Liberty Bindings and Profiles Specification

Version 1.0

11 July 2002

Document Description: liberty-architecture-bindings-and-profiles-v1.0

Notice

Liberty Alliance Project
Licensing Administrator
c/o IEEE-ISTO
445 Hoes Lane, P.O. Box 1331
Piscataway, NJ 08855-1331, USA

67 ## Editor

68 Jason Rouault, Hewlett-Packard Company

69 ## Contributors

70

71 The following Liberty Alliance Project Sponsor companies contributed to the development of
72 this specification:

73

ActivCard
American Express Travel Related Services
America Online, Inc.
Bank of America
Bell Canada
Catavault
Cingular Wireless
Cisco Systems, Inc.
Citigroup
Cyberun Corporation
Deloitte & Touche LLP
EarthLink, Inc.
Electronic Data Systems, Inc.
Entrust, Inc.
Ericsson
Fidelity Investments
France Telecom
Gemplus
General Motors
Hewlett-Packard Company
i2 Technologies, Inc.
Intuit Inc.
MasterCard International
Nextel Communications
Nippon Telegraph and Telephone Company
Nokia Corporation
Novell, Inc.
NTT DoCoMo, Inc.
OneName Corporation
Openwave Systems Inc.
PricewaterhouseCoopers LLP
Register.com
RSA Security Inc
Sabre Holdings Corporation
SAP AG
SchlumbergerSema
Sony Corporation
Sun Microsystems, Inc.
United Airlines
VeriSign, Inc.
Visa International
Vodafone Group Plc
Wave Systems

74

75

76

77

78

Table of 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 [SAMLCore] and makes use of adaptations of the SAML profiles in [SAMLBind]. 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 [LibertyGloss].

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:

- The prefix `lib:` stands for the Liberty namespace `http://projectliberty.org/schemas/core/2002/08/`
- 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#` (see [XMLSig]).
- 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.

166 ## 2.1 SOAP Binding for Liberty

167 Because the Liberty protocols are an extension of the SAML protocol (see [SAMLCore]) and a
168 SOAP protocol binding for SAML has been defined, the SOAP binding for Liberty MUST adhere to
169 the processing rules for the "SOAP binding for SAML" as specified in [SAMLBind] unless
170 otherwise noted. Just like SAML, the SOAP binding for Liberty uses HTTP as the transport
171 mechanism.

172 ## 2.2 Example of Message Exchange Using SOAP over HTTP

173 The following is an example of a Liberty-enabled client and proxy request that asks for an assertion
174 containing an authentication statement from an identity provider.

```
175 POST /LibertyService HTTP/1.1
176 Host: www.example.com
177 Content-Type: text/xml
178 Content-Length: nnn
179 SOAPAction: "http://projectliberty.org/architecture#AuthnRequest"
180 <SOAP-ENV:Envelope
181     xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
182     <SOAP-ENV:Body>
183         <lib:AuthnRequest xmlns:samlp:="…" xmlns:saml="…"
184               xmlns:ds="…" xmlns:lib="…">
185             <ds:Signature> … </ds:Signature>
186             …
187         </lib:AuthnRequest>
188     </SOAP-ENV:Body>
189 </SOAP-ENV:Envelope>
```

190

191 The following is an example of the corresponding response, which supplies an assertion containing
192 the authentication statement as requested.

```
193 HTTP/1.1 200 OK
194 Content-Type: text/xml
195 Content-Length: nnnn
196
197 <SOAP-ENV:Envelope
198     xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
199     <SOAP-ENV:Body>
200         <lib:AuthnResponseEnvelope xmlns:samlp="…" xmlns:saml="…"
201             xmlns:ds="…" xmlns:lib="…">
202             <lib:AuthnResponse . . .>
203         <Status>
204            <StatusCode value="samlp:Success"/>
205         </Status>
206               <ds:Signature> … </ds:Signature>
207             <lib:Assertion>
208               …
209             </lib:Assertion>
210          </lib:AuthnResponse>
211         </lib:AuthnResponseEnvelope>
212     </SOAP-Env:Body>
213 </SOAP-ENV:Envelope>
```

214

215 # 3 Profiles

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

220 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 Profile**: The profile by which service providers specify a Principal's opaque handle (or *name identifier*) that an identity provider will use when communicating to the service provider about the Principal.
- **Identity 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.

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 [SSLv3]) 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 Service providers MUST authenticate identity providers using identity provider server-side X.509 certificates.

4 Identity providers MAY authenticate service providers using service provider client-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 `<lib:AuthnRequest>` messages SHOULD be digitally signed and verified. All `<lib:Assertion>` elements MUST be digitally signed and verified. For signing and verification of `<lib:AuthnRequest>` and `<lib:Assertion>` elements, identity providers 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. Certificates and private keys MUST be suitable for long-term signatures.

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, and 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 2.1).

12 In all SOAP message exchanges, the SOAP receiver MUST authenticate the SOAP Sender.

3.1.1 User Agent

A user agent, unless otherwise noted in the specific profile, 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 [SSLv3]) or TLS 1.0 (see [RFC2246]) or any subsequent protocols that are backwards compatible with SSL 3.0 and/or TLS 1.0 either directly or via a proxy (for example, a WAP gateway).
- Minimum URL length of 256 bytes.

Additionally, to support the optional identity provider introduction profile, either the user agent or a proxy must support session cookies (see [RFC2109]). Support for persistent cookies will yield a more seamless user experience.

3.1.2 Formatting and Encoding of Protocol Messages

All Liberty 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 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 method of GET as defined in [HTML4].

The original Liberty XML protocol message MUST be encoded as follows:

- The `<query>` component parameter name MUST be the Liberty XML protocol message element or attribute name.
- The `<query>` component parameter value MUST be the value of the Liberty XML protocol message element or attribute value.
- When the original message element has multiple values, the value of the `<query>` component parameter MUST be a space-delimited list.
- Some of the referenced protocol message elements and attributes are optional. If an optional element or attribute does not appear in the original Liberty 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.

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 omitting 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`.

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

The following signature algorithms and their identifiers 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 detects 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:AuthnResquest>` 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]">
       <samlp:RespondWith> lib:AuthnResponse </samlp:RespondWith>
       <lib:ProviderID> [ProviderID] </ lib:ProviderID>
       <lib:ForceAuthn> [ForceAuthn] </lib:ForceAuthn>
       <lib:IsPassive> [IsPassive] </lib:IsPassive>
       <lib:Federate> [Federate] </lib:Federate>
       <lib:ProtocolProfile> [ProtocolProfile] </lib:ProtocolProfile>
       <lib:AuthnContext>
         <lib:authnContextMinimumClassRef>
             [AuthnContextMinimumClassRef]
         </lib:authnContextMinimumClassRef>
         <lib:authnContextStatementRef> [AuthnContextStatementRef] </lib:authnContextStatementRef>
         <lib:AuthnContextClassRef> [AuthnContextClassRef] </lib:AuthnContextClassRef>
      </lib:AuthnContext>
       <lib:RelayState> [RelayState] </lib:RelayState>
</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, ForceAuthn, IsPassive, Federate, ProtocolProfile, AuthnContextStatementRef, AuthnContextMinimumClassRef, AuthnContextClassRef, RelayState`

