principal and transmit that information to a service provider. The `<lib:IDPProvidedNameIdentifier>` MAY be changed without changing any federations. The reasons an identity provider may wish to change the name identifier for a Principal are implementation dependent, and thus outside the scope of this specification. Changing the `<lib:IDPProvidedNameIdentifier>` MAY be accomplished in either an HTTP-Redirect-Based or SOAP/HTTP mode.

3.3.1.1. HTTP-Redirect-Based Profile

A HTTP-redirect-based register name identifier profile cannot be self-initiated by an identity provider, but must be a triggered by a message, such as an `<lib:AuthnRequest>`. We note that we do not normatively specify when and how the identity provider can initiate this profile - that is left to the discretion of the identity provider. As an example, it may be triggered by a message, such as an `<lib:AuthnRequest>`. When the identity provider decides to initiate the profile in this case, it will insert this profile between the `AuthnRequest/AuthnResponse` transactions.

The HTTP-redirect-based profile relies on using HTTP 302 redirects to communicate register name identifier messages from the identity provider to the service provider. The HTTP-Redirect Register Name Identifier Profile (Figure 6) illustrates this transaction.

The following URI-based identifier MUST be used when referencing this specific profile:

URI: `http://projectliberty.org/profiles/rni-idp-http`

This URI identifier MUST be specified in the service provider metadata element `RegisterNameIdentifierProtocolProfile` when the service provider intends to indicate to the identity provider a preference for receiving register name identifier messages via an HTTP 302 redirect.

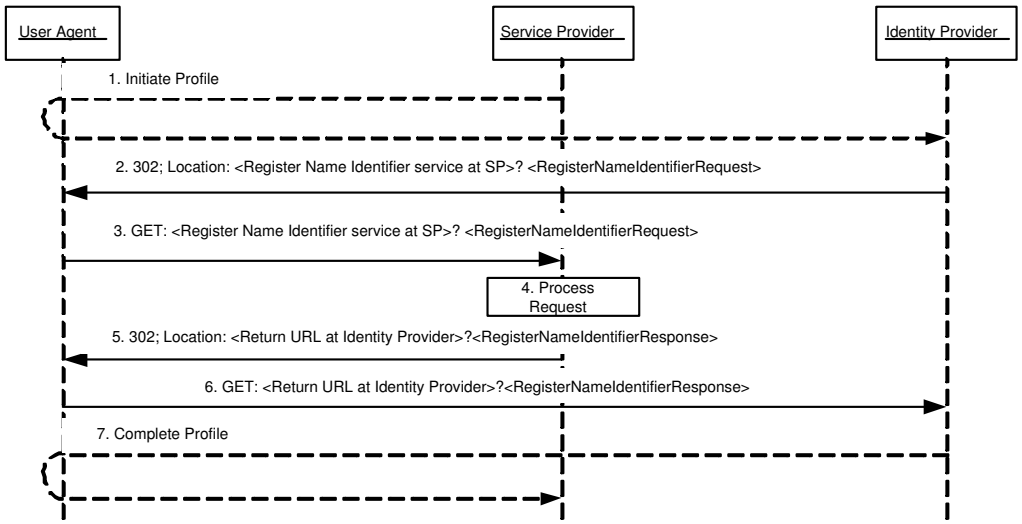


Figure 6. Register Name Identifier Profile.

In an example scenario, the service provider makes an `<lib:AuthnRequest>` to the identity provider for authentication of the Principal's User Agent (step 1). The identity provider effects an `<lib:IDPProvidedNameIdentifier>` change in the service provider via a URL redirection. The profile is as follows:

3.3.1.1.1. Step 1: Initiate Profile

This interaction is not normatively specified as part of the profile, but shown for illustrative purposes.

3.3.1.1.2. Step 2: Redirecting to the Service Provider Register Name Identifier Service

In step 2, the identity provider redirects the user agent to the register name identifier service at the service provider.

The redirection MUST adhere to the following rules:

- The Location HTTP header MUST be set to the service provider's register name identifier service URL.
- The service provider's register name identifier service URL MUST specify `https` as the URL scheme; if another scheme is specified, the identity provider MUST NOT redirect to the service provider.
- The Location HTTP header MUST include a `<query>` component containing the `<lib:RegisterNameIdentifierRequest>` protocol message as defined in [LibertyProtSchema] with formatting as specified in Section 3.1.2.

The HTTP response MUST take the following form:

```
<HTTP-Version> 302 <Reason Phrase>
<other headers>
Location : https://<Service Provider Register Name Identifier service URL>?<query>
<other HTTP 1.0 or 1.1 components>
```

where:

`<Service Provider Register Name Identifier service URL>`

This element provides the host name, port number, and path components of the register name identifier service URL at the service provider.

`<query>= ...<URL-encoded RegisterNameIdentifierRequest>...`

The <query> component MUST contain a single register name identifier request.

3.3.1.1.3. Step 3: Accessing the Service Provider Register Name Identifier Service

In step 3, the user agent accesses the service provider's register name identifier service URL with the `<lib:RegisterNameIdentifierRequest>` information attached to the URL fulfilling the redirect request.

3.3.1.1.4. Step 4: Processing the Register Name Identifier Request

In step 4, the service provider MUST process the `<lib:RegisterNameIdentifierRequest>` according to the rules defined in [LibertyProtSchema].

The service provider MAY remove the old name identifier after registering the new name identifier.

3.3.1.1.5. Step 5: Redirecting to the Identity Provider return URL with the Register Name Identifier Response

In step 5, the service provider's register name identifier service responds and redirects the user agent back to identity provider using a return URL location specified in the RegisterNameIdentifierServiceReturnURL metadata element. If the URL-encoded `<lib: RegisterNameIdentifierRequest>` message received in step 3 contains a parameter named RelayState, then the service provider MUST include a `<query>` component containing the same RelayState parameter and its value in its response to the identity provider.

The redirection MUST adhere to the following rules:

- The Location HTTP header MUST be set to the identity providers return URL specified in the RegisterNameIdentifierServiceReturnURL metadata element.
- The identity provider's return URL MUST specify `https` as the URL scheme; if another scheme is specified, the service provider MUST NOT redirect to the identity provider.
- The Location HTTP header MUST include a `<query>` component containing the `<lib:RegisterNameIdentifierResponse>` protocol message as defined in [LibertyProtSchema] with formatting as specified in Section 3.1.2.

The HTTP response MUST take the following form:

```
<HTTP-Version> 302 <Reason Phrase>
<other headers>
Location : https://<Identity Provider Service Return URL >?<query>
<other HTTP 1.0 or 1.1 components>
```

where:

`<Identity Provider Service Return URL>`

This element provides the host name, port number, and path components of the return URL at the identity provider.

`<query>= ...<URL-encoded RegisterNameIdentifierResponse>...`

The `<query>` component MUST contain a single register name identifier response. The `<URL-encoded RegisterNameIdentifierResponse>` component MUST contain the identical RelayState parameter and its value that was received in the URL-encoded register name identifier message obtained in step 3. If no RelayState parameter was provided in the step 3 message, then a RelayState parameter MUST NOT be specified in the `<URL-encoded RegisterNameIdentifierResponse>`.

3.3.1.1.6. Step 6: Accessing the Identity Provider return URL with the Register Name Identifier Response

In step 6, the user agent accesses the identity provider's return URL location fulfilling the redirect request.

3.3.1.1.7. Step 7: Complete profile

This concludes the initial sequence, which triggered the initiation of this profile.

3.3.1.2. SOAP/HTTP-Based Profile

The following URI-based identifier MUST be used when referencing this specific profile:

URI: `http://projectliberty.org/profiles/rni-idp-soap`

This URI identifier MUST be specified in the service provider metadata element RegisterNameIdentifierProtocolProfile when the service provider intends to indicate to the identity provider a preference for receiving register name identifier messages via SOAP over HTTP.

The steps involved in the SOAP/HTTP-based profile MUST utilize the SOAP binding for Liberty as defined in Section 2.1. See Figure 7.

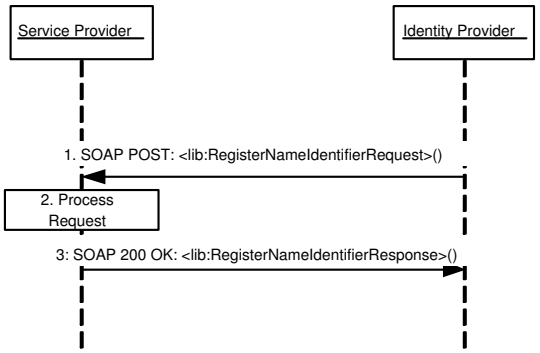


Figure 7. SOAP/HTTP-based profile for registering name identifiers

3.3.1.2.1. Step 1 Initiate Profile

In step 1, the identity provider sends a `<lib:RegisterNameIdentifierRequest>` protocol message to the service provider's SOAP endpoint specifying `<lib:SPProvidedNameIdentifier>`, `<lib:IDPProvidedNameIdentifier>`, and `<lib:OldProvidedNameIdentifier>` as defined in [LibertyProtSchema]. The `<lib:SPProvidedNameIdentifier>` will only contain a value if the service provider has previously used the register name identifier profile.

3.3.1.2.2. Step 2: Process Request

Service provider records new `<lib:IDPProvidedNameIdentifier>`.

3.3.1.2.3. Step 3: Response to Register Name Identifier

The service provider, after successfully registering the new `<lib:IDPProvidedNameIdentifier>` provided by the identity provider, MUST respond with a `<lib:RegisterNameIdentifierResponse>` according to the processing rules defined in [LibertyProtSchema].

3.3.2. Register Name Identifier Initiated at Service Provider

A service provider may register, or change a `<lib:SPProvidedNameIdentifier>` which is a name identifier it expects the identity provider to use when communicating with it about the Principal. Until it registers a `<lib:SPProvidedNameIdentifier>`, an identity provider will continue to use the current `<lib:IDPProvidedNameIdentifier>` when referring to the Principal.

3.3.2.1. HTTP-Redirect-Based Profile

The HTTP-redirect-based profile relies on the use of an HTTP 302 redirect to communicate a register name identifier message from the service provider to the identity provider.

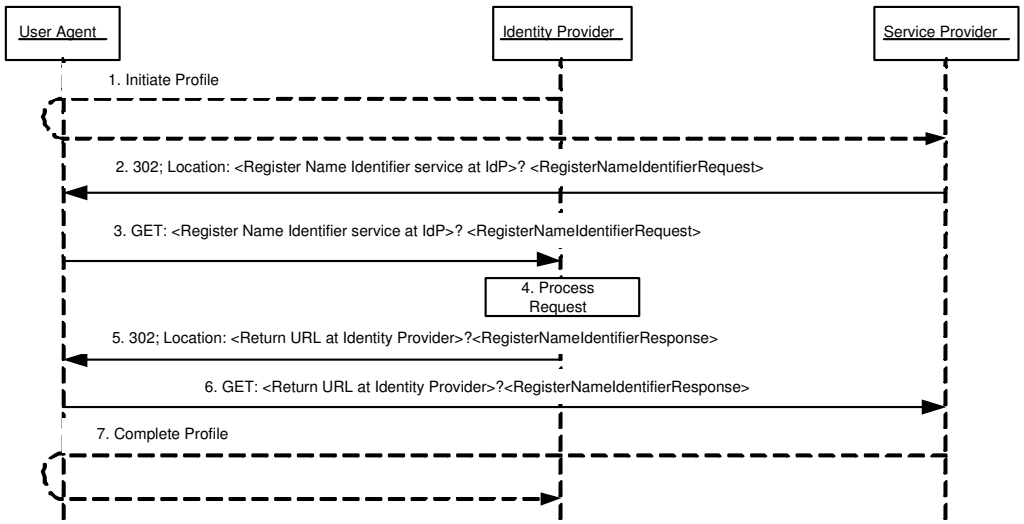


Figure 8. SP-Initiated Register Name Identifier Profile.

