

1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28

Liberty Bindings and Profiles Specification

Version 1.0

11 July 2002

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

28 **Notice**

29

30 Copyright © 2002 ActivCard; American Express Travel Related Services; America Online, Inc.;
31 Bank of America; Bell Canada; Catavault; Cingular Wireless; Cisco Systems, Inc.; Citigroup;
32 Cyberun Corporation; Deloitte & Touche LLP; EarthLink, Inc.; Electronic Data Systems, Inc.;
33 Entrust, Inc.; Ericsson; Fidelity Investments; France Telecom; Gemplus; General Motors; Hewlett-
34 Packard Company; i2 Technologies, Inc.; Intuit Inc.; MasterCard International; Nextel
35 Communications; Nippon Telegraph and Telephone Company; Nokia Corporation; Novell, Inc.;
36 NTT DoCoMo, Inc.; OneName Corporation; Openwave Systems Inc.; PricewaterhouseCoopers LLP;
37 Register.com; RSA Security Inc; Sabre Holdings Corporation; SAP AG; SchlumbergerSema; Sony
38 Corporation; Sun Microsystems, Inc.; United Airlines; VeriSign, Inc.; Visa International; Vodafone
39 Group Plc; Wave Systems. All rights reserved.

40

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

46

47 Implementation of this Specification may involve the use of one or more of the following United
48 States Patents claimed by AOL Time Warner, Inc.: No.5,774,670, No.6,134,592, No.5,826,242, No.
49 5,825,890, and No.5,671,279. The Sponsors of the Specification take no position concerning the
50 evidence, validity or scope of the claimed subject matter of the aforementioned patents.

51 Implementation of certain elements of this Specification may also require licenses under third party
52 intellectual property rights other than those identified above, including without limitation, patent
53 rights. The Sponsors of the Specification are not and shall not be held responsible in any manner for
54 identifying or failing to identify any or all such intellectual property rights that may be involved in
55 the implementation of the Specification.

56

57 **This Specification is provided "AS IS", and no participant in the Liberty Alliance makes any**
58 **warranty of any kind, express or implied, including any implied warranties of merchantability,**
59 **non-infringement or third party intellectual property rights, and fitness for a particular**
60 **purpose.**

61

62 Liberty Alliance Project
63 Licensing Administrator
64 c/o IEEE-ISTO
65 445 Hoes Lane, P.O. Box 1331
66 Piscataway, NJ 08855-1331, USA

67

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

74

75

76

77

78

78	Table of Contents	
79	1 Introduction	5
80	1.1 Notation.....	5
81	2 Protocol Bindings	5
82	2.1 SOAP Binding for Liberty	6
83	2.2 Example of Message Exchange Using SOAP over HTTP	6
84	3 Profiles.....	6
85	3.1 Common Requirements	7
86	3.1.1 User Agent.....	8
87	3.1.2 Formatting and Encoding of Protocol Messages.....	8
88	3.1.3 Provider Metadata	11
89	3.2 Single Sign-On and Federation Profiles	11
90	3.2.1 Common Interactions and Processing Rules.....	11
91	3.2.2 Liberty Browser Artifact Profile	15
92	3.2.3 Liberty Browser POST Profile.....	18
93	3.2.4 Liberty WML POST Profile.....	19
94	3.2.5 Liberty-Enabled Client and Proxy Profile.....	22
95	3.3 Register Name Identifier Profile.....	27
96	3.3.1 SOAP/HTTP-Based Profile.....	27
97	3.4 Identity Federation Termination Notification Profiles	28
98	3.4.1 Federation Termination Notification Initiated at Identity Provider	29
99	3.4.2 Federation Termination Notification Initiated at Service Provider.....	33
100	3.5 Single Logout Profiles	34
101	3.5.1 Single Logout Initiated at Identity Provider.....	35
102	3.5.2 Single Logout Initiated at Service Provider	41
103	3.6 Identity Provider Introduction.....	45
104	3.6.1 Common Domain Cookie.....	45
105	3.6.2 Setting the Common Domain Cookie	45
106	3.6.3 Obtaining the Common Domain Cookie.....	47
107	4 Security Considerations.....	49
108	4.1 Introduction.....	49
109	4.2 General Requirements.....	50
110	4.2.1 Security of SSL and TLS	50
111	4.2.2 Security Implementation	50
112	4.3 Classification of Threats	50
113	4.3.1 Threat Model.....	50
114	4.3.2 Rogue and Spurious Entities	50
115	4.3.3 Active and Passive Attackers	51
116	4.3.4 Scenarios	51
117	4.4 Threat Scenarios and Countermeasures.....	51
118	4.4.1 Common Threats for All Profiles.....	52
119	4.4.2 Single Sign-On and Federation	52
120	4.4.3 Name Registration.....	55
121	4.4.4 Federation Termination: HTTP-Redirect-Based Profile.....	55
122	4.4.5 Single Logout: HTTP-Redirect-Based Profile	55
123	4.4.6 Identity Provider Introduction.....	55
124	5 References	55
125		
126		

126 1 Introduction

127 This specification defines the bindings and profiles of the Liberty protocols and messages to HTTP-
128 based communication frameworks. This specification relies on the SAML core framework in
129 [[SAMLCore](#)] and makes use of adaptations of the SAML profiles in [[SAMLBind](#)]. A separate
130 specification, [[LibertyProtSchema](#)], is used to define the Liberty protocols and messages used within
131 the profiles. Definitions for Liberty-specific terms can be found in [[LibertyGloss](#)].