- Maximum size: 748 bytes + 81 * number of `AuthnContextClassRefs`

- Example of `<lib:AuthnRequest>` message URL-encoded and signed `(772 bytes)`:

```
http://idp.example.com/authn?RequestID=RMvY34pg%2FV9aGJ5yw0HL0AcjcqQF&MajorVersion=1&MinorVersion
=0&IssueInstant=2002-05
15T00%3A58%3A19&ProviderID=http%3A%2F%2Fsp.example.com%2Fliberty%2F&ForceAuthn=true&IsPassive=fal
se&Federate=true&ProtocolProfile=http%3A%2F%2Fprojectliberty.org%2Fprofiles%2Fbrws-
post&AuthnContextClassRef=http%3A%2F%2Fprojectliberty.org%2Fauthnctx%2Fprofiles%2FPassword-over-
HTTP&RelayState=03mhakSms5tMQ0WRDCEzpF7BNcywZa75FwIcSSEPvbkoFxaQHCuNnc5yChIdDlWc7JBV9Xbw3avRBK7VF
sPl2X&SigAlg=http%3A%2F%2Fwww.w3.org%2F2000%2F09%2Fxmldsig%23rsa-
sha1&Signature=EoD8bNr2jEOe%2Fumon6oU%2FZGIIF7gbJAe4MLUUMrD%2BPP7P8Yf3gfdZG2qPJdNAJkzVHGfO8W8DzpQ
%0D%0AsDTTd5VP9MLPcvxbFQoF0CJJmvL26cPsuc54q7ourcH0jJ%2F2UkDq4DAlYlZ5kPIg%2BtrykgLz0U%2BS%0D%0ANqp
NHkjh6W3YkGv7RBs%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]">
                <lib:ProviderID>[ProviderID]</lib:ProviderID>
                <saml:NameIdentifier NameQualifier="[NameQualifier]"
                            Format="[NameFormat]>
                        [Name]
                </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, Name`

3.1.2.1.4 URL-Encoded <lib:LogoutNotification>

The original `<lib:LogoutNotification>` message:

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

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

403 `RequestID, MajorVersion, MinorVersion, IssueInstant, ProviderID,`
404 `NameQualifier, NameFormat, Name`

405 3.1.3 Provider Metadata

406 The majority of the Liberty profiles defined in this document rely on metadata that specify the
407 policies that govern the behavior of the service provider or identity provider. These provider
408 metadata are typically shared out of band between an identity provider and a service provider prior to
409 the exchange of Liberty protocol messages. The provider metadata relevant to each profile are listed
410 in this document at the beginning of the profile category. Refer to [LibertyProtSchema] for a
411 complete enumeration of the Liberty provider metadata elements and their associated schema.

412 3.2 Single Sign-On and Federation Profiles

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

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

420 • `ProviderID` — Used to uniquely identify the service provider to the identity provider and
421 is documented in these profiles as “service provider ID.”

422 • `SingleSignOnServiceURL` — The URL at the identity provider that the service provider
423 should use when sending single sign-on and federation requests. It is documented in these
424 profiles as “single sign-on service URL.”

425 • `AssertionConsumerServiceURL` — The URL at the service provider that an identity
426 provider should use when sending single sign-on or federation responses. It is documented in
427 these profiles as “assertion consumer service URL.”

428 • `SOAPEndPoint` — The SOAP endpoint location at the service provider or identity provider
429 to which Liberty SOAP messages are sent.

430 3.2.1 Common Interactions and Processing Rules

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

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

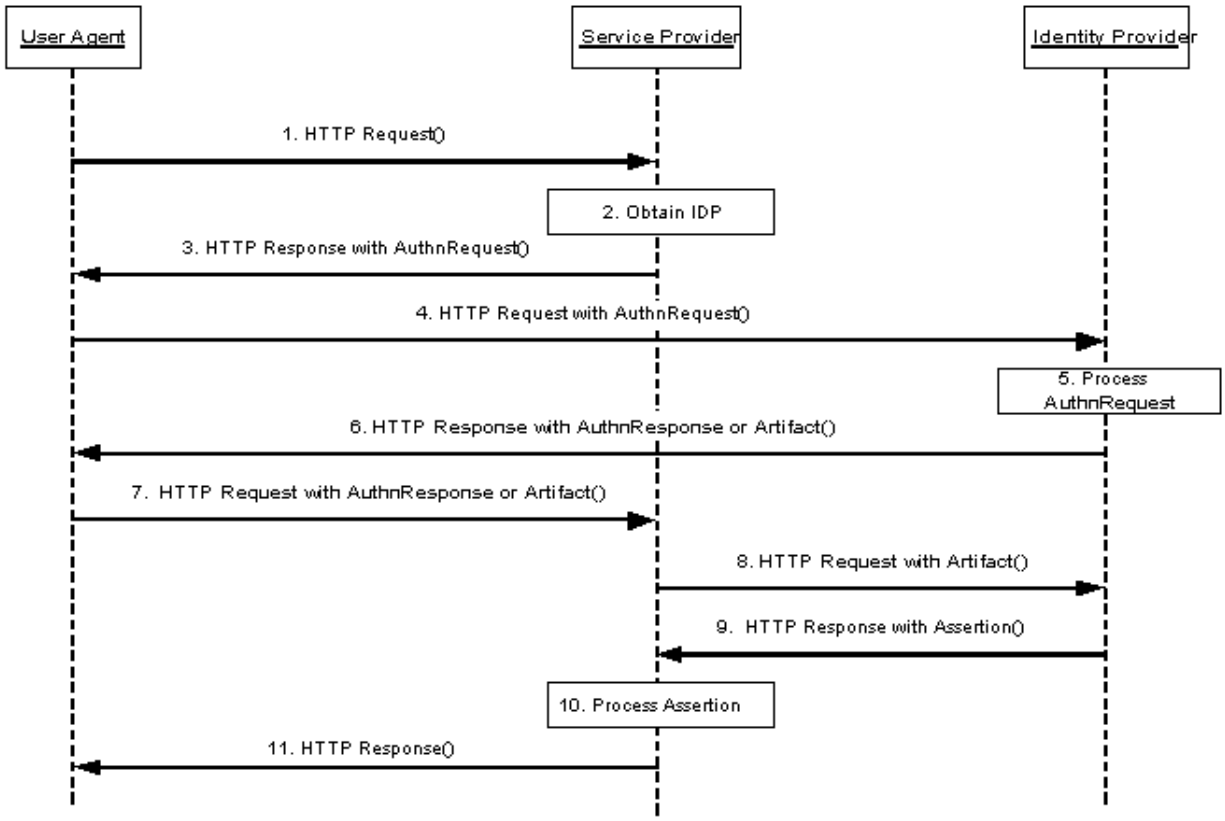


Figure 1: Basic single sign-on profile

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 URI contain a `<query>` component at its end where

`<query>=...LRURL=<return URL>...`

The `<query>` component can be used to convey information about the originally requested resource at the service provider. It is RECOMMENDED that the `<query>` parameter be named `LRURL` and its value be the URL originally requested by the user agent.

It is RECOMMENDED that the HTTP request be made over either SSL 3.0 (see [SSLv3]) or TLS 1.0 (see [RFC2246]) to maintain confidentiality and message integrity in step 1.

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.

Step 3: HTTP Response with <AuthnRequest>

In step 3, the service provider's intersite transfer service responds and redirects the user agent to the single sign-on 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 sign-on service URL.
- The identity provider's single sign-on service URL MUST specify `https` as the URL scheme; if another scheme is specified, the service provider MUST NOT redirect to the identity provider.

  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 Location HTTP header MUST include a `<query>` component containing the `<lib:AuthnRequest>` protocol message as defined in [LibertyProtSchema] with formatting as specified in 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, `LRURL` 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 know how to formulate the HTTP response to the user agent in step 11.

The HTTP response MUST take the following form:

```
<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>
```

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.

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 attached to the URL fulfilling the redirect request.

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, this point is when authentication at the identity provider could occur. However, whether the identity provider should authenticate the Principal 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.

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.

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>`, a SAML artifact, or an error.

The form and contents of the HTTP request in this step are profile-dependent.

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 [SAMLBind].

The `<samlp:Request>` MUST be digitally signed by the service provider.

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 `<lib:Assertion>`. Otherwise, it returns an appropriate error code, as defined within the "SOAP binding for SAML" (see [SAMLBind]).

The `<samlp:Response>` message MAY be digitally signed. The `<lib:Assertion>` contained in the message MUST be digitally signed by the identity provider.

The `<lib:Assertion>` elements contained within the `<samlp:Response>` message returned by the identity provider MUST include a `<lib:SPProvidedNameIdentifier>` element if one has been defined. When the identity provider returns multiple assertions within `<samlp:Response>`, it MUST return exactly one `<lib:Assertion>` for each SAML artifact found in the corresponding `<samlp:Request>` element. The case where fewer or greater number of assertions is returned within the `<samlp:Response>` element MUST be treated as an error state by the service provider.

The identity provider MUST return a response with zero assertions if a `<samlp:Request>` is received from any service provider other than the service provider for which the SAML artifact was originally issued.

The `<saml:ConfirmationMethod>` element of the assertion MUST be set to the value specified in [SAMLCore] for "SAML Artifact," and the `<saml:SubjectConfirmationData>` element MUST be present with its value being the SAML artifact supplied to obtain the assertion.

Step 10: Process Assertion

In step 10, the service provider processes the `<lib: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 `<lib:Assertion>` must be verified.

The service provider processing of the assertion MUST adhere to the rules defined in [SAMLCore] 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 the `<saml:advice>`. Similarly, the information in the `<lib:RelayState>` element MAY be obtained and used in further processing by the service provider.

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 Browser Artifact Profile

The Liberty browser 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 [SAMLBind]. See Figure 2.

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 browser 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 browser artifact profile for single sign-on.

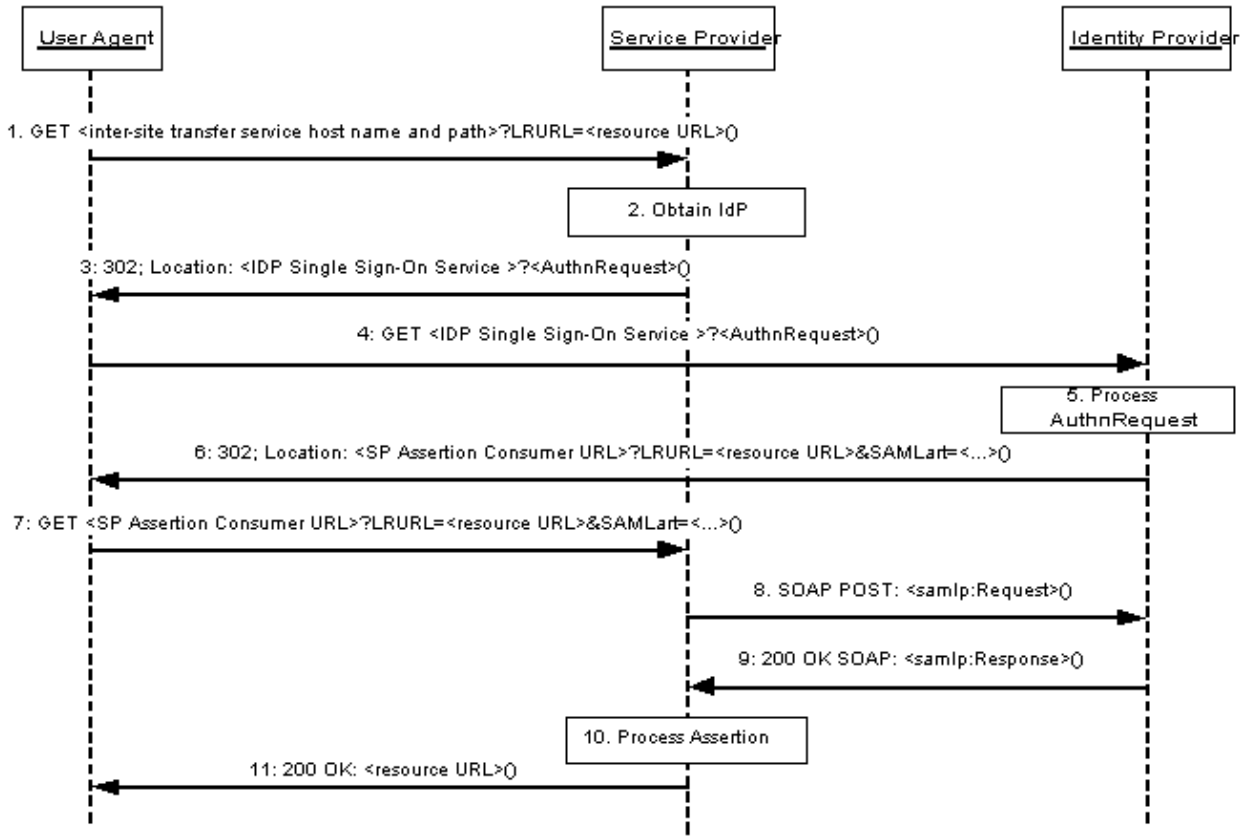


Figure 2: Liberty browser artifact profile for single sign-on

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 3.2.1 for the single sign-on profiles MUST be followed. Additionally, the following rules MUST be observed as they relate to steps 6 and 7:

Step 6: Redirecting to the Service Provider

In step 6, the identity provider performs a redirection to the service provider's assertion consumer service URL including a SAML artifact in the `<query>` component of the URL.

The redirection MUST adhere to the following rules:

- The Location HTTP header MUST be set to the service provider's assertion consumer service URL, the value of which was determined based upon the `<lib:ProviderID>` element of the `<lib:AuthnRequest>` message.
- The service provider's assertion consumer 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>` parameter `SAMLart`, the value of which is the SAML artifact on success or empty on failure. Additionally, if the `<lib:AuthnRequest>` processed in step 5 included a value for the `<lib:RelayState>`

element, then a parameter named `LRURL` with a value set to the value of the `<lib:RelayState>` element MUST be included in the `<query>` component.

The HTTP response MUST take the following form:

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

where

`<Service Provider assertion consumer service URL>`
This element provides the host name, port number, and path components of an assertion consumer service URL at the service provider.

`<query>= …SAMLart=<SAML Artifact>…LRURL=<resource URL> …`
At least one SAML artifact MUST be included in the `<query>` component. A single `LRURL` MUST be included if a value for `<RelayState>` was provided in the `<lib:AuthnRequest>`. If more than one SAML artifact is included the `<query>` component, all artifacts MUST have the same `IdentityProviderID`.

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)` stands for 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 := IdentityProviderID AssertionHandle
IdentityProviderID:= 20-byte_sequence
AssertionHandle   := 20-byte_sequence
```