The following URI-based identifier MUST be used when referencing this specific profile:

URI: `http://projectliberty.org/profiles/rni-sp-http`

A HTTP-redirect-based register name identifier profile can be self-initiated by a service provider to change the `<lib:SPProvidedNameIdentifier;>`. This does not normatively specify when and how the service provider can initiate this profile; that is left to the discretion of the service provider. The HTTP-redirect-based profile relies on

using HTTP 302 redirects to communicate register name identifier messages from the service provider to the identity provider. The service provider effects a `<lib:SPProvidedNameIdentifier>` change in the identity provider via a URL redirection. For a discussion of the interactions and processing steps, refer to Section 3.3.1.1. When reviewing that profile, interchange all references to service provider and identity provider in the interaction diagram and processing steps 2-6. See Figure 8. Note that in step 4 the old SPProvidedNameIdentifier SHOULD be removed at the IdP.

3.3.2.2. SOAP/HTTP-Based Profile

The SOAP/HTTP-based profile relies on using SOAP over HTTP to communicate register name identifier messages from the service provider to the identity provider. For a discussion of the interactions and processing steps, refer to Section 3.3.1.2. When reviewing that profile, interchange all references to service provider and identity provider in the interaction diagram and processing steps. See Figure 9.

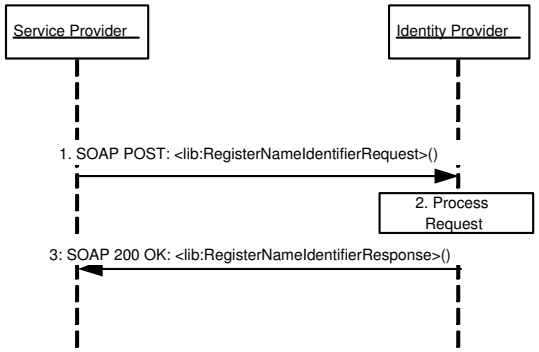


Figure 9. SP-Initiated SOAP/HTTP-based profile for registering name identifiers

The following URI-based identifier MUST be used when referencing this specific profile:

URI: `http://projectliberty.org/profiles/rni-sp-soap`

In step 1, the service provider sends a `<lib:RegisterNameIdentifierRequest>` protocol message to the identity provider's SOAP endpoint specifying `<lib:SPProvidedNameIdentifier>`, `<lib:IDPProvidedNameIdentifier>`, and `<lib:OldProvidedNameIdentifier>` as defined in [LibertyProtSchema]. The `<lib:OldProvidedNameIdentifier>` will only contain a value if the service provider has previously used the register name identifier profile.

3.4. Identity Federation Termination Notification Profiles

The Liberty identity federation termination notification profiles specify how service providers and identity providers are notified of federation termination (also known as defederation).

Note:

Other means of federation termination are possible, such as federation expiration and termination of business agreements between service providers and identity providers. These means of federation termination are outside the scope of this specification.

Identity federation termination can be initiated at either the identity provider or the service provider. The Principal SHOULD have been authenticated by the provider at which identity federation termination is being initiated. The available profiles are defined in Section 3.4.1 and Section 3.4.2, depending on whether the identity federation termination notification process was initiated at the identity provider or service provider:

- Federation Termination Notification Initiated at Identity Provider

- HTTP-Redirect-Based: Relies on an HTTP 302 redirect to communicate between the identity provider and the service provider.
- SOAP/HTTP-Based: Relies on a SOAP call from the identity provider to the service provider.

- Federation Termination Notification Initiated at Service Provider

  - HTTP-Redirect-Based: Relies on an HTTP 302 redirect to communicate between the service provider and the identity provider.
  - SOAP/HTTP-Based: Relies on a SOAP call from the service provider to the identity provider.

The interactions and processing rules for the SOAP/HTTP-based and HTTP-redirect-based profiles are essentially the same regardless of whether federation termination notification was initiated at the service provider or at the identity provider.

The identity federation termination notification profiles make use of the following metadata elements, as defined in [LibertyMetadata]:

- `FederationTerminationServiceURL` - The URL at the service provider or identity provider to which identity federation termination notifications are sent. It is documented in these profiles as "federation termination service URL."
- `FederationTerminationServiceReturnURL` - The URL used by the service provider or identity provider when redirecting the user agent at the end of the federation termination notification profile process.
- `FederationTerminationNotificationProtocolProfile` - Used by the identity provider to determine which federation termination notification profile MUST be used when communicating with the service provider.
- `SOAPEndpoint` - The SOAP endpoint location at the service provider or identity provider to which Liberty SOAP messages are sent.

3.4.1. Federation Termination Notification Initiated at Identity Provider

The profiles in Section 3.4.1.1 and Section 3.4.1.2 are specific to identity federation termination when initiated at the identity provider. Effectively, when using these profiles, the identity provider is stating to the service provider that it will no longer provide the Principal's identity information to the service provider and that the identity provider will no longer respond to any requests by the service provider on behalf of the Principal.

3.4.1.1. HTTP-Redirect-Based Profile

The HTTP-redirect-based profile relies on using HTTP 302 redirect to communicate federation termination notification messages from the identity provider to the service provider. See Figure 10.

The following URI-based identifier MUST be used when referencing this specific profile:

URI: `http://projectliberty.org/profiles/fedterm-idp-http`

This URI identifier MUST be specified in the service provider metadata element FederationTerminationNotification-ProtocolProfile when the service provider intends to indicate to the identity provider a preference for receiving federation termination notifications via an HTTP 302 redirect.

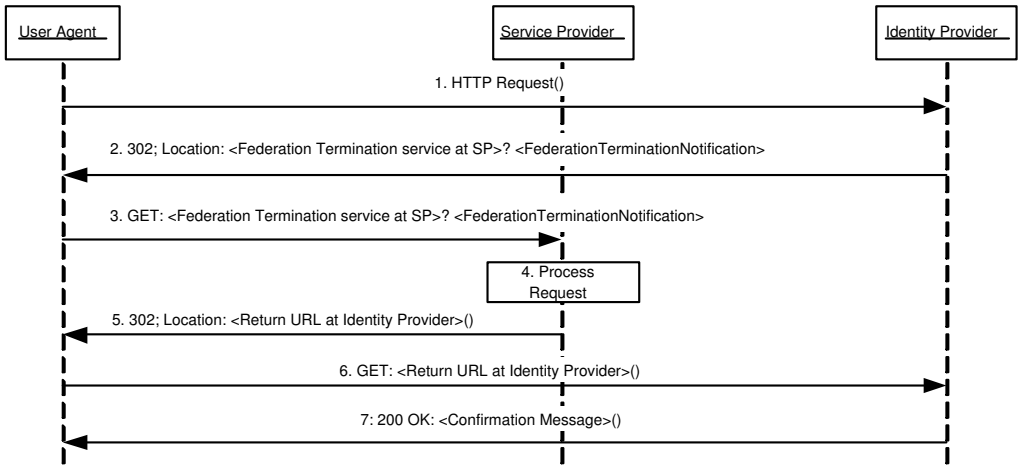


Figure 10. HTTP-redirect-based profile for federation termination

This profile description assumes the following preconditions:

- The Principal's identity at the service provider is federated with his/her identity at the identity provider.
- The Principal has requested to the identity provider that the federation be terminated.
- The Principal has authenticated with the identity provider.

3.4.1.1.1. Step 1: Accessing the Federation Termination Service

In step 1, the user agent accesses the identity federation termination service URL at the identity provider specifying the service provider with which identity federation termination should occur. How the service provider is specified is implementation-dependent and, as such, is out of the scope of this specification.

3.4.1.1.2. Step 2: Redirecting to the Service Provider

In step 2, the identity provider's federation termination service URL responds and redirects the user agent to the federation termination service at the service provider.

The redirection MUST adhere to the following rules:

- The Location HTTP header MUST be set to the service provider's federation termination service URL.
- The service provider's federation termination service URL MUST specify https as the URL scheme; if another scheme is specified, the identity provider MUST NOT redirect to the service provider.
- The Location HTTP header MUST include a `<query>` component containing the `<lib:FederationTerminationNotification>` protocol message as defined in [LibertyProtSchema] with formatting as specified in Section 3.1.2.

The HTTP response MUST take the following form:

```
<HTTP-Version> 302 <Reason Phrase>
<other headers>
Location : https://<Service Provider Federation Termination service URL>?<query>
<other HTTP 1.0 or 1.1 components>
```

where:

`<Service Provider Federation Termination service URL>`

This element provides the host name, port number, and path components of the federation termination service URL at the service provider.

`<query>= ...<URL-encoded FederationTerminationNotification>...`

The `<query>` component MUST contain a single terminate federation request.

3.4.1.1.3. Step 3: Accessing the Service Provider Federation Termination Service

In step 3, the user agent accesses the service provider's federation termination service URL with the `<lib:FederationTerminationNotification>` information attached to the URL fulfilling the redirect request.

3.4.1.1.4. Step 4: Processing the Notification

In step 4, the service provider MUST process the `<lib:FederationTerminationNotification>` according to the rules defined in [LibertyProtSchema].

The service provider MAY remove any locally stored references to the name identifier it received from the identity provider in the `<lib:FederationTerminationNotification>`.

3.4.1.1.5. Step 5: Redirecting to the Identity Provider Return URL

In step 5, the service provider's federation termination service responds and redirects the user agent back to identity provider using a return URL location specified in the FederationTerminationServiceReturnURL metadata element. If the URL-encoded `<lib:FederationTerminationNotification>` message received in step 3 contains a parameter named RelayState, then the service provider MUST include a `<query>` component containing the same RelayState parameter and its value in its response to the identity provider.

No success or failure message should be conveyed in this HTTP redirect. The sole purpose of this redirect is to return the user agent to the identity provider where the federation termination process began.

The HTTP response MUST take the following form:

```
<HTTP-Version> 302 <Reason Phrase>
<other headers>
Location : https://<Identity Provider Service Return URL >?<query>
<other HTTP 1.0 or 1.1 components>
```

where:

`<Identity Provider Service Return URL>`

This element provides the components of the return URL at the identity provider.

1544 `<query>= . . .RelayState=<. . .>`

1545 The `<query>` component MUST contain the identical RelayState parameter and its value that was received in the
1546 URL-encoded federation termination message obtained in step 3. If no RelayState parameter was provided in the step
1547 3 message, then a RelayState parameter MUST NOT be specified in the `<query>` component.

1548 #### 3.4.1.1.6. Step 6: Accessing the Identity Provider Return URL

1549 In step 6, the user agent accesses the identity provider's return URL location fulfilling the redirect request.

1550 #### 3.4.1.1.7. Step 7: Confirmation

1551 In step 7, the user agent is sent an HTTP response that confirms the requested action of identity federation termination
1552 with the specific service provider.