132 1.1 Notation

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

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

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

143

144 Example code listings appear like this.

145

146 Note: Non-normative notes and explanations appear like this.

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

- 150 • The prefix `lib:` stands for the Liberty namespace
151 `http://projectliberty.org/schemas/core/2002/08/`
- 152 • The prefix `saml:` stands for the SAML assertion namespace (see [[SAMLCore](#)]).
- 153 • The prefix `samlp:` stands for the SAML request-response protocol namespace (see
154 [[SAMLCore](#)]).
- 155 • The prefix `ds:` stands for the W3C XML signature namespace,
156 `http://www.w3.org/2000/09/xmldsig#` (see [[XMLSig](#)]).
- 157 • The prefix `SOAP-ENV:` stands for the SOAP 1.1 namespace,
158 `http://schemas.xmlsoap.org/soap/envelope` (see [[SOAP1.1](#)]).

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

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

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

164 2 Protocol Bindings

165 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:

- 221 • **Single Sign-On and Federation:** The profiles by which a service provider obtains an
222 authentication assertion from an identity provider facilitating single sign-on and identity
223 federation.
- 224 • **Name Registration Profile:** The profile by which service providers specify a Principal's
225 opaque handle (or *name identifier*) that an identity provider will use when communicating to
226 the service provider about the Principal.
- 227 • **Identity Termination Notification:** The profiles by which service providers and identity
228 providers are notified of federation termination.
- 229 • **Single Logout:** The profiles by which service providers and identity providers are notified of
230 authenticated session termination.
- 231 • **Identity Provider Introduction:** The profile by which a service provider discovers which
232 identity providers a Principal may be using.

233 3.1 Common Requirements

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

- 236 1 All HTTP requests and responses MUST be drawn from either HTTP 1.1 (see [RFC2616]) or
237 HTTP 1.0 (see [RFC1945]). When an HTTP redirect is specified, the HTTP response MUST
238 have a status code of "302." According to HTTP 1.1 and HTTP 1.0, the use of status code 302 is
239 recommended to indicate "the requested resource resides temporarily under a different URI."
240 The response may also include additional headers and an optional message.
- 241 2 When `https` is specified as the `<scheme>` for a URL, the HTTP connection MUST be made
242 over either SSL 3.0 (see [SSLv3]) or TLS 1.0 (see [RFC2246]) or any subsequent protocols that
243 are backwards compatible with SSL 3.0 and/or TLS 1.0. Other security protocols MAY be used
244 as long as they implement equivalent security measures.
- 245 3 Service providers MUST authenticate identity providers using identity provider server-side
246 X.509 certificates.
- 247 4 Identity providers MAY authenticate service providers using service provider client-side X.509
248 certificates.
- 249 5 The authenticated identity of an identity provider MUST be securely available to a Principal
250 before the Principal presents his/her personal authentication data to that identity provider.
- 251 6 `<lib:AuthnRequest>` messages SHOULD be digitally signed and verified. All
252 `<lib:Assertion>` elements MUST be digitally signed and verified. For signing and
253 verification of `<lib:AuthnRequest>` and `<lib:Assertion>` elements, identity providers
254 and service providers SHOULD use certificates and private keys that are distinct from the
255 certificates and private keys applied for SSL or TLS channel protection. Certificates and private
256 keys MUST be suitable for long-term signatures.
- 257 7 In transactions between service providers and identity providers, requests MUST be protected
258 against replay, and received responses MUST be checked for correct correspondence with issued
259 requests. (Note: Other steps may intervene between the issuance of a request and its eventual
260 response within a multistep transaction involving redirections.) Additionally, time-based
261 assurance of freshness MAY be provided.
- 262 8 Each service provider within a circle of trust MUST be configured to enable identification of the
263 identity providers whose authentications it will accept, and each identity provider MUST be

264 configured to enable identification of the service providers it intends to serve. (Note: The format
265 of this configuration is a local matter and could, for example, be represented as lists of names or
266 as sets of X.509 certificates of other circle of trust members).

267 9 Circle of trust bilateral agreements on selecting certificate authorities, obtaining X.509
268 credentials, establishing and managing trusted public keys, and tracking lifecycles of
269 corresponding credentials are assumed and not in scope for this specification.

270 10 The `<scheme>` of the URL for SOAP endpoints MUST be `https`.

271 11 All SOAP message exchanges MUST adhere to the SOAP protocol binding for Liberty (see
272 2.1).