`IdentityProviderID` 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 `IdentityProviderID` 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 `IdentityProviderID` 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 `IdentityProviderID` 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. The domain name used within this URL is registered with an appropriate authority and administered by the identity provider.
- The identity provider constructs the `IdentityProviderID` component of the artifact by taking the SHA-1 hash of the identification URL.
- The identity provider SHOULD employ best effort to ensure the uniqueness of the generated `IdentityProviderID`.

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 values of at least eight bytes in size. These values should be padded to a total length of 20 bytes.

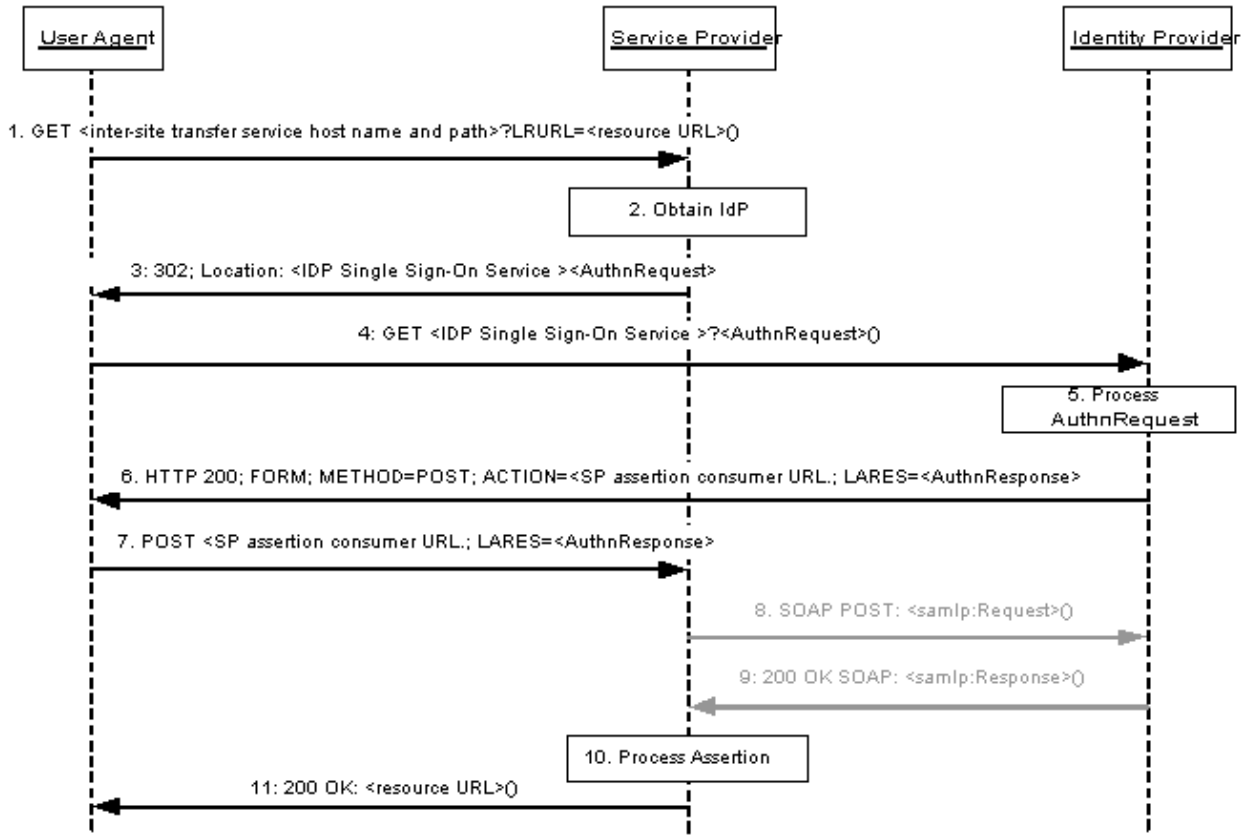
3.2.3 Liberty Browser POST Profile

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 [SAMLBind].

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.



652 **Figure 3: Liberty browser POST profile for single sign-on**

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

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

658 **Step 6: Generating and Supplying the <AuthnResponse>**

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

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

668 Multiple `<lib:Assertion>` elements MAY be included in the response. The identity provider
669 MUST digitally sign all the assertions included in the response.

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

672 **Step 7: Posting the Form Containing the <AuthnResponse>**

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

675 ### 3.2.4 Liberty WML POST Profile

676 The Liberty WML POST profile relies on the use of WML events to instruct a WML browser to
677 submit a HTTP form. This profile is an adaptation of the "Browser/form post profile" for SAML as
678 documented in [SAMLBind]. See Figure 4.

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

681 URI: `http://projectliberty.org/profiles/wml-post`

682 WML browsers are typical on mobile handsets. The browsers on such handsets communicate via a
683 dedicated proxy, a WAP gateway. This proxy converts the Wireless Session Protocol of the handset
684 into HTTP. Note: The service provider and identity provider will be contacted using only HTTP.

685 The WML profile described in this section allows for the transportation of signed Liberty messages
686 that are up to approximately 1100 bytes; the length is limited by the overall size of the WML deck.
687 Many WAP browsers do not accept WML decks that are larger than 1300 bytes (after WML
688 tokenizing).

689 A user agent for this profile, typically a standard WAP browser on a mobile handset, MUST support
690 WAP WML 1.0, 1.1, 1.2, or 1.3 (see [WML1.3]) in addition to the features listed in 3.1.

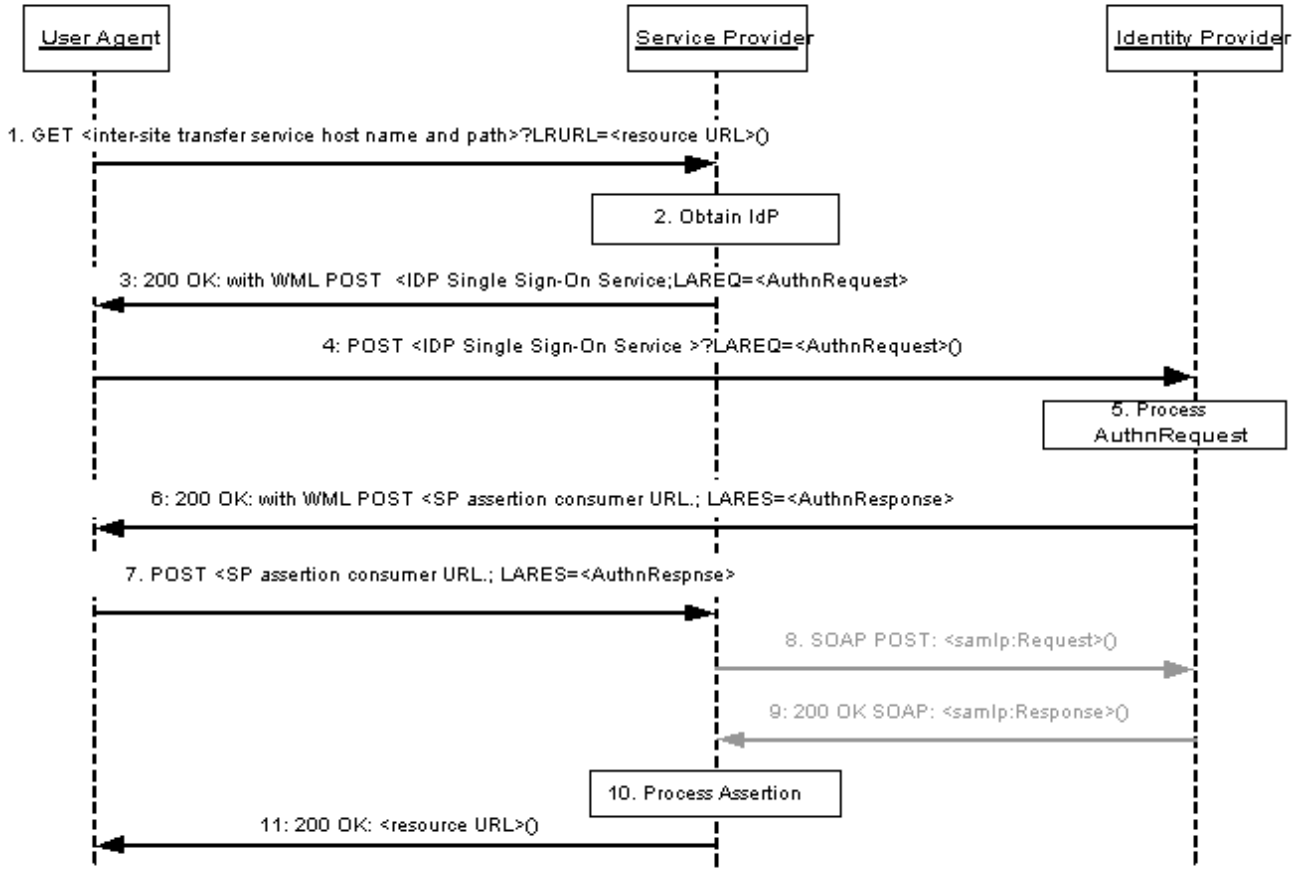


Figure 4: Liberty WML POST profile for single sign-on

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 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, 4, 6, and 7:

Step 3: HTTP Response with <AuthnRequest>

In step 3, the service provider's intersite transfer service responds and instructs the user agent to POST an `<lib:AuthnRequest>` to the single sign-on service URL at the identity provider.