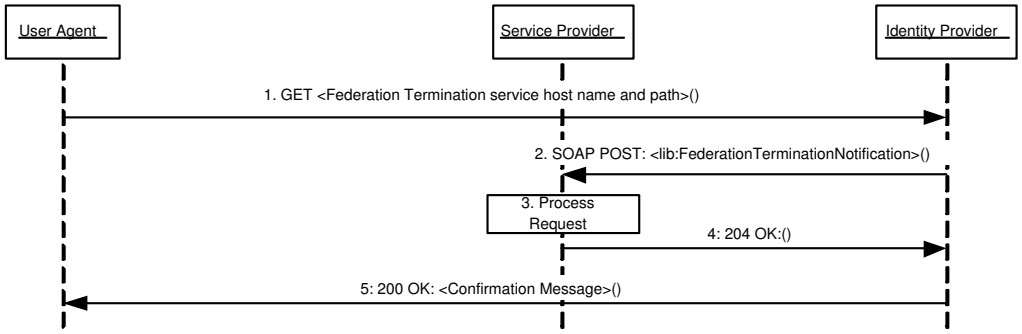
1553 ### 3.4.1.2. SOAP/HTTP-Based Profile

1554 The SOAP/HTTP-based profile relies on using asynchronous SOAP over HTTP to communicate federation termination
1555 notification messages from the identity provider to the service provider. See Figure 11.

1556 The following URI-based identifier MUST be used when referencing this specific profile:

1557 URI: `http://projectliberty.org/profiles/fedterm-idp-soap`

1558 This URI identifier MUST be specified in the service provider metadata element FederationTerminationNotification-
1559 ProtocolProfile when the service provider intends to indicate to the identity provider a preference for receiving feder-
1560 ation termination notifications via SOAP over HTTP.



1561

1562 **Figure 11. SOAP/HTTP-based profile for federation termination**

1563 This profile description assumes the following preconditions:

1564 • The Principal's identity at the service provider is federated with his/her identity at the identity provider.

1565 • The Principal has authenticated with the identity provider.

1566 • The Principal has requested that the identity provider terminate the federation.

3.4.1.2.1. Step 1: Accessing the Federation Termination Service

In step 1, the user agent accesses the identity federation termination service URL at the identity provider specifying the service provider for with which identity federation termination should occur. How the service provider is specified is implementation-dependent and, as such, is out of the scope of this specification.

3.4.1.2.2. Step 2: Notification of Federation Termination

In step 2, the identity provider sends an asynchronous SOAP over HTTP notification message to the service provider's SOAP endpoint. The SOAP message MUST contain exactly one `<lib:FederationTerminationNotification>` element in the SOAP body and adhere to the construction rules defined in [LibertyProtSchema].

If a SOAP fault occurs, the identity provider SHOULD employ best effort to resolve the fault condition and resend the federation termination notification message to the service provider.

3.4.1.2.3. Step 3: Processing the Notification

In step 3, the service provider MUST process the `<lib:FederationTerminationNotification>` according to the rules defined in [LibertyProtSchema].

The service provider MAY remove any locally stored references to the name identifier it received from the identity provider in the `<lib:FederationTerminationNotification>`.

3.4.1.2.4. Step 4: Responding to the Notification

In step 4, the service provider MUST respond to the `<lib:FederationTerminationNotification>` with an HTTP 204 OK response.

3.4.1.2.5. Step 5: Confirmation

In step 5, the user agent is sent an HTTP response that confirms the requested action of identity federation termination with the specific service provider.

3.4.2. Federation Termination Notification Initiated at Service Provider

The profiles in Section 3.4.2.1 and Section 3.4.2.2 are specific to identity federation termination notification when initiated by a Principal at the service provider. Effectively, when using this profile, the service provider is stating to the identity provider that the Principal has requested that the identity provider no longer provide the Principal's identity information to the service provider and that service provider will no longer ask the identity provider to do anything on the behalf of the Principal.

It is RECOMMENDED that the service provider, after initiating or receiving a federation termination notification, invalidate the local session for the Principal that was authenticated at the identity provider with which federation has been terminated. If the Principal was locally authenticated at the service provider, the service provider MAY continue to maintain a local session for the Principal. If the Principal wants to engage in a single sign-on session with identity provider again, the service provider MUST first federate with identity provider the given Principal.

3.4.2.1. HTTP-Redirect-Based Profile

The HTTP-redirect-based profile relies on the use of an HTTP 302 redirect to communicate a federation termination notification message from the service provider to the identity provider. For a discussion of the interactions and processing steps, refer to Section 3.4.1.1. When reviewing that profile, interchange all references to service provider and identity provider in the interaction diagram and processing steps.

The following URI-based identifier MUST be used when referencing this specific profile:

URI: `http://projectliberty.org/profiles/fedterm-sp-http`

This URI identifier is really only meant for service provider consumption and as such is not needed in any provider metadata.

3.4.2.2. SOAP/HTTP-Based Profile

The SOAP/HTTP-based profile relies on using asynchronous SOAP over HTTP to communicate federation termination notification messages from the service provider to the identity provider. For a discussion of the interactions and processing steps, refer to 3.4.1.2. When reviewing that profile, interchange all references to service provider and identity provider in the interaction diagram and processing steps.

The following URI-based identifier MUST be used when referencing this specific profile:

URI: `http://projectliberty.org/profiles/fedterm-sp-soap`

This URI identifier is really only meant for service provider consumption and as such is not needed in any provider metadata.

3.5. Single Logout Profiles

The single logout profiles synchronize session logout functionality across all sessions that were authenticated by a particular identity provider. The single logout can be initiated at either the identity provider or the service provider. In either case, the identity provider will then communicate a logout request to each service provider with which it has established a session for the Principal. The negotiation of which single logout profile the identity provider uses to communicate with each service provider is based upon the SingleLogoutProtocolProfile provider metadata element defined in [LibertyProtSchema].

The available profiles are defined in Section 3.5.1 and Section 3.5.2, depending on whether the single logout is initiated at the identity provider or service provider:

- *Single Logout Initiated at Identity Provider*
  - HTTP-Based: Relies on using either HTTP 302 redirects or HTTP GET requests to communicate logout requests from an identity provider to the service providers.
  - SOAP/HTTP-Based: Relies on SOAP over HTTP messaging to communicate logout requests from an identity provider to the service providers.
- *Single Logout Initiated at Service Provider*
  - HTTP-Redirect-Based: Relies on an HTTP 302 redirect to communicate a logout request with the identity provider.
  - SOAP/HTTP-Based: Relies on SOAP over HTTP messaging to communicate a logout request from a service provider to an identity provider.

The single logout profiles make use of the following metadata elements, as defined in [LibertyMetadata]:

- `SingleLogoutServiceURL` — The URL at the service provider or identity provider to which single logout requests are sent. It is described in these profiles as "single logout service URL."

- `SingleLogoutServiceReturnURL`— The URL used by the service provider when redirecting the user agent to the identity provider at the end of the single logout profile process.
- `SingleLogoutProtocolProfile` — Used by the identity provider to determine which single logout request profile MUST be used when communicating with the service provider.
- `SOAPEndpoint` — The SOAP endpoint location at the service provider or identity provider to which Liberty SOAP messages are sent.

3.5.1. Single Logout Initiated at Identity Provider

The profiles in 3.5.1.1 through 3.5.1.2 are specific to a single logout when initiated by a user agent at the identity provider.

3.5.1.1. HTTP-Based Profile

The HTTP-based profile defines two possible implementations that an identity provider may use. The first implementation relies on using HTTP 302 redirects, while the second uses HTTP GET requests. The choice of implementation is entirely dependent upon the type of user experience the identity provider provides.

The following URI-based identifier MUST be used when referencing either implementation for this specific profile:

URI: `http://projectliberty.org/profiles/slo-idp-http`

This URI identifier MUST be specified in the service provider metadata element SingleLogoutProtocolProfile when the service provider intends to indicate to the identity provider a preference for receiving logout requests via either an HTTP redirect or an HTTP GET.

3.5.1.1.1. HTTP-Redirect Implementation

The HTTP-Redirect implementation uses HTTP 302 redirects to communicate a logout request to each service provider for which the identity provider has provided authentication assertions during the Principal's current session if the service provider indicated a preference to receive logout requests via the HTTP based profile. See Figure 12.

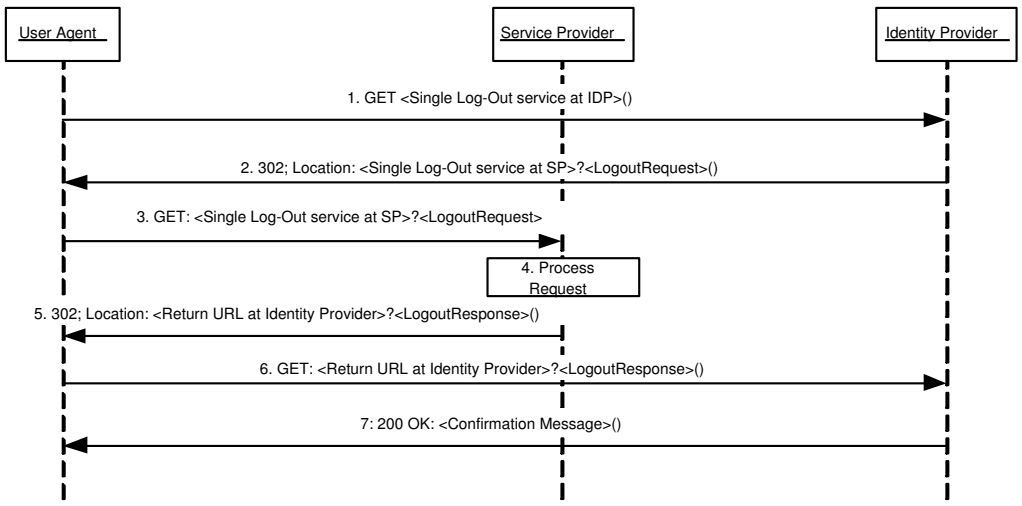


Figure 12. HTTP-Redirect implementation for single logout initiated at identity provider

Notes:

Steps 2 through 6 may be an iterative process for requesting logouts by each service provider that has been issued authentication assertions during the Principal's current session and has indicated a preference to receive logout requests via the HTTP based profile.

[RFC2616] indicates a client should detect infinite redirection loops because such loops generate network traffic for each redirection. This requirement was introduced because previous versions of the specification recommended a maximum of five redirections. Content developers should be aware that some clients might implement such a fixed limitation.

3.5.1.1.1.1. Step 1: Accessing the Single Logout Service at the Identity Provider

In step 1, the user agent accesses the single logout service URL at the identity provider indicating that all service providers for which this identity provider has provided authentication assertions during the Principal's current session must be notified of session termination.

3.5.1.1.1.2. Step 2: Redirecting to the Single Logout Service at the Service Provider

In step 2, the identity provider's single logout service responds and redirects the user agent to the single logout service URL at each service provider for which the identity provider has provided an authentication assertion during the Principal's current session with the identity provider.

The redirections MUST adhere to the following rules:

- The Location HTTP header MUST be set to the service provider's single logout service URL.
- The service provider's single logout service URL MUST specify https as the URL scheme; if another scheme is specified, the identity provider MUST NOT redirect to the service provider.
- The Location HTTP header MUST include a `<query>` component containing the `<lib:LogoutRequest>` protocol message as defined in [LibertyProtSchema] with formatting as specified in 3.1.2.

The HTTP response MUST take the following form:

```
<HTTP-Version> 302 <Reason Phrase>
<other headers>
Location : https://<Service Provider Single Log-Out service URL>?<query>
<other HTTP 1.0 or 1.1 components>
```

where:

`<Service Provider Single Log-Out service URL>`

This element provides the host name, port number, and path components of the single logout service URL at the service provider.

`<query>= ...<URL-encoded LogoutRequest>...`

The `<query>` MUST contain a single logout request.

3.5.1.1.1.3. Step 3: Accessing the Service Provider Single Logout Service

In step 3, the user agent accesses the service provider's single logout service URL with the `<lib:LogoutRequest>` information attached to the URL fulfilling the redirect request.

3.5.1.1.1.4. Step 4: Processing the Request

In step 4, the service provider MUST process the `<lib:LogoutRequest>` according to the rules defined in [LibertyProtSchema].

The service provider MUST invalidate the session(s) of the Principal referred to in the name identifier it received from the identity provider in the <lib:LogoutRequest>.

3.5.1.1.1.5. Step 5: Redirecting to the Identity Provider Return URL

In step 5, the service provider's single logout service responds and redirects the user agent back to the identity provider using the return URL location obtained from the SingleLogoutServiceReturnURL metadata element. If the URL-encoded <lib:LogoutRequest> message received in step 3 contains a parameter named RelayState, then the service provider MUST include a <query> component containing the same RelayState parameter and its value in its response to the identity provider.

The purpose of this redirect is to return the user agent to the identity provider so that the single logout process may continue in the same fashion with other service providers.

The HTTP response MUST take the following form:

```
<HTTP-Version> 302 <Reason Phrase>
<other headers>
Location : https://<Identity Provider Service Return URL>?<query>
<other HTTP 1.0 or 1.1 components>
```

where:

`<Identity Provider Service Return URL>`

This element provides the host name, port number, and path components of the return URL at the identity provider.

`<query>= ...<URL-encoded LogoutResponse>`

The <query> component MUST contain a single logout response. The <URL-encoded LogoutResponse> MUST contain the identical RelayState parameter and its value that was received in the URL-encoded logout request message obtained in step 3. If no RelayState parameter was provided in the step 3 message, then a RelayState parameter MUST NOT be specified in the <URL-encoded LogoutResponse>.

3.5.1.1.1.6. Step 6: Accessing the Identity Provider Return URL

In step 6, the user agent accesses the identity provider's return URL location fulfilling the redirect request.

3.5.1.1.1.7. Step 7: Confirmation

In step 7, the user agent is sent an HTTP response that confirms the requested action of a single logout has been completed.

3.5.1.1.2. HTTP-GET Implementation

The HTTP-GET implementation uses HTTP GET requests to communicate logout requests to each service provider for which the identity provider has provided authentication during the Principal's current session if the service provider indicated a preference to receive logout requests via the HTTP based profile. See Figure 13.

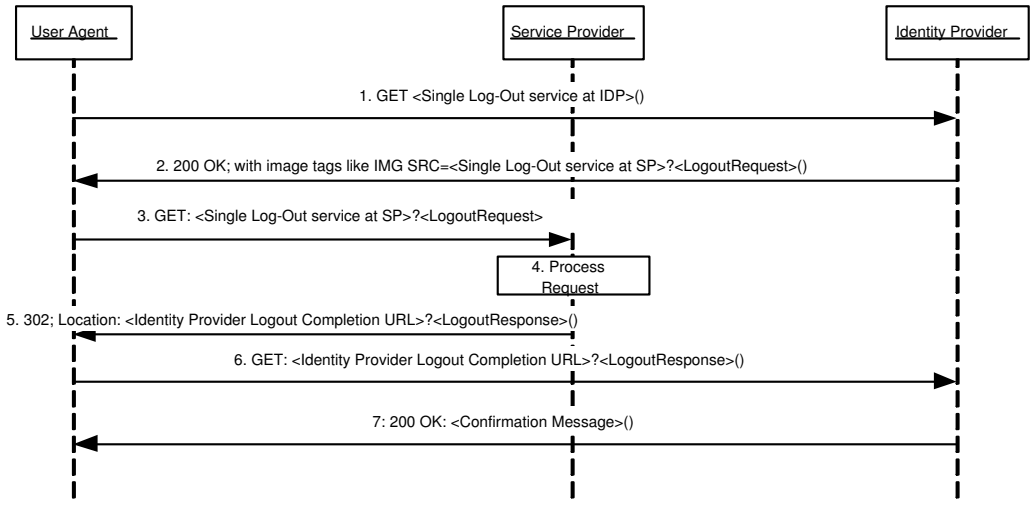


Figure 13. HTTP-GET implementation for single logout initiated at identity provider

Note:

Steps 3 through 7 may be an iterative process for requesting logout of each service provider that has been issued authentication assertions during the Principal's current session and has indicated a preference to receive logout requests via the HTTP based profile.

3.5.1.1.2.1. Step 1: Accessing the Single Logout Service at the Identity Provider

In step 1, the user agent accesses the single logout service URL at the identity provider indicating that all service providers for which this identity provider has provided authentication assertions during the Principal's current session must be notified of session termination and requested to logout the Principal.

3.5.1.1.2.2. Step 2: HTML Page Returned to User Agent with Image Tags

In step 2, the identity provider's single logout service responds with an HTML page that includes image tags referencing the logout service URL for each of the service providers for which the identity provider has provided an authentication assertion during the Principal's current session. The list of image tags MUST be sent in a standard HTTP 200 response to the user agent.

The image tag loads on the HTML page MUST adhere to the following rules:

- The SRC attribute MUST be set to the specific service provider's single logout service URL.
- The service provider's single logout service URL MUST specify https as the URL scheme.
- The service provider's single logout service URL MUST include a `<query>` component containing the `<lib:LogoutRequest>` protocol message as defined in [LibertyProtSchema] with formatting as specified in 3.1.2.

3.5.1.1.2.3. Step 3: Accessing the Service Provider Single Logout Service

In step 3, the user agent, as a result of each image load, accesses the service provider's single logout service URL with `<lib:LogoutRequest>` information attached to the URL. This step may occur multiple times if the HTTP response includes multiple image tag statements (one for each service provider that has been issued authentication assertions during the Principal's current session).

3.5.1.1.2.4. Step 4: Processing the Request

In step 4, the service provider MUST process the `<lib:LogoutRequest>` according to the rules defined in [LibertyProtSchema].

The service provider MUST invalidate the session of the Principal referred to in the name identifier it received from the identity provider in the `<lib:LogoutRequest>`.

3.5.1.1.2.5. Step 5: Redirecting to the Identity Provider Logout Completion URL

In step 5, the service provider's single logout service responds and redirects the image load back to the identity provider's logout completion URL. This location will typically point to an image that will be loaded by the user agent to indicate that the logout is complete (for example, a checkmark).

The logout completion URL is obtained from the SingleLogoutServiceReturnURL metadata element.

The HTTP response MUST take the following form:

```
<HTTP-Version> 302 <Reason Phrase>
<other headers>
Location : https://<Identity Provider Logout Completion URL>?<query>
<other HTTP 1.0 or 1.1 components>
```

where:

`<Identity Provider Logout Completion URL>`

This element provides the host name, port number, and path components of the identity provider logout completion URL at the identity provider.

`<query>=...<URL-encoded LogoutResponse>`

The `<query>` component MUST contain a single logout response. The `<URL-encoded LogoutResponse>` component MUST contain the identical RelayState parameter and its value that was received in the URL-encoded logout request message obtained in step 3. If no RelayState parameter was provided in step 3 then a RelayState message MUST NOT be specified in the `<URL-encoded LogoutResponse>`.

3.5.1.1.2.6. Step 6: Accessing the Identity Provider Logout Completion URL

In step 6, the user agent accesses the identity provider's logout completion URL fulfilling the redirect request.

3.5.1.1.2.7. Step 7: Confirmation

In step 7, the user agent is sent an HTTP response that confirms the requested action of a single logout has been completed.

Note:

One method for seamlessly returning the user agent back to the identity provider is for the HTML page generated in step 2 to include a script that runs when the page is completely loaded (all logouts completed) that will initiate the redirect to the identity provider.

3.5.1.2. SOAP/HTTP-Based Profile

The SOAP/HTTP-based profile uses SOAP over HTTP messaging to communicate a logout request to each service provider for which the identity provider has provided authentication assertions during the Principal's current session if the service provider indicated a preference to receive logout request via the SOAP/HTTP-based profile. See Figure 14.

The following URI-based identifier MUST be used when referencing this specific profile:

URI: `http://projectliberty.org/profiles/slo-idp-soap`

This URI identifier MUST be specified in the service provider metadata element SingleLogoutProtocolProfile when the service provider intends to indicate to the identity provider a preference for receiving logout requests via SOAP over HTTP.

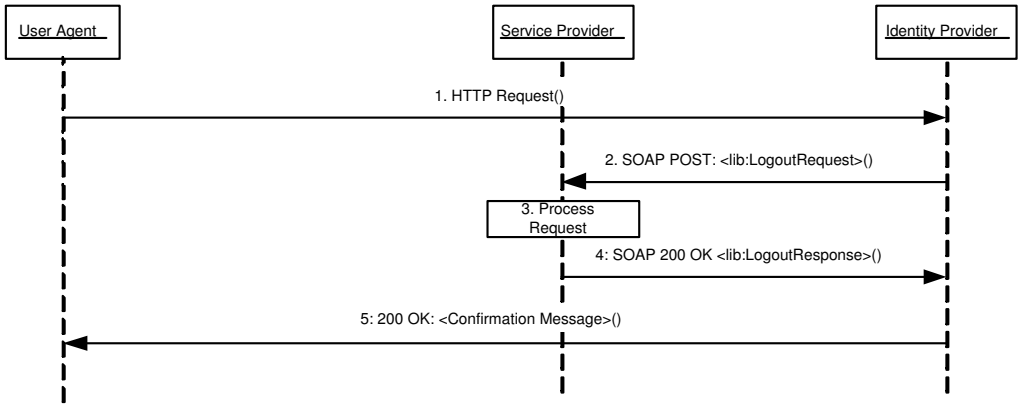


Figure 14. SOAP/HTTP-based profile for single logout initiated at identity provider

Note:

Steps 2 through 4 may be an iterative process for each service provider that has been issued authentication assertions during the Principal's current session and has indicated a preference to receive logout requests via the SOAP/HTTP message profile.

3.5.1.2.1. Step 1: Accessing the Single Logout Service

In step 1, the user agent accesses the single logout service URL at the identity provider via an HTTP request.

3.5.1.2.2. Step 2: Logout Request

In step 2, the identity provider sends a SOAP over HTTP request to the SOAP endpoint of each service provider for which it provided authentication assertions during the Principal's current session. The SOAP message MUST contain exactly one `<lib:LogoutRequest>` element in the SOAP body and adhere to the construction rules defined in [LibertyProtSchema].

If a SOAP fault occurs, the identity provider SHOULD employ best efforts to resolve the fault condition and resend the single logout request to the service provider.

3.5.1.2.3. Step 3: Processing the Logout Request

In step 3, the service provider MUST process the `<lib:LogoutRequest>` according to the rules defined in [LibertyProtSchema].

The service provider MUST invalidate the session for the Principal specified by the name identifier provided by the identity provider in the `<lib:LogoutRequest>`.