The form contents MUST contain 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 (refer to [RFC2045]) to the `<lib:AuthnRequest>` and all its elements. The identity provider's single sign-on service URL used as the target of the form POST MUST specify `https` as the URL scheme; if another scheme is specified, the service provider MUST NOT issue the POST of the `<lib:AuthnRequest>` to the identity provider.

> Note: One method for seamlessly instructing the user agent to POST the `<lib:AuthnRequest>` is to include a WML deck (see Chapter 17 in [HTML4]) within the HTTP 200 response. The following is an example of how the WML code could be structured:

```
…
<wml>
<card id="redirect" title="Log In">
  <onenterforward>
    <go method="post" href="<Identity Provider Single Sign-On service URL>" >
  <postfield name="LAREQ" Value="(<lib:AuthnRequest>)" />
</go>
  </onenterforward>
  <onenterbackward>
    <prev/>
  </onenterbackward>
  <p>
    Contacting IdP. Please wait…
  </p>
…
</card>
…
</wml>
```

It is recommended that the `<go>` element be contained within a `<onenterforward>` element of the first `<card>` in the WML deck. The `<go>` element will ensure that the browser will post the authentication request as soon as the WML code is processed. In addition it is recommended to add an `<onenterbackward>` element to ensure that a Principal will not be presented with the redirect card when navigating backwards.

Step 4: HTTP Request with <AuthnRequest>

In step 4, the user agent issues the HTTP POST request containing the `<lib:AuthnRequest>` to the identity provider.

Step 6: HTTP Response with <AuthnResponse>

In step 6, the identity provider's single sign-on service instructs the user agent to POST a `<lib:AuthnResponse>` to the assertion consumer service URL at the service provider.

The form MUST be constructed so that it requests a POST to the service provider's assertion consumer service URL with the 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 its elements. Multiple SAML assertions MAY be included in the response. The identity provider MUST digitally sign the assertions included in the response. 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.

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

Note: As in step 3, one way of achieving this step is to use a WML deck.

Step 7: HTTP POST with <AuthnResponse>

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

3.2.5 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 a 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 as well as Liberty-enabled proxies.

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`

All LECPs, in addition to meeting the common requirements for profiles in 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 they make. The preferred method is the Liberty-Enabled header. The formats of the Liberty-Enabled header and User-Agent header entry are defined 3.2.5.1.

3.2.5.1 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 [RFC 2616].

```
Liberty-LECP = "Liberty-LECP" ":" 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 [RFC 2616]. `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. 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 simply advertises what version it does support, and 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, namespace collisions can be effectively prevented without having to introduce yet another centralized registration agency.

LECPs MAY include the Liberty-Agent header in their 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 just using User-Agent is that LECP may be a Liberty-enabled proxy. In that 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-LECP 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-LECP` 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.

3.2.5.2 Interactions

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

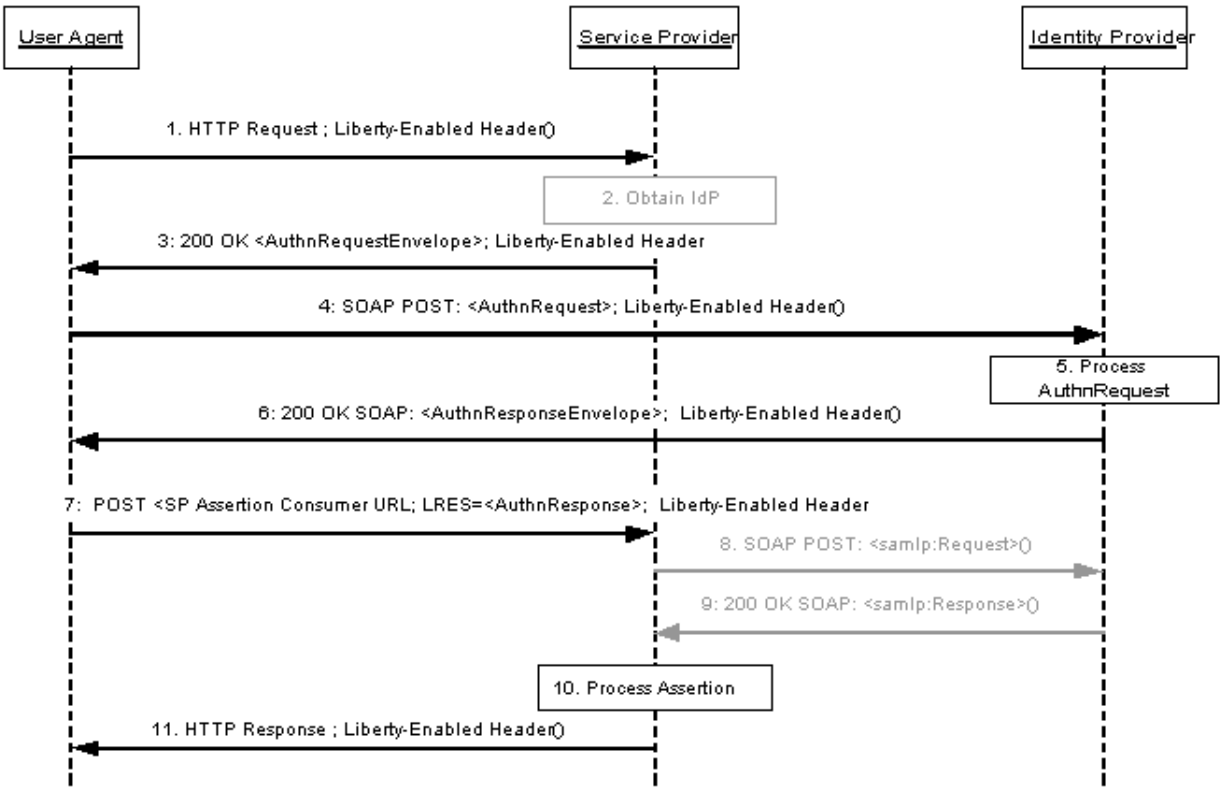


Figure 5: Liberty-enabled client and proxy profile for single sign-on

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 `<lib:Assertion>`, respectively, for mutual authentication.

When implementing this profile, processing rules for steps 5, 10, and 11 defined in 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:

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 a 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.

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]. The `<lib:AuthnRequestEnvelope>` MUST be encoded by applying a base64 transformation (refer to [RFC2045]) to the `<lib:AuthnRequestEnvelope>` and all its elements.

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 3.2.5.1. 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 provide its assertion consumer service URL 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="http://projectliberty.org/schemas/core/2002/08/">
   <lib:AssertionConsumerServiceURL>
       https://service-provider.com/LibertyLogin
   </lib:AssertionConsumerServiceURL>
   <lib:IDPList >
     . . . IdP list goes here . . .
   </lib:IDPList>
   <lib:AuthnRequest >
     . . . AuthnRequest goes here . . .
   </lib:AuthnRequest>
 </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.

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 named `LARES` with its value the `<lib:AuthnResponse>` protocol message as defined in [LibertyProtSchema] with formatting as specified in 3.1.2.

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`.

> Note: Identity providers that wish to authenticate LECPs via name and password exchange can do so by challenging the LECP using HTTP authentication mechanisms. Because the structure of the LECP profile requires all communications to occur over `https`, the name and password pair can be securely sent either over HTTP basic or digest authentication.

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, 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].

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:AuthnResonseEnvelope>` 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>` obtained from the `<lib:AuthnRequestEnvelope>` in step 4. The POST MUST be a form that contains the field `LARES` with the value being the

942 `<lib:AuthnResponse>` protocol message as defined in [LibertyProtSchema]. The
943 `<lib:AuthnResponse>` MUST be encoded by applying a base64 transformation (refer to
944 [RFC2045]) to the `<lib:AuthnResponse>` and all its elements. The service provider's assertion
945 consumer service URL used as the target of the form POST MUST specify `https` as the URL
946 scheme; if another scheme is specified, it MUST be treated as an error by the identity provider.

947 If the LECP discovers an error (for example, syntax error in identity provider response), the LECP
948 MUST return to the service provider a `<lib:AuthnResponse>` with a `<samlp:Status>`
949 indicating the appropriate error element as defined in [LibertyProtSchema]. The
950 `<lib:AuthnResponse>` containing the error status MUST be sent using a POST to the service
951 provider's assertion consumer service URL. The POST MUST be a form that contains the field
952 named `LARES` with its value being the `<lib:AuthnResponse>` protocol message as defined in
953 [LibertyProtSchema] with formatting as specified 3.1.2. Any `<lib:AuthnResponse>` messages
954 created by the identity provider MUST not be sent to the service provider.

955 3.3 Register Name Identifier Profile

956 This section defines the profile by which a service provider may register a name identifier for a
957 Principal that the identity provider MUST use when communicating with the service provider about
958 that Principal. This message exchange is optional. When it is not used, the identity provider will
959 always communicate to the service provider about the Principal using the name identifier provided
960 by the identity provider (that is, `<IDPProvidedNameIdentifier>`). Only one profile is specified:
961 SOAP/HTTP-based.

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

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

964 The register name identifier profile makes use of the following metadata element, as defined in
965 [LibertyProtSchema]:

966 - `SOAPEndPoint` — The SOAP endpoint location at the service provider or identity
967 provider to which Liberty SOAP messages are sent.

968 3.3.1 SOAP/HTTP-Based Profile

969 The steps involved in the SOAP/HTTP-based profile MUST utilize the SOAP binding for Liberty as
970 defined in 2.1. See Figure 6.

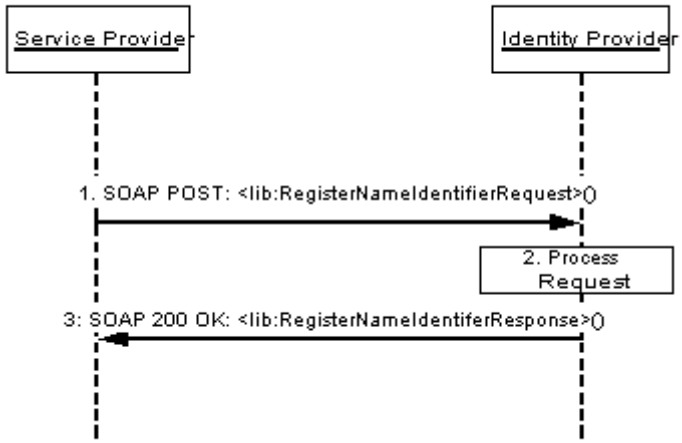


Figure 6: SOAP/HTTP-based profile for registering name identifiers

Step 1: Request to Register Name Identifier

In step 1, the service provider sends a `<lib:RegisterNameIdentifierRequest>` protocol message to the identity provider's SOAP endpoint specifying both the `<lib:IDPProviderNameIdentifier>` and the `<lib:SPProviderNameIdentifier>` as specified in [LibertyProtSchema].

Step 2: Response to Register Name Identifier

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

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 3.4.1 and 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 a 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 a 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 [LibertyProtSchema]:

- `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 3.4.1.1 and 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 7.

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

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

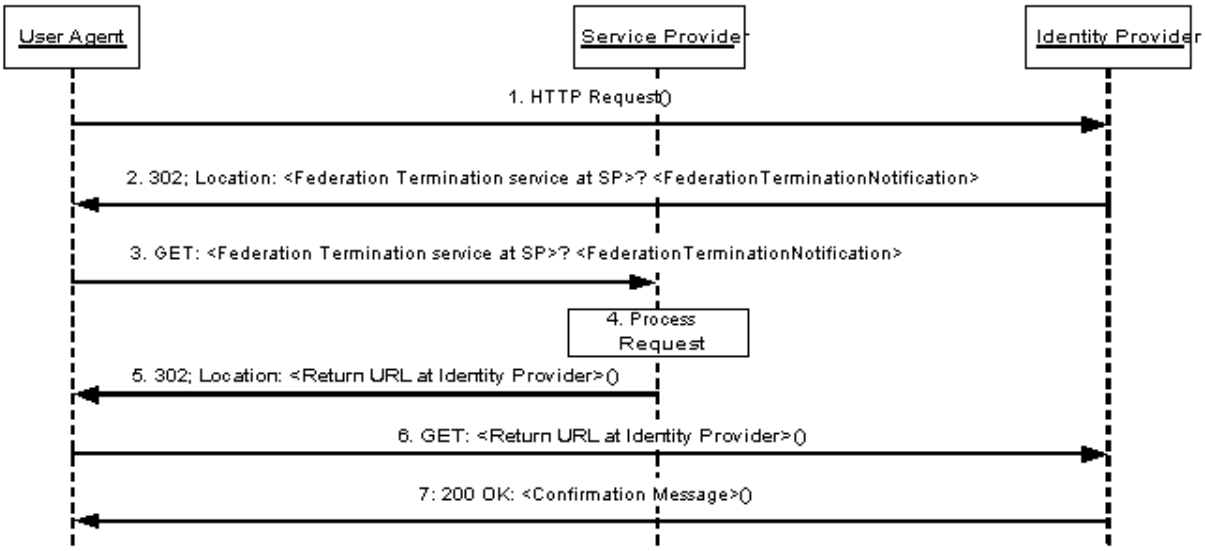


Figure 7: HTTP-redirect-based profile for federation termination

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.

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 whom 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.

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 3.1.2.

Note: Additionally, the URL-encoded `<lib:FederationTerminationNotification>` message MAY also include a parameter named `RELAYSTATE` with a value set to the URL (and/or other information) to be used by the identity provider in the HTTP response to the user agent at the completion of federation termination in step 7.

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.

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:FederationTermintationNotification>` information attached to the URL fulfilling the redirect request.

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>`.

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 Return URL >?<query>
<other HTTP 1.0 or 1.1 components>
```