3.5.1.2.4. Step 4: Responding to the Request

In step 4, the service provider MUST respond to the `<lib:LogoutRequest>` with a SOAP 200 OK `<lib:LogoutResponse>` message.

3.5.1.2.5. Step 5: Confirmation

In step 5, the user agent is sent an HTTP response that confirms the requested action of single logout has completed.

3.5.2. Single Logout Initiated at Service Provider

The profiles in Section 3.5.2.1 and Section 3.5.2.2 are specific to the Principal' initiation of the single logout request process at the service provider.

3.5.2.1. HTTP-Based Profile

The HTTP-based profile relies on using an HTTP 302 redirect to communicate a logout request with the identity provider. The identity provider will then communicate a logout request to each service provider with which it has established a session for the Principal using the service provider' preferred profile for logout request from the identity provider (see Section 3.5.1). See Figure 15.

The following URI-based identifier MUST be used when referencing this specific profile:

URI: `http://projectliberty.org/profiles/slo-sp-http`

This URI identifier is intended for service provider consumption and is not needed in provider metadata.

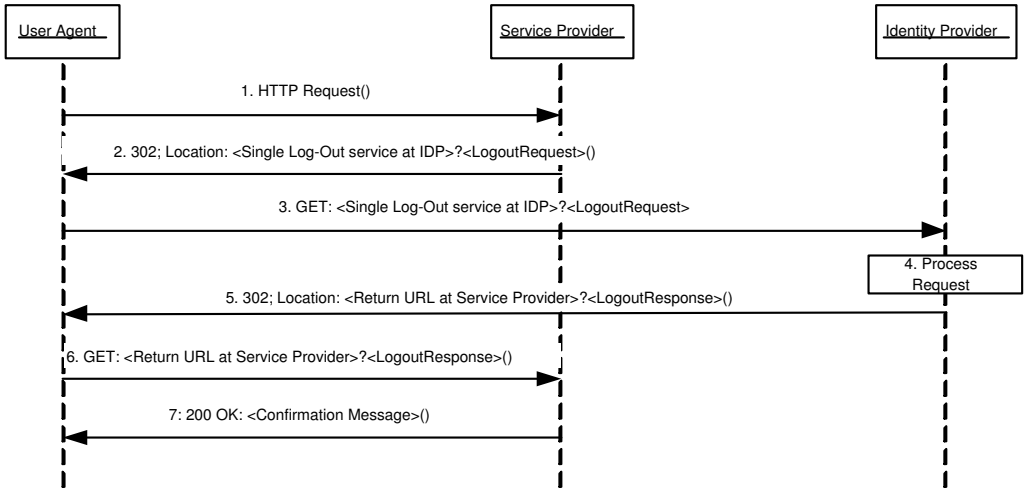


Figure 15. HTTP-redirect-based profile for single logout initiated at service provider

Note:

Step 4 may involve an iterative process by the identity provider to implement the preferred profile for logout requests for each service provider that has been issued authentication assertions during the Principal's current session.

3.5.2.1.1. Step 1: Accessing the Single Logout Service at the Service Provider

In step 1, the user agent accesses the single logout service URL at the service provider indicating that session logout is desired at the associated identity provider and all service providers for which this identity provider has provided

authentication assertions during the Principal's current session. If a current session exists for the Principal at the service provider, it is RECOMMENDED that the service provider terminate that session prior to step 2.

3.5.2.1.2. Step 2: Redirecting to the Single Logout Service at the Identity Provider

In step 2, the service provider's single logout service responds and redirects the user agent to the single logout service URL at the identity provider.

The redirection MUST adhere to the following rules:

- The Location HTTP header MUST be set to the identity provider's single logout service URL.
- The identity provider's single logout service URL MUST specify https as the URL scheme; if another scheme is specified, the service provider MUST NOT redirect to the identity provider.
- The Location HTTP header MUST include a `<query>` component containing the `<lib:LogoutRequest>` protocol message as defined in [LibertyProtSchema] with formatting as specified in 3.1.2.

The HTTP response MUST take the following form:

```
<HTTP-Version> 302 <Reason Phrase>
<other headers>
Location : https://<Identity Provider single log-out service URL>?<query>
<other HTTP 1.0 or 1.1 components>
```

where:

`<Identity Provider single log-out service URL>`

This element provides the host name, port number, and path components of the single logout service URL at the identity provider.

`<query>= ...<URL-encoded LogoutRequest>...`

The `<query>` MUST contain a single logout request.

3.5.2.1.3. Step 3: Accessing the Identity Provider Single Logout Service

In step 3, the user agent accesses the identity provider's single logout service URL with the `<lib:LogoutRequest>` information attached to the URL fulfilling the redirect request.

3.5.2.1.4. Step 4: Processing the Request

In step 4, the identity provider MUST process the `<lib:LogoutRequest>` according to the rules defined in [LibertyProtSchema].

Each service provider for which the identity provider has provided authentication assertions during the Principal's current session MUST be notified via the service provider's preferred profile for logout request from the identity provider (see Section 3.5.1).

The identity provider's current session with the Principal MUST be terminated, and no more authentication assertions for the Principal are to be given to service providers.

3.5.2.1.5. Step 5: Redirecting to the Service Provider Return URL

In step 5, the identity provider's single logout service responds and redirects the user agent back to service provider using the return URL location obtained from the SingleLogoutServiceReturnURL metadata element. If the URL-encoded `<lib:LogoutRequest>` message received in step 3 contains a parameter named RelayState, then the identity provider MUST include a `<query>` component containing the same RelayState parameter and its value in its response to the service provider.

The purpose of this redirect is to return the user agent to the service provider.

The HTTP response MUST take the following form:

```
<HTTP-Version> 302 <Reason Phrase>
<other headers>
Location : https://<Service Provider Return Service URL>?<query>
<other HTTP 1.0 or 1.1 components>
```

where:

`<Service Provider Service Return URL>`

This element provides the host name, port number, and path components of the return URL location at the service provider.

`<query>= ...<URL-encoded LogoutResponse>`

The `<query>` component MUST contain a single logout response. The `<URL-encoded LogoutResponse>` component MUST contain the identical RelayState parameter and its value that was received in the URL-encoded logout request message obtained in step 3. If no RelayState parameter was provided in the step 3 message, then a RelayState parameter MUST NOT be specified in the `<URL-encoded LogoutResponse>`.

3.5.2.1.6. Step 6: Accessing the Service Provider Return URL

In step 6, the user agent accesses the service provider's return URL location fulfilling the redirect request.

3.5.2.1.7. Step 7: Confirmation

In step 7, the user agent is sent an HTTP response that confirms the requested action of a single logout has been completed.

3.5.2.2. SOAP/HTTP-Based Profile

The SOAP/HTTP-based profile relies on using SOAP over HTTP messages to communicate a logout request to the identity provider. The identity provider will then communicate a logout request to each service provider it has established a session with for the Principal via the service provider' preferred profile for logout requests from the identity provider (see Section 3.5.1). See Figure 16.

The following URI-based identifier MUST be used when referencing this specific profile:

URI: `http://projectliberty.org/profiles/slo-sp-soap`

This URI identifier is intended for service provider consumption and is not needed in provider metadata.

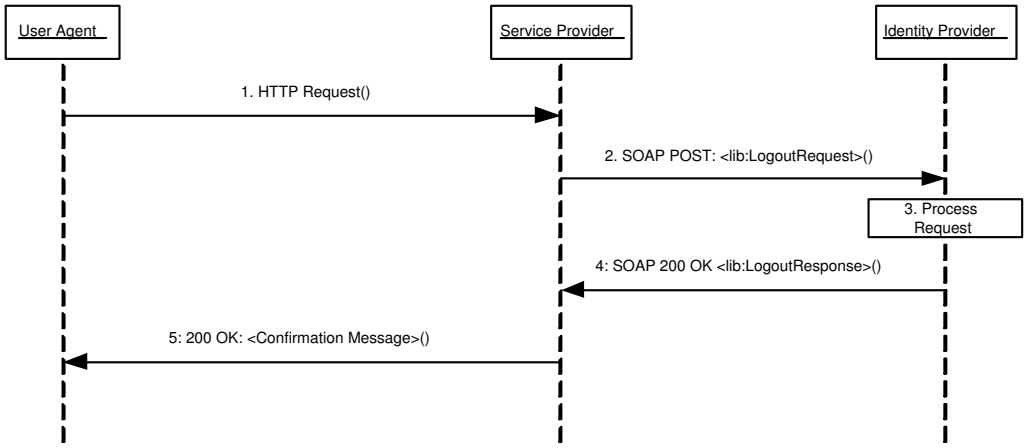


Figure 16. SOAP/HTTP-based profile for single logout initiated at service provider

Note:

Step 3 may involve an iterative process by the identity provider to implement the preferred profile for logout requests for each service provider that has been issued authentication assertions during the Principal's current session.

3.5.2.2.1. Step 1: Accessing Single Logout Service

In step 1, the user agent accesses the single logout service URL at the service provider via an HTTP request.

3.5.2.2.2. Step 2: Logout Request

In step 2, the service provider sends a SOAP over HTTP request to the identity provider's SOAP endpoint. The SOAP message MUST contain exactly one `<lib:LogoutRequest>` element in the SOAP body and adhere to the construction rules as defined in [LibertyProtSchema].

If a SOAP fault occurs, the service provider SHOULD employ best efforts to resolve the fault condition and resend the single logout request to the identity provider.

3.5.2.2.3. Step 3: Processing the Logout Request

In step 3, the identity provider MUST process the `<lib:LogoutRequest>` according to the rules defined in [LibertyProtSchema].

Each service provider for which the identity provider has provided authentication assertions during the Principal's current session MUST be requested to logout the Principal via the service provider's preferred profile for logout requests from the identity provider. If the identity provider determines that one or more of service providers to which it has provided assertions regarding this Principal do not support the SOAP profiles for the single logout, the identity provider MUST return a `<lib:LogoutResponse>` containing a status code of `<lib:UnsupportedProfile>`. The service provider MUST then re-submit its LogoutRequest via the HTTP profile described above.

Note:

If the identity provider is proxying authentication, based on authentication assertions from a second (proxied) identity provider (see Dynamic Proxying of Identity Providers in [LibertyProtSchema]), then the identity provider MUST follow these processing steps, *prior* to attempting to propagate the Single Logout request to service providers for which they have provided authentication assertions (as described above):

1. If the identity provider determines that the proxied identity provider does not support the SOAP/HTTP profile of Single Logout, it MUST respond to the requesting service provider with a `<lib:LogoutResponse>` containing a status code of `<lib:UnsupportedProfile>`.

2. If the identity provider is able to proceed with SOAP/HTTP-based Single Logout, then it MUST initiate the SP-initiated SOAP/HTTP profile of Single Logout described in Section 3.5.2.2, *acting in the role of service provider* toward the proxied identity provider. The proxied identity provider (in processing the SP-initiated SLO request) may determine that some service provider for which it provided authentication assertions does not support the SOAP/HTTP profile of Single Logout, and might thus return a `<lib:UnsupportedProfile>` status response. If this situation occurs, the proxying identity provider MUST return a `<lib:UnsupportedProfile>` status response to the requesting service provider.

The identity provider's current session with the Principal MUST be terminated, and no more authentication assertions for the Principal are to be given to service providers.

3.5.2.2.4. Step 4: Responding to the Logout Request

In step 4, the identity provider MUST respond to the `<lib:LogoutRequest>` with a SOAP 200 OK `<lib:LogoutResponse>` message.