where

`<Identity Provider Return URL>`
This element provides the host name, port number, and path components of the return URL at the

identity provider.

`<query>= ...RELAYSTATE=<...>`

The `<query>` component MUST contain the identical `RELAYSTATE` parameter and its value that was received in the URL-encoded federation termination 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 `<query>` component.

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.

Step 7: Confirmation

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

3.4.1.2 SOAP/HTTP-Based Profile

The SOAP/HTTP-based profile relies on using asynchronous SOAP over HTTP to communicate federation termination notification messages from the identity provider to the service provider. See Figure 8.

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

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

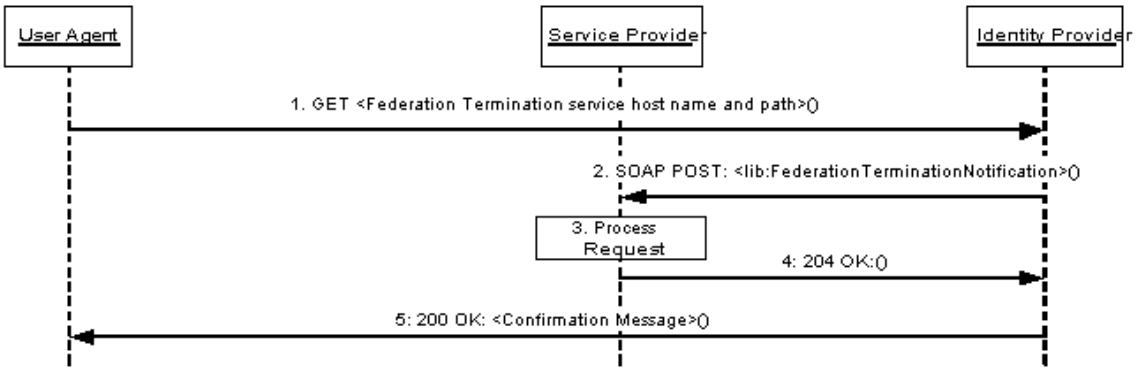


Figure 8: SOAP/HTTP-based profile for federation termination

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.

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 whom 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.

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.

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>`.

Step 4: Responding to the Notification

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

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 3.4.2.1 and 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.

3.4.2.1 HTTP-Redirect-Based Profile

The HTTP-redirect-based profile relies on the use of a 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 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`

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`

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 notification 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 3.5.1 and 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-Redirect-Based:** Relies on using HTTP 302 redirects to communicate logout notifications from an identity provider to the service providers.
  - **HTTP-GET-Based:** Relies on using HTTP GET requests to communicate logout notifications from an identity provider to the service providers.
  - **SOAP/HTTP-Based:** Relies on asynchronous SOAP over HTTP messaging to communicate a logout notifications from an identity provider to the service providers.
- **Single Logout Initiated at Service Provider**
  - **HTTP-Redirect-Based:** Relies on a HTTP 302 redirect to communicate a logout notification with the identity provider.
  - **SOAP/HTTP-Based:** Relies on asynchronous SOAP over HTTP messaging to communicate a logout notification from a service provider to an identity provider.

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

- `SingleLogOutServiceURL` — The URL at the service provider or identity provider to which single logout notifications 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 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.

3.5.1 Single Logout Initiated at Identity Provider

The profiles in 3.5.1.1 through 3.5.1.3 are specific to single logout when initiated by a user agent at the identity provider.

3.5.1.1 HTTP-Redirect-Based Profile

The HTTP-redirect-based profile uses HTTP 302 redirects to communicate a logout notification 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 notification via the HTTP redirect profile. See Figure 9.

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

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

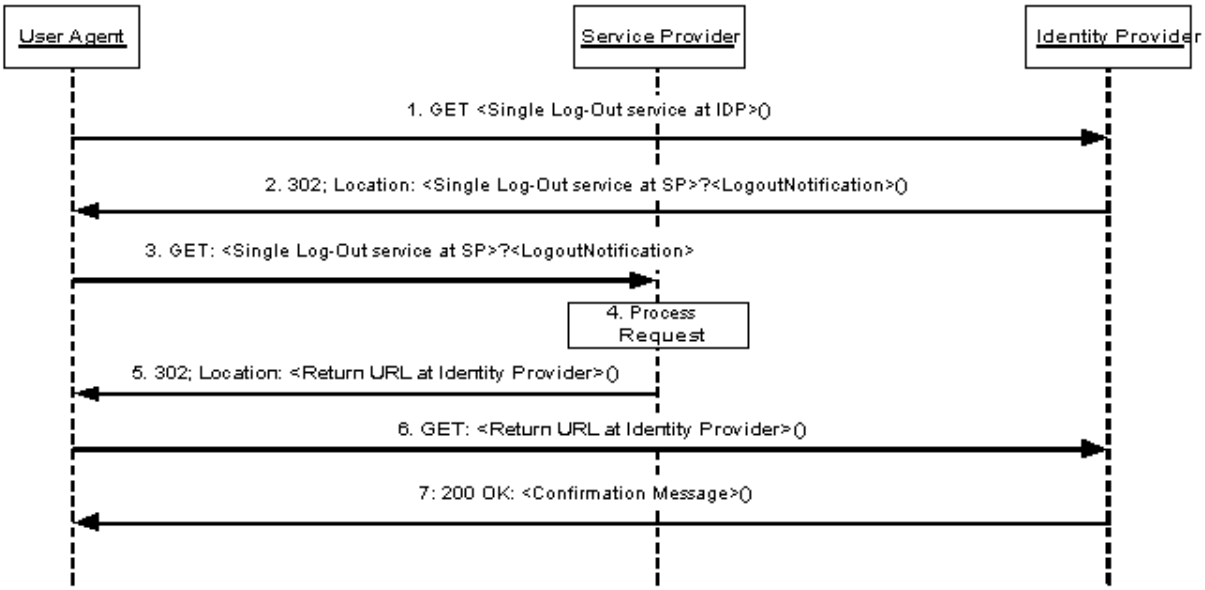


Figure 9: HTTP-redirect-based profile for single logout initiated at identity provider

Note: Steps 2 through 6 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 notification via the HTTP redirect profile.

> Note: [RFC 2616] 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.

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.