3.5.2.2.5. Step 5: Confirmation

In step 5, the user agent is sent an HTTP response that confirms the requested action of single logout was completed.

3.6. Identity Provider Introduction

This section defines the profiles by which a service provider discovers which identity providers a Principal is using. In identity federation networks having more than one identity provider, service providers need a means to discover which identity providers a Principal uses. The introduction profile relies on a cookie that is written in a domain that is common between identity providers and service providers in an identity federation network. The domain that the identity federation network predetermines for a deployment is known as the common domain in this specification, and the cookie containing the list of identity providers is known as the common domain cookie.

Implementation of this profile is OPTIONAL. Whether identity providers and service providers implement this profile is a policy and deployment issue outside the scope of this specification. Also, which entities host web servers in the common domain is a deployment issue and is outside the scope of this specification.

3.6.1. Common Domain Cookie

The name of the cookie MUST be _liberty_idp. The format of the cookie content MUST be a list of base64-encoded (see [RFC2045]) identity provider succinct IDs separated by a single white space character. The identity provider IDs MUST adhere to the creation rules as defined in Section 3.2.2.2. The identity provider ID is a metadata element, as defined in [LibertyMetadata].

The common domain cookie writing service SHOULD append the identity provider ID to the list. If the identity provider ID is already present in the list, it MAY remove and append it when authentication of the Principal occurs. The intent is that the most recently established identity provider session is the last one in the list.

The cookie MUST be set with no Path prefix or a Path prefix of "/". The Domain MUST be set to ".[common-domain]" where [common-domain] is the common domain established within the identity federation network for use with the introduction protocol. The cookie MUST be marked as Secure.

The cookie SHOULD be URL-encoded.

Cookie syntax should be in accordance with [RFC2965] or [NetscapeCookie].

The cookie MAY be either session or persistent. This choice may be made within an identity federation network, but should apply uniformly to all providers in the network (see [LibertyImplGuide]) for more details on cookies).

3.6.2. Setting the Common Domain Cookie

After the identity provider authenticates a Principal, it MAY set the common domain cookie. The means by which the identity provider sets the cookie are implementation-specific so long as the cookie is successfully set with the parameters given above. One possible implementation strategy follows and should be considered non-normative. The identity provider may:

- Have previously established a DNS and IP alias for itself in the common domain
- Redirect the user agent to itself using the DNS alias using a URL specifying `"https"` as the URL scheme. The structure of the URL is private to the implementation and may include session information needed to identify the user-agent.
- Set the cookie on the redirected user agent using the parameters specified above.
- Redirect the user agent back to itself, or, if appropriate, to the service provider.

3.6.3. Obtaining the Common Domain Cookie

When a service provider needs to discover which identity providers the Principal uses, it invokes a protocol exchange designed to present the common domain cookie to the service provider after it is read by an HTTP server in the common domain.

If the HTTP server in the common domain is operated by the service provider, the service provider MAY redirect the user agent to an identity provider's intersite transfer service for an optimized single sign-on process.

The specific means by which the service provider reads the cookie are implementation-specific so long as it is able to cause the user agent to present cookies that have been set with the parameters given in Section 3.6.1. One possible implementation strategy is described as follows and should be considered non-normative. Additionally, it may be sub-optimal for some applications.

- Have previously established a DNS and IP alias for itself in the common domain
- Redirect the user agent to itself using the DNS alias using a URL specifying `"https"` as the URL scheme. The structure of the URL is private to the implementation and may include session information needed to identify the user-agent.
- Set the cookie on the redirected user agent using the parameters specified above
- Redirect the user agent back to itself, or, if appropriate, to the service provider.

3.7. NameIdentifier Mapping Profile

The NameIdentifier mapping profile specifies how a service provider may obtain a NameIdentifier for a Principal it has federated in the "namespace" of a different service provider, by querying an identity provider that has federated the Principal with both service providers. This NameIdentifier may be used to obtain additional information about a Principal from a SAML authority, or used for other non-specific purposes. In most cases, the encryption profile in the following section will be used to obfuscate and time-limit this identifier to restrict its use.

The NameIdentifier mapping profile makes use of the following metadata elements, as defined in [LibertyMetadata]:

- `NameIdentifierMappingProtocolProfile` - A URI indicating the profile of this protocol supported by the identity provider.
- `SOAPEndpoint` - The SOAP endpoint location at the identity provider to which Liberty SOAP messages are sent.

3.7.1. SOAP-based NameIdentifier Mapping

The SOAP-based profile relies on a SOAP request and response to query for and return the NameIdentifier. A requesting service provider issues a SOAP request to an identity provider, requesting a different NameIdentifier for a named Principal in the namespace of a service provider or affiliation group. This NameIdentifier may then be used to query another Liberty provider offering SAML services for additional information about the named Principal. See Figure 17.

The following URI-based identifier MUST be used when referencing this specific profile:

URI: `http://projectliberty.org/profiles/nim-sp-http`

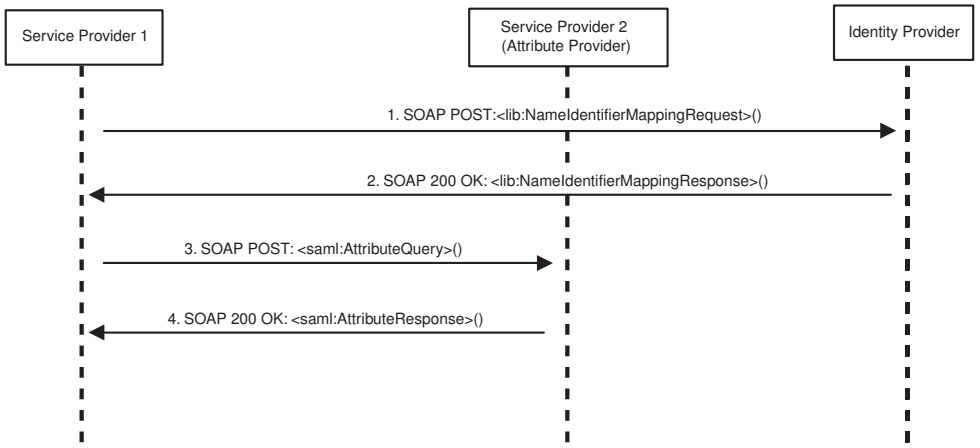


Figure 17. SOAP-based profile for name identifier mapping

3.7.1.1. Step 1: Issuance and processing of Request

In step 1, the service provider sends a SOAP over HTTP request to the SOAP endpoint of the identity provider it is querying. The SOAP message MUST contain exactly one `<lib:NameIdentifierMappingRequest>` element in the SOAP body and adhere to the construction rules defined in [LibertyProtSchema].

The identity provider MUST process the `<lib:NameIdentifierMappingRequest>` according to the rules defined in [LibertyProtSchema]. The identity provider is NOT required to honor the request for a mapped NameIdentifier, but it MUST respond to the request with an appropriate status.

3.7.1.2. Step 2: Responding to the Request

In step 3, the identity provider MUST respond to the `<lib:NameIdentifierMappingRequest>` with a SOAP 200 OK `<lib:NameIdentifierMappingResponse>` message.

3.7.1.3. Step 3: Requesting SAML attributes using a mapped NameIdentifier

Note: This step is not normatively specified by Liberty, and is shown only for illustrative purposes. The requesting service provider may use the mapped NameIdentifier of the Principal to issue a `<saml:AttributeQuery>`. This MUST adhere to the rules specified in [SAMLCore11]

3.7.1.4. Step 4: Returning a <saml:AttributeStatement>

Note: This step is not normatively specified by Liberty, and is shown only for illustrative purposes. A service provider receiving a `<saml:AttributeQuery>` may return a `<saml:AttributeStatement>`. This action MUST conform to the rules specified in [SAMLCore11].

3.7.1.5. Security Considerations

In addition to the usual considerations relating to Liberty and SAML protocols (see [SAMLCore11]), an identity provider SHOULD encrypt or otherwise obfuscate the NameIdentifier returned to the requesting service provider, so that it is opaque to the requester. A way of accomplishing this is described in the next section.

Because the identifier gives the receiving provider a persistent way of referencing the principal, it should only be returned subject to the policies set by the principal or other authorized party.

3.8. NameIdentifier Encryption Profile

The Liberty NameIdentifier encryption profile allows a principal's NameIdentifier to be encrypted such that only the identity or service provider possessing the decryption key can deduce the identity of the principal when the NameIdentifier is included in a SAML or Liberty protocol message. The identifier is encrypted in such a fashion that it is a different value when requested by different providers or multiple times, reducing the chance for correlation of the encrypted value across multiple logical transactions.

The NameIdentifier encryption profile make use of the following metadata element, as defined in [LibertyMetadata]:

- `KeyDescriptor` - Defines a public key to use when wrapping the keys used in encrypting data for a provider (the key-encrypting key)

3.8.1. XML Encryption-based NameIdentifier Encryption

The XML Encryption-based profile relies on the use of [xmlenc-core] to format and encode the resulting encrypted identifier and possibly the wrapped encryption key.

The following URI-based identifier MUST be used when referencing this specific profile:

URI: `urn:liberty:iff:nameid:encrypted`

3.8.1.1. Step 1: Encrypting and encoding a NameIdentifier value.

The encrypting provider first transforms the original `<saml:NameIdentifier>` element into a `<EncryptableNameIdentifier>` element, which is an extension of the original element. The `NameQualifier`, `IssueInstant`, and `Nonce` attributes are set as defined by [LibertyProtSchema].

If not already generated for the target provider, an encryption key is generated and is then itself encrypted with the key specified in the target provider's `<KeyDescriptor>` metadata element with a `use` attribute of `encryption`, or with a predefined key exchanged out of band. The wrapped encryption key is placed into a `<xenc:EncryptedKey>` element.

If the symmetric encryption key is not included, because it has been exchanged out of band, and/or is being reused, then the encrypting provider MUST include additional information in the `<xenc:EncryptedData>` element that indicates to the target provider which decryption key to use in decrypting the identifier. This information MUST be sufficient to identify the key to use without the target knowing the encrypting provider's identity before decryption occurs.

The encryption key is then applied to the `<EncryptableNameIdentifier>` element, producing an `<xenc:EncryptedData>` element with a `Type` of `http://www.w3.org/2001/04/xmlenc#Element`.

The resulting `<xenc:EncryptedData>` element, and optionally the `<xenc:EncryptedKey>` element, are then enclosed in an `<EncryptedNameIdentifier>` element. The element is base-64 encoded and the result is placed into a `<saml:NameIdentifier>` whose `Format` attribute MUST be `urn:liberty:iff:nameid:encrypted`.

3.8.1.2. Step 2: Decoding and decrypting a NameIdentifier value.

The decrypting provider first decodes the base-64 encoded data and recovers the `<EncryptedNameIdentifier>` element.

The `<xenc:EncryptedData>` and optional `<xenc:EncryptedKey>` elements are then used to recover the symmetric encryption key and algorithm and decrypt the `<EncryptableNameIdentifier>` element.

The provider can then examine the attributes to determine the identity federation to which the name identifier applies.