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:LogoutNotification>` protocol message as defined in [LibertyProtSchema] with formatting as specified in 3.1.2.

  Note: Additionally, the URL-encoded `<lib:LogoutNotification>` message MAY also include a parameter named `RELAYSTATE` with a value set to the URL (and/or other information) to be used by the identity provider in the HTTP response to the user agent at the completion of federation termination in step 7.

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 LogoutNotification>…`
: The `<query>` MUST contain a single logout notification request.

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:LogoutNotification>` information attached to the URL fulfilling the redirect request.

Step 4: Processing the Notification

In step 4, the service provider MUST process the `<lib:LogoutNotification>` 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:LogoutNotification>`.

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:LogoutNotification>` 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 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 Return URL>?<query>
<other HTTP 1.0 or 1.1 components>
```

where

`<Identity Provider Return URL>`
: This element provides the host name, port number, and path components of the return URL at the identity provider.

`<query>= …RELAYSTATE=<…>`
: The `<query>` component MUST contain the identical `RELAYSTATE` parameter and its value that was received in the URL-encoded logout notification 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 `<query>` component.

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.

Step 7: Confirmation

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

3.5.1.2 HTTP-GET-Based Profile

The HTTP-GET-based profile uses HTTP GET requests to communicate logout notifications 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 notifications via the HTTP GET profile. See Figure 10.

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

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

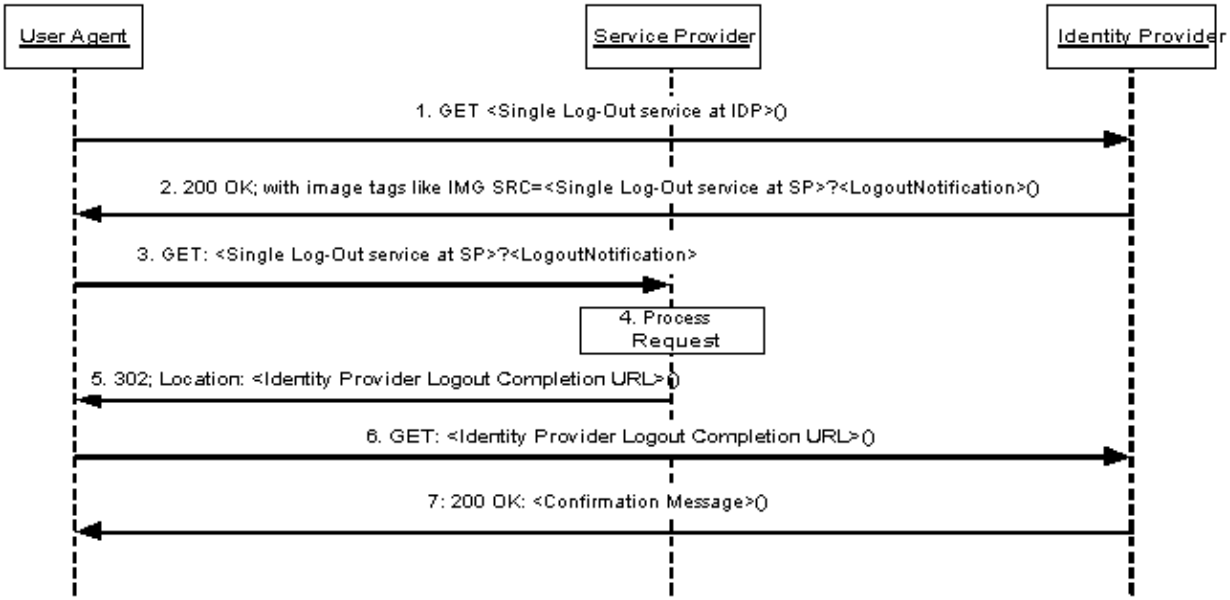


Figure 10: HTTP-GET-based profile for single logout initiated at identity provider

Note: Steps 3 through 7 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 notification via the HTTP-GET-based profile.

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.

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:LogoutNotification>` protocol message as defined in [LibertyProtSchema] with formatting as specified in 3.1.2.

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:logoutNotification>` 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).

Step 4: Processing the Notification

In step 4, the service provider MUST process the `<lib:LogoutNotification>` 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:LogoutNotification>`.

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>
<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.

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.

Step 7: Confirmation

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

3.5.1.3 SOAP/HTTP-Based Profile

The SOAP/HTTP-based profile uses asynchronous SOAP over HTTP messaging to communicate a logout notification 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 notification via the SOAP/HTTP-based profile. See Figure 11.

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

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

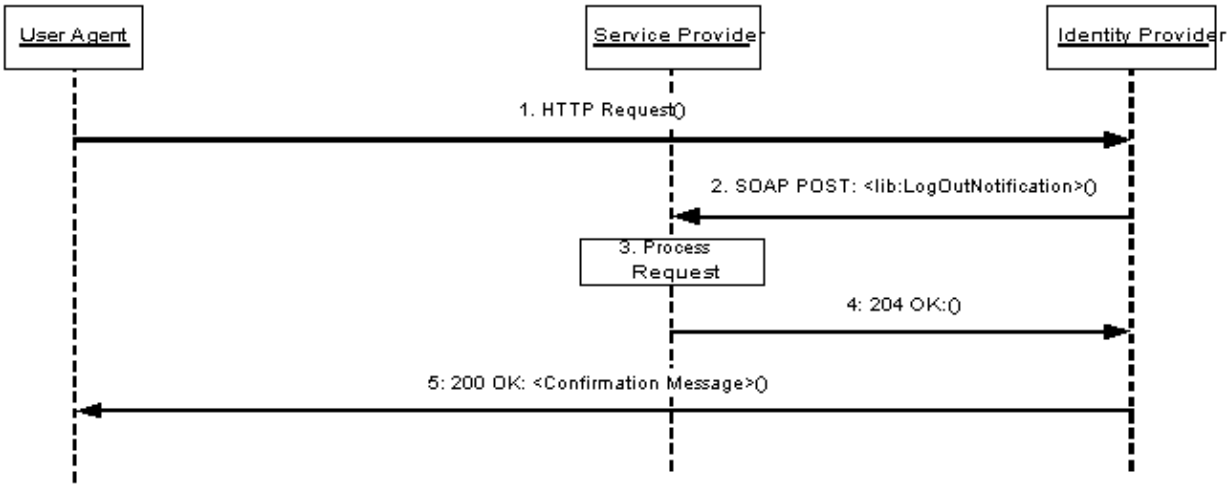


Figure 11: SOAP/HTTP-based profile for single logout initiated at identity provider

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 notifications via the SOAP/HTTP asynchronous message profile.

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.

Step 2: Notification of Logout

In step 2, the identity provider sends an asynchronous SOAP over HTTP notification message 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:logoutNotification>` 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 single logout notification message to the service provider.

Step 3: Processing the Notification

In step 3, the service provider MUST process the `<lib:LogoutNotification>` 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:LogoutNotification>`.

Step 4: Responding to the Notification

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

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 3.5.2.1 and 3.5.2.2 are specific to when the Principal initiates the single logout notification process at the service provider.

3.5.2.1 HTTP-Redirect-Based Profile

The HTTP-redirect-based profile relies on using a HTTP 302 redirect to communicate a logout notification with the identity provider. The identity provider will then communicate a logout notification to each service provider with which it has established a session for the Principal using the service provider's preferred profile for logout notification from the identity provider (see 3.5.1). See Figure 12.

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

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

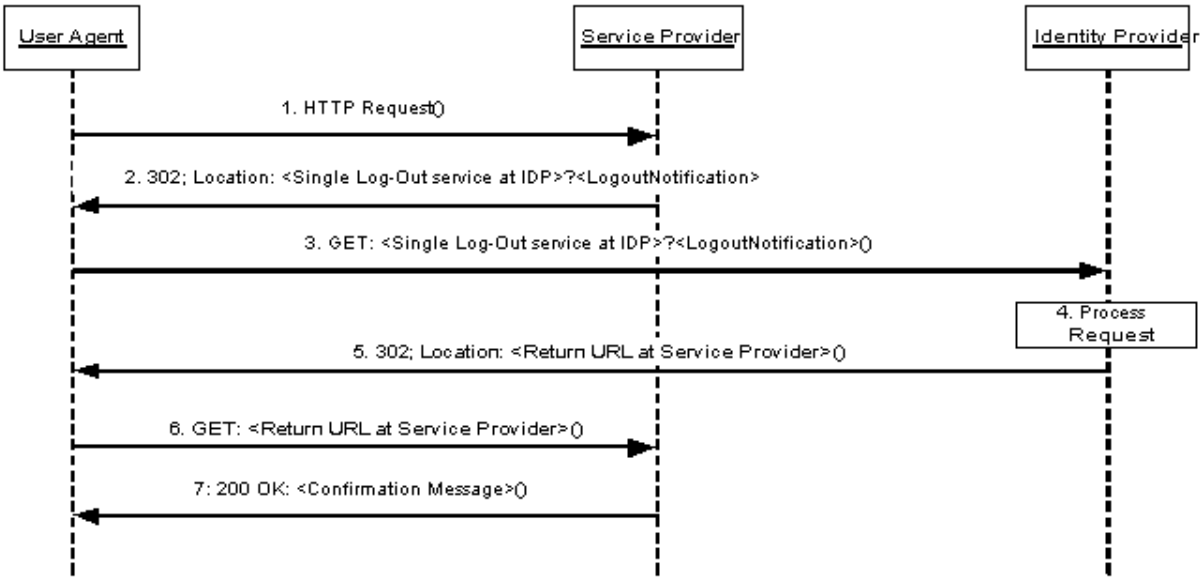


Figure 12: HTTP-redirect-based profile for single logout initiated at service provider

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

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 RECOMMEDED that the service provider terminate that session prior to step 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:LogoutNotification>` protocol message as defined in [LibertyProtSchema] with formatting as specified in 3.1.2.

  Note: Additionally, the URL-encoded `<lib:LogoutNotification>` message MAY also include a parameter named `RELAYSTATE` with a value set to the URL (and/or other information) to be used by the identity provider in the HTTP response to the user agent at the completion of federation termination in step 7.

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 LogoutNotification>…`
: The `<query>` MUST contain a single logout notification request.

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:LogoutNotification>` information attached to the URL fulfilling the redirect request.

Step 4: Processing the Request

In step 4, the identity provider MUST process the `<lib:LogoutNotification>` 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 notification from the identity provider (see 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.

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:LogoutNotification>` 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.

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 service provider.

The HTTP response MUST take the following form:

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

where

`<Service Provider Return URL>`
: This element provides the host name, port number, and path components of the return URL location at the service provider.

`<query>= …RELAYSTATE=<…>`
: The `<query>` component MUST contain the identical `RELAYSTATE` parameter and its value that was received in the URL-encoded logout notification 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 `<query>` component.

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.

Step 7: Confirmation

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

3.5.2.2 SOAP/HTTP-Based Profile

The SOAP/HTTP-based profile relies on using asynchronous SOAP over HTTP messages to communicate a logout notification with the identity provider. The identity provider will then communicate a logout notification to each service provider it has established a session with for the Principal via the service provider's preferred profile for logout notification from the identity provider (see 3.5.1). See Figure 13.

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

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

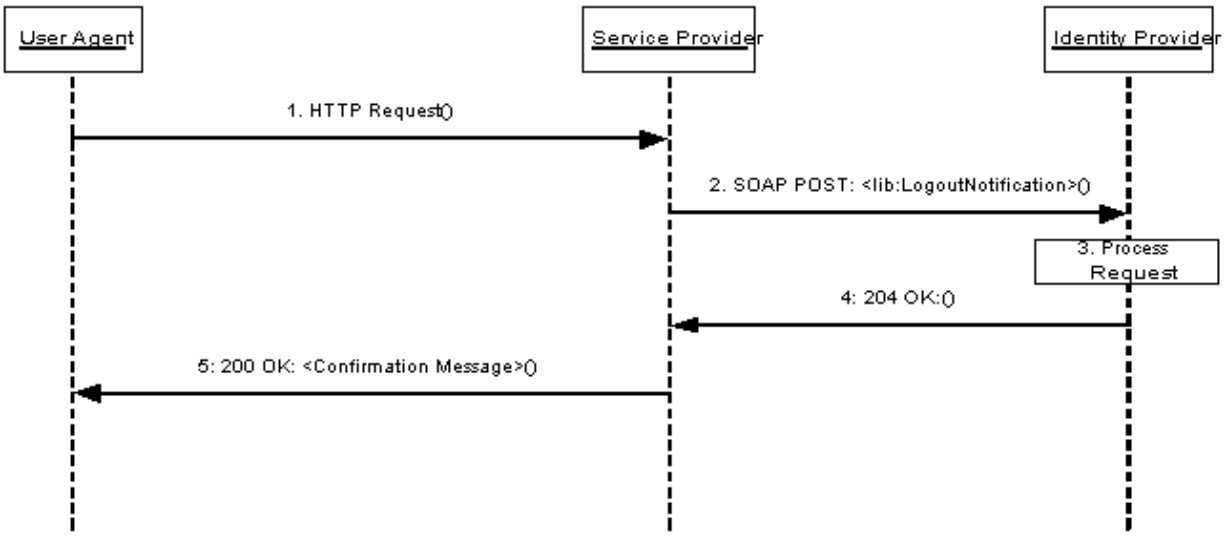


Figure 13: SOAP/HTTP-based profile for single logout initiated at service provider

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

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.

Step 2: Notification of Logout

In step 2, the service provider sends an asynchronous SOAP over HTTP notification message to the identity provider's SOAP endpoint. The SOAP message MUST contain exactly one `<lib:logoutNotification>` 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 effort to resolve the fault condition and resend the single logout notification message to the identity provider.

Step 3: Processing the Notification

In step 3, the identity provider MUST process the `<lib:LogoutNotification>` 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 notification from the identity provider. If any of the active service provider's preferred single logout profile is HTTP-redirect-based or HTTP-GET-based, then the identity provider MUST respond to the sender of the `<lib:LogoutNotificaton>` message in step 2 with a SOAP fault. The SOAP fault MUST adhere to the following rules;

- The `FaultCode` MUST specify "Server."

- The `FaultString` must specify "Cannot execute Single Log Out using web service."

This SOAP fault condition will indicate to the service provider that it MUST redirect the user agent to the identity provider to achieve single logout.

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.

Step 4: Responding to the Notification

In step 4, the identity provider MUST respond to the `<lib:LogoutNotification>` with a HTTP 204 OK response, unless a SOAP error was sent as specified in step 3.

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 IDs separated by a single white space character. The identity provider IDs MUST adhere to the creation rules as defined in 3.2.2.2. The identity provider ID is a metadata element, as described in 3.1.3 and defined in [LibertyProtSchema].

The cookies MAY be either session or persistent (see [RFC2109]), the implementation of which is a policy and deployment issue of the identity federation network.

3.6.2 Setting the Common Domain Cookie

After an identity provider authenticates a Principal, it MAY set the common domain cookie. See Figure 14. The profile for setting the cookie follows the figure.

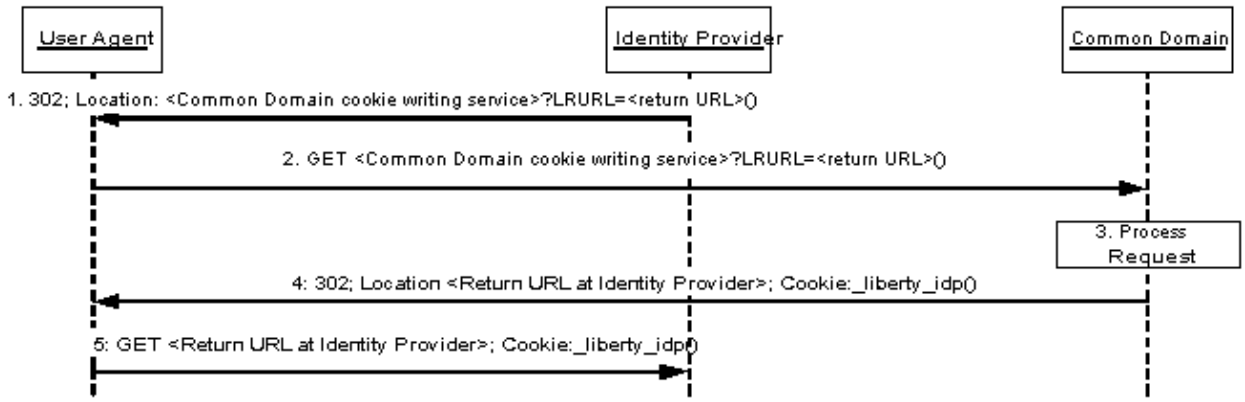


Figure 14: Setting the common domain cookie

Step 1: Redirecting to Cookie Writing Service

In step 1, the identity provider redirects the user agent to a cookie writing service on a HTTP server in the common domain. The redirection MUST adhere to the following rules:

- The Location HTTP header MUST be set to a URL hosting the cookie writing service in the common domain.
- The common domain's cookie writing service URL MUST specify `https` as the URL scheme; if another scheme is specified, the identity provider MUST NOT redirect to the common domain.
- The Location HTTP header MUST include a `<query>` parameter `LRURL`, the value of which MUST be a URL-encoded URL that the cookie writing service will use as the location of a redirect back to the identity provider.

  Note: For a more seamless experience, the `Return-URL` parameter can be the location of a redirect back to the service provider.

The HTTP response MUST take the following form:

```
<HTTP-Version> 302 <Reason Phrase>
<other headers>
Location : https://<Common Domain cookie writing service URL>?<query>
<other HTTP 1.0 or 1.1 components>
```

where

`<Common Domain cookie writing service URL>`
: This element provides the host name, port number, and path components of the cookie writing service URL on the HTTP server in the common domain.

`<query>= …LRURL=<return URL>`
: The return URL description MUST be included in the `<query>` component to indicate the URL to use in the redirect back to the identity provider.

Step 2: Accessing Cookie Writing Service

In step 2, the user agent accesses the common domain cookie writing service with the return URL (`LRURL`) information attached to the URL fulfilling the redirect request.

The common domain cookie `_liberty_idp` is presented if it exists.

Step 3: Process the Request

In step 3, 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.

Step 4: Redirecting to Identity Provider Return URL

In step 4, the common domain cookie writing service responds with a redirection of the user agent back to the Identity provider using the `LRURL` specified in step 2.

The redirection MUST adhere to the following rules:

- The Location HTTP header MUST be set to the return URL for the identity provider.
- The Set-Cookie header MUST be included in the response, specifying `_liberty_idp` as the name.
- The identity provider return URL MUST specify `https` as the URL scheme; if another scheme is specified, the common domain cookie writing service MUST NOT redirect to the identity provider.

The HTTP response MUST take the following form:

```
<HTTP-Version> 302 <Reason Phrase>
Set-Cookie: _liberty_idp=<value>;Version="1";Domain"…";Max-Age=…
<other headers>
Location : https://<Identity Provider Return URL>
<other HTTP 1.0 or 1.1 components>
```

where

`<Identity Provider Return URL>`
: This element provides the host name, port number, and path components of the return URL at the identity provider.

`Set-Cookie:`
: The Set-Cookie header MUST include the cookie `_liberty_idp` with its value as previously defined in 3.6.1. The cookie MUST be marked as `secure`.

Step 5: Accessing Identity Provider Return URL

In step 5, the user agent accesses the identity provider's return URL and fulfills the redirect request.

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. See Figure 15.

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.

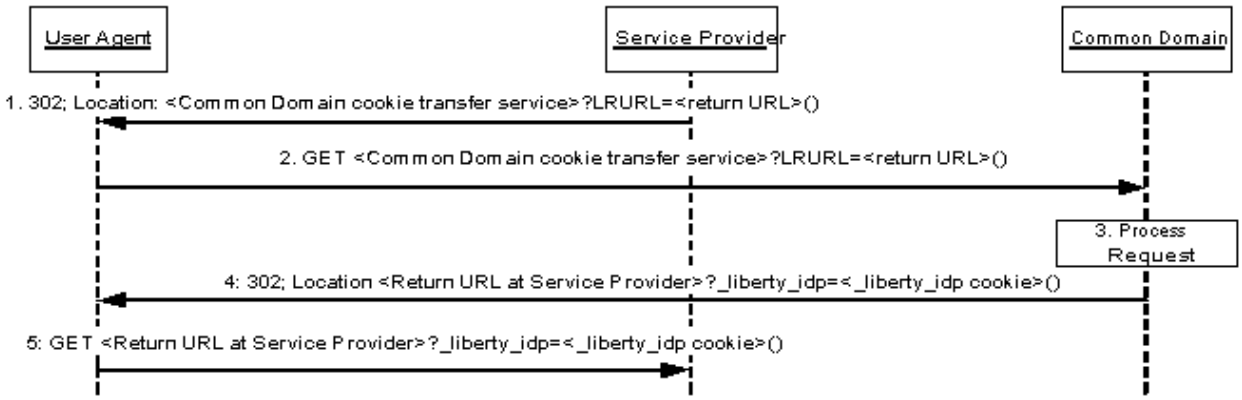


Figure 15: Obtaining the common domain cookie

Step 1: Redirecting to Cookie Transfer Service

In step 1, the service provider redirects the user agent to a cookie transfer service on a HTTP server in the common domain.

The redirection MUST adhere to the following rules:

- The Location HTTP header MUST be set to a URL hosting the cookie transfer service in the common domain.
- The common domain's cookie transfer service URL MUST specify `https` as the URL scheme; if another scheme is specified, the service provider MUST NOT redirect to the common domain.
- The Location HTTP header MUST include a `<query>` parameter `LRURL`, the value of which MUST be a URL-encoded URL that the cookie transfer service will use as the location of a redirect back to the service provider.

The HTTP response MUST take the following form:

```
<HTTP-Version> 302 <Reason Phrase>
<other headers>
Location : https://<Common Domain cookie transfer service URL>?<query>
<other HTTP 1.0 or 1.1 components>
```

where

`<Common Domain cookie writing service URL>`
: This element provides the host name, port number, and path components of the cookie transfer service URL on the HTTP server in the common domain.

`<query>= ...LRURL=<return URL>`
: The return URL description MUST be included in the `<query>` component to indicate the URL to use in the redirect back to the service provider.

Step 2: Accessing Cookie Transfer Service

In step 2, the user agent accesses the common domain cookie transfer service with the return URL (`LRURL`) information attached to the URL fulfilling the redirect request.

The common domain cookie `_liberty_idp` is presented if it exists.

Step 3: Process the Request

In step 3, the common domain cookie transfer service MUST read the `_liberty_idp` cookie and redirect the user agent back to the service provider with the value of the cookie included as a `<query>` parameter.

Step 4: Redirecting to Service Provider Return URL

In step 4, the common domain cookie transfer service responds with a redirection of the user agent back to the service provider using the `LRURL` specified in step 2. The redirection MUST adhere to the following rules:

- The Location HTTP header MUST be set to return URL for the service provider.
- The common domain's cookie writing service URL MUST specify `https` as the URL scheme; if another scheme is specified, the common domain cookie transfer service MUST NOT redirect to the service provider.
- The Location HTTP header MUST include a `<query>` parameter `_liberty_idp`, whose value MUST be the value of the `_liberty_idp` cookie.

The HTTP response MUST take the following form:

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

where

`<Service Provider Return URL>`
: This element provides the host name, port number, and path components of the return URL at the service provider.

`<query>= …_liberty_idp=<_liberty_idp cookie>`
: The `_liberty_idp` cookie value MUST be included in the `<query>` component to indicate the list of identity providers to the service provider.

Step 5: Accessing Service Provider Return URL

In step 5, the user agent accesses the service provider's return URL with the `_liberty_idp` cookie information attached to the URL and fulfills the redirect request.

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 heavily utilize the SAML protocols. Therefore, the security considerations published along with the SAML specification have direct relevance (see [SAMLCore], [SAMLBind], 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 by no means 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 generator
- 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 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 be able to launch active or passive attacks on the system. A passive attack is one where the attacker does not inject any traffic or modify any 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. Then threats that are specific to individual profiles are considered. In each discussion the threat is described as well as

what countermeasures exist in the profile or what additional countermeasures 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>`.

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 3.2), 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 [SAMLBind]

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 [SAMLBind]

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

4.4.2.2 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 among themselves, they can track the Principal's behavior among them.

Countermeasure: The `<lib:IDPProvidedNameidentifier>` is required to be unique for each identity provider to service provider relationship.

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:LogoutNotification>` 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:LogoutNotification>` message

Description: An Unsigned `<lib:LogoutNotification>` 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:LogoutNotification>` message.

Countermeasure: Sign the `<lib:LogoutNotification>` message.

4.4.6 Identity Provider Introduction

No known threats.

5 References

[Anders] Rundgren, A., "A suggestion on how to implement SAML browser bindings without using Artifacts," **http://www.x-obi.com/OBI400/andersr-browser-artifact.ppt**, August 2001.

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

[LibertyGloss] Mauldin, H., "Liberty Glossary," **https://66.34.4.93/members/technology expert group/draft-liberty-tech-glossary-07.pdf**, April 2002.

[LibertyProtSchema] Beatty, J., "Liberty Protocols and Schemas Specification," **https://66.34.4.93/members/technology 20expert group/architecture/draft-liberty-architecture-protocols-schemas-08.pdf**, April 2002.

[Rescorla-Sec] Rescorla, E., et al., "Guidelines for Writing RFC Text on Security Considerations," **http://www.ietf.org/internet-drafts/draft-rescorla-sec-cons-04.txt**, February 2002.

[RFC1750] Eastlake, D., Croker, S., Schiller, J., "Randomness Recommendations for Security," **ftp://ftp.isi.edu/in-notes/rfc1750.txt**, RFC 1750, December 1994

[RFC1945] Berners-Lee, T., Fielding, R., Frystyk, H., "Hypertext Transfer Protocol -- HTTP/1.0," **ftp://ftp.isi.edu/in-notes/rfc1945.txt**, RFC 1945, May 1996.

[RFC2045] Freed, N., Borenstein, N., "Multipurpose Internet Mail Extensions (MIME) Part One: Format of Internet Message Bodies," **ftp://ftp.isi.edu/in-notes/rfc2045.txt**, RFC 2045, November 1996.

[RFC2109] Kristol, D., Montulli, L., "HTTP State Management Mechanism," **ftp://ftp.isi.edu/in-notes/rfc2109.txt**, RFC 2109, February 1997.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels," **ftp://ftp.isi.edu/in-notes/rfc2119.txt**, RFC 2119, March 1997.

[RFC2246] Dierks, T., Allen, C., "The TLS Protocol Version 1.0," **ftp://ftp.isi.edu/in-notes/rfc2246.txt**, RFC 2246, January 1999.

[RFC2396] Berners-Lee, T., Fielding, R., Masinter, L., "Uniform Resource Identifiers (URI): Generic Syntax," **ftp://ftp.isi.edu/in-notes/rfc2396.txt**, RFC 2396, August 1998.

[RFC2616] Fielding, R., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P., Berners-Lee, T., "Hypertext Transfer Protocol -- HTTP/1.1," **ftp://ftp.isi.edu/in-notes/rfc2616.txt**, RFC 2616, June 1999.

[RFC2617] Franks, J., Hallam-Baker, P., Hostetler, J., Lawrence, S., Leach, P., Luotonen, A., Stewart, L., "HTTP Authentication: Basic and Digest Access Authentication," **ftp://ftp.isi.edu/in-notes/rfc2617.txt**, RFC 2617, June 1999.

[RFC2774] Nielsen, H., Leach P., Lawrence, S., "An HTTP Extension Framework," **ftp://ftp.isi.edu/in-notes/rfc2774.txt**, RFC 2774, February 2000.

[RFC3106] Eastlake, D., Goldstein, T., "ECML v1.1: Field Specifications for E-Commerce," **ftp://ftp.isi.edu/in-notes/rfc3106.txt**, RFC 3106, April 2001

[SAMLBind] Mishra, Prateek et al., "Bindings and Profiles for the OASIS Security Assertion Markup Language (SAML)," **http://www.oasis-open.org/committees/security/docs/draft-sstc-bindings-model-14.doc**, OASIS, April 2002.

[SAMLCore] Hallam-Baker, P., et al., "Assertions and Protocol for the OASIS Security Assertion Markup Language (SAML)," **http://www.oasis-open.org/committees/security/docs/draft-sstc-core-31.pdf**, OASIS, April 2002.

[SAMLGloss] Hodges, J., et al., "Glossary for the OASIS Security Assertion Markup Language (SAML)," **http://www.oasis-open.org/committees/security/docs/draft-sstc-glossary-02.pdf**, OASIS, December 2001.

[SAMLReqs] Platt, D., et al., "SAML Requirements and Use Cases," **http://www.oasis-open.org/committees/security/docs/draft-sstc-saml-reqs-01.pdf**, OASIS, December 2001.

[SAMLSec] Hodges, J., et al., "Security Considerations for the OASIS Security Assertion Markup Language (SAML)," **http://www.oasis-open.org/committees/security/docs/draft-sec-consider-04.pdf**, OASIS, December 2001.

[Schema1] Thompson, H. S., et al., "XML Schema Part 1: Structures," **http://www.w3.org/TR/xmlschema-1/**, W3C Recommendation, May 2001.

[Schema2] Biron, P. V., et al., "XML Schema Part 2: Datatypes," **http://www.w3.org/TR/xmlschema-2**, W3C Recommendation, May 2001.

[ShibMarlena] Erdos, M., "Shibboleth Architecture DRAFT v1.1," **http://middleware.internet2.edu/shibboleth/docs/draft-erdos-shibboleth-architecturel-01.pdf**, June 2001.

[SOAP1.1] Box, D., et al., "Simple Object Access Protocol (SOAP) 1.1," **http://www.w3.org/TR/SOAP**, W3C Note, May 2000.

[SSLv3] Freier, A. O., Karlton, P., Kocher, P., "The SSL Protocol Version 3.0," **http://www.mozilla.org/projects/security/pki/nss/ssl/draft302.txt**, Internet Draft, November 1996.

[WML1.3] **"**Wireless Application Protocol Wireless Markup Language Specification Version 1.3," Wireless Application Protocol Forum, Ltd., **http://www.wapforum.org/**, February 2000

[XMLSig] Eastlake, D., Reagle, J., Solo, D., "XML-Signature Syntax and Processing," **http://www.w3.org/TR/xmldsig-core/**, W3C Recommendation, February 2002.