3.8.2. Security Considerations

The profile is designed to meet the needs of providers in addressing the security considerations of other profiles, such as the NameIdentifier Mapping Profile in the previous section. To insure the integrity of this profile, either symmetric encryption keys MUST NOT be reused, or if they are, then symmetric encryption keys MUST be reused between different principals federated with a given provider and MUST NOT be reused between different providers. It is RECOMMENDED that symmetric encryption keys, if reused, be renewed periodically. Furthermore, reuse of keys REQUIRES that a chaining mode with a unique initialization vector generated per encryption be used.

4. Security Considerations

4.1. Introduction

This section describes security considerations associated with Liberty protocols for identity federation, single sign-on, federation termination, and single logout.

Liberty protocols, schemas, bindings, and profiles inherit and use extensively the SAML protocols. Therefore, the security considerations published along with the SAML specification have direct relevance (see [SAMLCore11], [SAMLBind11], and [SAMLSec]). Throughout this section if, for any reason, a specific consideration or countermeasure does not apply or differs, notice of this fact is made; and a description of alternatives is supplied, where possible.

4.2. General Requirements

4.2.1. Security of SSL and TLS

SSL and TLS utilize a suite of possible cipher suites. The security of the SSL or TLS session depends on the chosen cipher suite. An entity (that is, a user agent, service provider, or identity provider) that terminates an SSL or TLS connection needs to offer (or accept) suitable cipher suites during the handshake. The following list of TLS 1.0 cipher suites (or their SSL 3.0 equivalent) is recommended.

- `TLS_RSA_WITH_RC4_128_SHA`
- `TLS_RSA_WITH_3DES_EDE_CBC_SHA`
- `TLS_DHE_DSS_WITH_3DES_EDE_CBC_SHA`

The above list is not exhaustive. The recommended cipher suites are among the most commonly used. Note: New cipher suites are added as they are standardized and should be considered for inclusion if they have sufficiently strong security properties. For example, it is anticipated that the AES-based cipher suites being standardized in the IETF will be widely adopted and deployed.

4.2.2. Security Implementation

The suitable implementation of security protocols is necessary to maintain the security of a system, including

- Secure random or pseudo-random number generation
- Secure storage

4.3. Classification of Threats

4.3.1. Threat Model

For an analysis of threat classifications, an Internet threat model has been used. In other words, the threat model assumes that intermediary and end-systems participating in Liberty protocol exchanges have not been compromised. However, where possible, the consequences and containment properties of a compromised system entity are described and countermeasures are suggested to bolster the security posture so that the exposure from a security breach is minimized.

Given the nature of the Internet, the assumption is made that deployment is across the global Internet and, therefore, crosses multiple administrative boundaries. Thus, an assumption is also made that the adversary has the capacity to engage in both passive and active attacks (see Section 4.3.3).

4.3.2. Rogue and Spurious Entities

Attackers may be classified based on their capabilities and the roles that they play in launching attacks on a Liberty system as follows:

- **Rogue Entities:** Entities that misuse their privileges. The rogue actors may be Principals, user agents, service providers, or identity providers. A rogue Principal is a legitimate participant who attempts to escalate its privileges or masquerade as another system Principal. A rogue user agent may, for instance, misuse the relationships between its associated Principals and an identity provider to launch certain attacks. Similarly, a rogue service provider may be able to exploit the relationship that it has either with a Principal or with an identity provider to launch certain attacks.

- **Spurious Entities:** Entities that masquerade as a legitimate entity or are completely unknown to the system. The spurious actors include Principals, user agents (i.e., user agents without associated legitimate Liberty Principals), service providers, or identity providers. A spurious service provider may, for instance, pretend to be a service provider that has a legitimate relationship with an identity provider. Similarly, a spurious Principal may be one that pretends to be a legitimate Principal that has a relationship with either a service provider or an identity provider.

4.3.3. Active and Passive Attackers

Both rogue and spurious entities may launch active or passive attacks on the system. In a passive attack the attacker does not inject traffic or modify traffic in any way. Such an attacker usually passively monitors the traffic flow, and the information that is obtained in that flow may be used at a later time. An active attacker, on the other hand, is capable of modifying existing traffic as well as injecting new traffic into the system.

4.3.4. Scenarios

The following scenarios describe possible attacks:

- **Collusion: The secret cooperation between two or more Liberty entities to launch an attack, for example,**

  - Collusion between Principal and service provider
  - Collusion between Principal and identity provider
  - Collusion between identity provider and service provider

- Collusion among two or more Principals
- Collusion between two or more service providers
- Collusion between two or more identity providers

- **Denial-of-Service Attacks:** The prevention of authorized access to a system resource or the delaying of system operations and functions.
- **Man-in-the-Middle Attacks:** A form of active wiretapping attack in which the attacker intercepts and selectively modifies communicated data to masquerade as one or more of the entities involved in a communication association.
- **Replay Attacks:** An attack in which a valid data transmission is maliciously or fraudulently repeated, either by the originator or by an adversary who intercepts the data and retransmits it, possibly as part of a masquerade attack.
- **Session Hijacking:** A form of active wiretapping in which the attacker seizes control of a previously established communication association.

4.4. Threat Scenarios and Countermeasures

In this section, threats that may apply to all the Liberty profiles are considered first. Threats that are specific to individual profiles are then considered. In each discussion the threat is described as well as the countermeasures that exist in the profile or the additional countermeasures that may be implemented to mitigate the threat.

4.4.1. Common Threats for All Profiles

Threat: Request messages sent in cleartext

Description: Most profile protocol exchanges do not mandate that all exchanges commence over a secure communication channel. This lack of transport security potentially exposes requests and responses to both passive and active attacks.

One obvious manifestation is when the initial contact is not over a secure transport and the Liberty profile begins to exchange messages by carrying the request message back to the user agent in the location header of a redirect.

Another such manifestation could be a request or response message which carries a URI that may be resolved on a subsequent exchange, for instance lib:AuthnContextClassRef. If this URI were to specify a less or insecure transport, then the exchange may be vulnerable to the types of attacks described above.

Countermeasure: Ensure that points of entry to Liberty protocol exchanges utilize the https URL `<scheme>` and that all interactions for that profile consistently exchange messages over `https`.

Threat: Malicious redirects into identity or service provider targets

Description: A spurious entity could issue a redirect to a user agent so that the user agent would access a resource that disrupts single sign-on. For example, an attacker could redirect the user agent to a logout resource of a service provider causing the Principal to be logged out of all existing authentication sessions.

Countermeasure: Access to resources that produce side effects could be specified with a transient qualifier that must correspond to the current authentication session. Alternatively, a confirmation dialog could be interposed that relies on a transient qualifier with similar semantics.

Threat: Relay state tampering or fabrication

Description: Some of the messages may carry a <lib:RelayState> element, which is recommended to be integrity-protected by the producer and optionally confidentiality-protected. If these practices are not followed, an adversary could trigger unwanted side effects. In addition, by not confidentiality-protecting the value of this element, a legitimate system entity could inadvertently expose information to the identity provider or a passive attacker.

Countermeasure: Follow the recommended practice of confidentiality- and integrity-protecting the <lib:RelayState> data. Note: Because the value of this element is both produced and consumed by the same system entity, symmetric cryptographic primitives could be utilized.

4.4.2. Single Sign-On and Federation

4.4.2.1. Common Interactions for All Single Sign-On and Federation Profiles

Threat: <lib:AuthnRequest> sent over insecure channel

Description: It is recommended that the initial exchange to access the intersite transfer service be conducted over a TLS-secured transport. Not following this recommendation can expose the exchange to both passive and active attacks.

Countermeasure: Deploy the intersite transfer service under an https scheme.

Threat: Unsigned <lib:AuthnRequest> message

Description: The signature element of an <lib:AuthnRequest> is optional and thus the absence of the signature could pose a threat to the identity provider or even the targeted service provider. For example, a spurious system entity could generate an unsigned <lib:AuthnRequest> and redirect the user agent to the identity provider. The identity provider must then consume resources.

Countermeasure: Sign the <lib:AuthnRequest>. The IDP can also verify the identity of the Principal in the absence of a signed request.

Threat: Replay of an authentication assertion

Description: After obtaining a valid assertion from an identity provider, either legitimately or surreptitiously, the entity replays the assertion to the Service at a later time. A digital signature must cover the entire assertion, thus elements within the assertion cannot be corrupted without detection during the mandatory verification step. However, it is possible to fabricate an <lib:AuthnResponse> with the valid assertion.

Countermeasure: The issuer should sign <lib:AuthnResponse> messages. Signing binds the <samlp:IssueInstant> of the response message to the assertion it contains. This binding accords the relying party the opportunity to temporally judge the response. Additionally, a valid signature over the response binds the <samlp:InResponseTo> element to the corresponding <lib:AuthnRequest>. (Specifying a short period that the authentication assertion can be relied upon will minimize, but not mitigate this threat. Binding the <lib:AssertionId> to the request <samlp:InResponseTo> element may also be handy.)

Threat: Fabricated <lib:AuthnResponse> denial of service

Description: An attacker captures the <samlp:RequestID> sent in an <lib:AuthnRequest> message by a service provider to an identity provider, and sends several spurious <lib:AuthnResponse> messages to the service provider with the same <samlp:InResponseTo>. Because the <samlp:InResponseTo> matches a <samlp:RequestID> that the service provider had used, the service provider goes through the process of validating the signature in the message. Thus, it is subject to a denial of service attack.

Countermeasure: A secure communication channel should be established before transferring requests and responses.

Threat: Collusion between two Principals

Description: After getting an artifact or <lib:AuthnResponse> in step 6 (see Section 3.2.1), a legitimate Principal A could pass this artifact or <lib:AuthnResponse> on to another Principal, B. Principal B is now able to use the artifact or <lib:AuthnResponse>, while the actual authentication happened via Principal A.

Countermeasure: Implementations where this threat is a concern MUST use the <saml:AuthenticationLocality> in the authentication statement. The IP address that Principal B uses would be different from the IP address within the <saml:AuthenticationLocality>. This countermeasure may not suffice when the user agent is behind a firewall or proxy server. IP spoofing may also circumvent this countermeasure.

Threat: Stolen artifact and subsequent Principal impersonation

Description: See Section 4.1.1.9.1 in [SAMLBind11]

Countermeasure: Identity providers MUST enforce a policy of one-time retrieval of the assertion corresponding to an artifact so that a stolen artifact can be used only once. Implementations where this threat is a concern MUST use the <saml:AuthenticationLocality> in the authentication statement. The IP address of a spurious user agent that attempts to use the stolen artifact would be different from IP address within the <saml:AuthenticationLocality>. The service provider may then be able to detect that the IP addresses differ. This countermeasure may not suffice when the user agent is behind a firewall or proxy server. IP address spoofing may also circumvent this countermeasure.

Threat: Stolen assertion and subsequent Principal impersonation

Description: See Section 4.1.1.9.1 in [SAMLBind11]

Countermeasure: Refer to the previous threat for requirements.

Threat: Rogue service provider uses artifact or assertion to impersonate Principal at a different service provider

Description: Because the <lib:AuthnResponse> contains the <lib:ProviderID>, this threat is not possible.

Countermeasure: None

Threat: Rogue identity provider impersonates Principal at a service provider

Description: Because the Principal trusts the identity provider, it is assumed that the identity provider does not misuse the Principal's trust.

Countermeasure: None

Threat: Identity provider modifies Principal during a session with a service provider

Description: A service provider whose session has exceeded the <ReauthenticateOnOrAfter> time must contact the Identity provider to get a new assertion. The new assertion might be for a different identity.

Countermeasure: Service providers should continue to follow assertion processing rules to ensure that the subject of any assertions received is actually the user for which the assertion is needed.

Threat: Rogue user attempts to impersonate currently logged-in legitimate Principal and thereby gain access to protected resources.

Description: Once a Principal is successfully logged into an identity provider, subsequent <AuthnRequest> messages from different service providers concerning that Principal will not necessarily cause the Principal to be reauthenticated. Principals must, however, be authenticated unless the identity provider can determine that an <AuthnRequest> is associated not only with the Principal's identity, but also with a validly authenticated identity provider session for that Principal.

Countermeasure: In implementations where this threat is a concern, identity providers MUST maintain state information concerning active sessions, and MUST validate the correspondence between an <AuthnRequest> and

an active session before issuing an <AuthnResponse> without first authenticating the Principal. Cookies posted by identity providers MAY be used to support this validation process, though Liberty does not mandate a cookie-based approach.

4.4.2.2. Liberty-Enabled Client and Proxy Profile

Threat: Intercepted <lib:AuthnRequestEnvelope> and <lib:AuthnResponse> and subsequent Principal impersonation.

Description: A spurious system entity can interject itself as a man-in-the-middle (MITM) between the user agent (LECP) and a legitimate service provider, where it acts in the service provider role in interactions with the LECP, and in the user agent role in interactions with the legitimate service provider. In this way, as a first step, the MITM is able to intercept the service provider's <lib:AuthnRequestEnvelope> (step 3 of Section 3.2.4) and substitute any URL of its choosing for the <lib:AssertionConsumerServiceURL> value before forwarding the <lib:AuthnRequestEnvelope> on to the LECP. Typically, the MITM will insert a URL value that points back to itself. Then, if the LECP subsequently receives a <lib:AuthnResponseEnvelope> from the identity provider (step 6 in Section 3.2.4) and subsequently sends the contained <lib:AuthnResponse> to the <lib:AssertionConsumerServiceURL> received from the MITM, the MITM will be able to masquerade as the Principal at the legitimate service provider.

Countermeasure: The identity provider specifies to the LECP the address to which the LECP must send the <lib:AuthnResponse>. The <lib:AssertionConsumerServiceURL> in the <lib:AuthnResponseEnvelope> element is for this purpose. This URL value is among the metadata that identity and service providers must exchange in the process of establishing their operational relationship (see Section 3.1 and Section 3.1.3).

4.4.2.3. Federation

Threat: Collusion among service providers can violate privacy of the Principal

Description: When a group of service providers collude to share the <lib:IDPProvidedNameIdentifier> of a Principal, they can track and in general compromise the privacy of the principal. More generally, this threat exists for any common data (e.g. phone number) shared by rogue system entities.

Countermeasure: The <lib:IDPProvidedNameIdentifier> is required to be unique for each identity provider to service provider relationship. However, this requirement does not eliminate the threat when there are rogue participants under the Principal's identity federation. The only protection is for Principals to be cautious when they choose service providers and understand their privacy policies.

Threat: Poorly generated name identifiers may compromise privacy

Description: The federation protocol mandates that the <lib:NameIdentifier> elements be unique within a Principal's federated identities. The name identifiers exchanged are pseudonyms and, to maintain the privacy of the Principal, should be resistant to guessing or derivation attacks.

Countermeasure: Name identifiers should be constructed using pseudo-random values that have no discernable correspondence with the Principal's identifier (or name) used by the entity that generates the name identifier.

4.4.3. Name Registration

No known threats.

4.4.4. Federation Termination: HTTP-Redirect-Based Profile

Threat: Attacker can monitor and disrupt termination

Description: During the initial steps, a passive attacker can collect the `<lib:FederationTerminationNotification>` information when it is issued in the redirect. This threat is possible because the first and second steps are not required to use https as the URL scheme. An active attacker may be able to intercept and modify the message conveyed in step 2 because the digital signature only covers a portion of the message. This initial exchange also exposes the name identifier. Exposing these data poses a privacy threat.

Countermeasure: All exchanges should be conducted over a secure transport such as SSL or TLS.

4.4.5. Single Logout: HTTP-Redirect-Based Profile

Threat: Passive attacker can collect a Principal's name identifier

Description: During the initial steps, a passive attacker can collect the `<lib:LogoutRequest>` information when it is issued in the redirect. Exposing these data poses a privacy threat.

Countermeasure: All exchanges should be conducted over a secure transport such as SSL or TLS.

Threat: Unsigned `<lib:LogoutRequest>` message

Description: An Unsigned `<lib:LogoutRequest>` could be injected by a spurious system entity thus denying service to the Principal. Assuming that the NameIdentifier can be deduced or derived then it is conceivable that the user agent could be directed to deliver a fabricated `<lib:LogoutRequest>` message.

Countermeasure: Sign the `<lib:LogoutRequest>` message. The identity provider can also verify the identity of a Principal in the absence of a signed request.

4.4.6. Identity Provider Introduction

No known threats.

References

Normative

[HTML4] Raggett, D., Le Hors, A., Jacobs, I, eds. (24 December 1999). "HTML 4.01 Specification ," Recommendation, W3C *http://www.w3.org/TR/html401/ [August 2003].*

[LibertyAuthnContext] Madsen, Paul, eds. "Liberty ID-FF Authentication Context Specification," Version 1.2-errata-v1.0, Liberty Alliance Project (12 September 2004). *http://www.projectliberty.org/specs*

[LibertyGlossary] "Liberty Technical Glossary," Version 1.3-errata-v1.0, Liberty Alliance Project (12 Aug 2004). *http://www.projectliberty.org/specs* Hodges, Jeff, eds.

[LibertyMetadata] Davis, Peter, eds. "Liberty Metadata Description and Discovery Specification," Version 1.0-errata-v2.0, Liberty Alliance Project (12 September 2004). *http://www.projectliberty.org/specs*

[LibertyProtSchema] Cantor, Scott, Kemp, John, eds. "Liberty ID-FF Protocols and Schema Specification," Version 1.2-errata-v2.0, Liberty Alliance Project (12 September 2004). *http://www.projectliberty.org/specs*

[SAMLBind11] Maler, Eve, Mishra, Prateek, Philpott, Rob, eds. (27 May 2003). "Bindings and Profiles for the OASIS Security Assertion Markup Language (SAML) V1.1," OASIS Committee Specification, version 1.1, Organization for the Advancement of Structured Information Standards *http://www.oasis-open.org/committees/documents.php?wg_abbrev=security*

[SAMLCore11] Maler, Eve, Mishra, Prateek, Philpott, Rob, eds. (27 May 2003). "Assertions and Protocol for the OASIS Security Assertion Markup Language (SAML) V1.1," OASIS Committee Specification, version 1.1, Organization for the Advancement of Structured Information Standards *http://www.oasis-open.org/committees/documents.php?wg_abbrev=security*

[SAMLSec] McClaren, C., eds. (05 November 2002). "Security Considerations for the OASIS Security Assertion Markup Language (SAML)," Version 1.0, OASIS Standard, Organization for the Advancement of Structured Information Standards *http://www.oasis-open.org/committees/security/#documents*

[Schema1] Thompson, Henry S., Beech, David, Maloney, Murray, Mendelsohn, Noah, eds. (May 2002). "XML Schema Part 1: Structures," Recommendation, World Wide Web Consortium *http://www.w3.org/TR/xmlschema-1/*

[SOAPv1.1] "Simple Object Access Protocol (SOAP) 1.1," Box, Don, Ehnebuske, David , Kakivaya, Gopal, Layman, Andrew, Mendelsohn, Noah, Nielsen, Henrik Frystyk, Winer, Dave, eds. World Wide Web Consortium W3C Note (08 May 2000). *http://www.w3.org/TR/2000/NOTE-SOAP-20000508/*

[SSL] Frier, A., Karlton, P., Kocher, P., eds. (November 1996). Netscape Communications Corporation "The SSL 3.0 Protocol," *http://www.netscape.com/eng/ssl3/*

[RFC1750] Eastlake , D., Crocker, S., Schiller, J., eds. (December 1994). "Randomness Recommendations for Security ," RFC 1750, Internet Engineering Task Force *http://www.ietf.org/rfc/rfc1750.txt [August 2003].*

[RFC1945] Berners-Lee, T., Fielding, R., Frystyk, H., eds. (May 1996). "Hypertext Transfer Protocol – HTTP/1.0 ," RFC 1945, Internet Engineering Task Force *http://www.ietf.org/rfc/rfc1945.txt [August 2003].*

[RFC2045] Freed, N., Borenstein, N., eds. (November 1996). "Multipurpose Internet Mail Extensions (MIME) Part One: Format of Internet Message Bodies ," RFC 2045., Internet Engineering Task Force *http://www.ietf.org/rfc/rfc2045.txt [November 1996].*

[RFC2965] Kristol, D., Montulli, L., eds. (October 2000). "HTTP State Management Mechanism," RFC 2965., Internet Engineering Task Force *http://www.ietf.org/rfc/rfc2965.txt [October 2000].*

[RFC2119] Bradner, S., eds. "Key words for use in RFCs to Indicate Requirement Levels," RFC 2119, The Internet Engineering Task Force (March 1997). *http://www.ietf.org/rfc/rfc2119.txt [March 1997]*.

[RFC2246] Dierks, T., Allen, C., eds. (January 1999). "The TLS Protocol," Version 1.0 RFC 2246, Internet Engineering Task Force *http://www.ietf.org/rfc/rfc2246.txt [January 1999]*.

[RFC2396] Berners-Lee, T., Fielding, R., Masinter, L., eds. (August 1998). "Uniform Resource Identifiers (URI): Generic Syntax," RFC 2396, The Internet Engineering Task Force *http://www.ietf.org/rfc/rfc2396.txt [August 1998]*.

[RFC2616] Fielding, R., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P., Berners-Lee, T., eds. (June 1999). "Hypertext Transfer Protocol – HTTP/1.1," RFC 2616, The Internet Engineering Task Force *http://www.ietf.org/rfc/rfc2616.txt [June 1999]*.

[RFC2617] Franks, J., Hallam-Baker, P., Hostetler, J., Lawrence, S., Leach, P., Stewart, L., eds. (June 1999). "HTTP Authentication: Basic and Digest Access Authentication," RFC 2617, The Internet Engineering Task Force *http://www.ietf.org/rfc/rfc2617.txt [June 1999]*.

[RFC3106] Eastlake , D., Goldstein, T., eds. (April 2001). "ECML v1.1: Field Specifications for E-Commerce ," RFC 3106, Internet Engineering Task Force *http://www.ietf.org/rfc/rfc3106.txt [April 2001]*.

[WML] "Wireless Markup Language Version 1.3 Specification," Version 1.3, Open Mobile Alliance *http://www.openmobilealliance.org/tech/affiliates/wap/wapindex.html*

[xmlenc-core] Eastlake, Donald, Reagle, Joseph, eds. (December 2002). "XML Encryption Syntax and Processing," W3C Recommendation, World Wide Web Consortium *http://www.w3.org/TR/xmlenc-core/*

Informative

[LibertyImplGuide] "Liberty ID-FF Implementation Guidelines," Version 1.2, Liberty Alliance Project (18 April 2004). *http://www.projectliberty.org/specs* Thompson, Peter, Champagne, Darryl, eds.

[NetscapeCookie] eds. "Persistent Client State HTTP Cookies," Netscape *http://wp.netscape.com/newsref/std/cookie_spec.html*