273 12 In all SOAP message exchanges, the SOAP receiver MUST authenticate the SOAP Sender.

274 **3.1.1 User Agent**

275 A user agent, unless otherwise noted in the specific profile, MUST support the following features to
276 be interoperable with the protocols in [[LibertyProtSchema](#)] and Liberty profiles in this document:

- 277 • HTTP 1.0 (see [[RFC1945](#)]) or HTTP 1.1 (see [[RFC2616](#)]).
- 278 • SSL 3.0 (see [[SSLv3](#)]) or TLS 1.0 (see [[RFC2246](#)]) or any subsequent protocols that are
279 backwards compatible with SSL 3.0 and/or TLS 1.0 either directly or via a proxy (for
280 example, a WAP gateway).
- 281 • Minimum URL length of 256 bytes.

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

285 **3.1.2 Formatting and Encoding of Protocol Messages**

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

288 **3.1.2.1 Encoding URL-embedded Messages**

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

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

- 293 • The `<query>` component parameter name MUST be the Liberty XML protocol message
294 element or attribute name.
- 295 • The `<query>` component parameter value MUST be the value of the Liberty XML protocol
296 message element or attribute value.
- 297 • When the original message element has multiple values, the value of the `<query>`
298 component parameter MUST be a space-delimited list.
- 299 • Some of the referenced protocol message elements and attributes are optional. If an optional
300 element or attribute does not appear in the original Liberty XML protocol message, then the
301 corresponding data item MUST be omitted from the URL encoded message.

- 302 • URLs appearing in the URL-encoded message SHOULD NOT exceed 80 bytes in length
303 (including %-escaping overhead). Likewise, the <lib:RelayState> data value SHOULD
304 NOT exceed 80 bytes in length.

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

- 308 • Include the signature algorithm identifier as a new <query> component parameter named
309 sigAlg, but omitting the signature.
- 310 • Sign the string containing the URL-encoded message. The string to be signed MUST include
311 only the <query> part of the URL (that is, everything after ? and before &Signature=).
312 Any required URL-escaping MUST be done before signing.
- 313 • Encode the signature using base64 (see [[RFC2045](#)]).
- 314 • Add the base64-encoded signature to the encoded message as a new data item named
315 Signature.

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

317 The following signature algorithms and their identifiers MUST be supported:

- 318 • DSAwithSHA1 — <http://www.w3.org/2000/09/xmlsig#dsa-sha1>
- 319 • RSAwithSHA1 — <http://www.w3.org/2000/09/xmlsig#rsa-sha1>

320 3.1.2.1.1 Size Limitations

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

328 3.1.2.1.2 URL-encoded <lib:AuthnRequest>

329 The original <lib:AuthnRequest> message:

```
330 <lib:AuthnRequest
331   RequestID=" [RequestID] "
332   MajorVersion=" [MajorVersion] "
333   MinorVersion=" [MinorVersion] "
334   IssueInstant=" [IssueInstant] ">
335   <samlp:RespondWith> lib:AuthnResponse </samlp:RespondWith>
336   <lib:ProviderID> [ProviderID] </ lib:ProviderID>
337   <lib:ForceAuthn> [ForceAuthn] </lib:ForceAuthn>
338   <lib:IsPassive> [IsPassive] </lib:IsPassive>
339   <lib:Federate> [Federate] </lib:Federate>
340   <lib:ProtocolProfile> [ProtocolProfile] </lib:ProtocolProfile>
341   <lib:AuthnContext>
342     <lib:authnContextMinimumClassRef>
343       [AuthnContextMinimumClassRef]
344     </lib:authnContextMinimumClassRef>
345     <lib:authnContextStatementRef> [AuthnContextStatementRef] </lib:authnContextStatementRef>
346     <lib:AuthnContextClassRef> [AuthnContextClassRef] </lib:AuthnContextClassRef>
347   </lib:AuthnContext>
348   <lib:RelayState> [RelayState] </lib:RelayState>
349 </lib:AuthnRequest>
```

350

- 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%2FV9aGJ5yw0HL0AcjqQF&MajorVersion=1&MinorVersion=0&IssueInstant=2002-0515T00%3A58%3A19&ProviderID=http%3A%2F%2Fsp.example.com%2Fliberty%2F&ForceAuthn=true&IsPassive=false&Federate=true&ProtocolProfile=http%3A%2F%2Fprojectliberty.org%2Fprofiles%2Fbrws-post&AuthnContextClassRef=http%3A%2F%2Fprojectliberty.org%2Fauthnctx%2Fprofiles%2Fpassword-over-HTTP&RelayState=03mhakSms5tMQ0WRDCEzpf7BNcywZa75FwIcsSEpVbkoFxaQHCuNnc5yChIdDlWc7JBV9Xbw3avRBK7VFsPl2X&SigAlg=http%3A%2F%2Fwww.w3.org%2F2000%2F09%2Fxmldsig%23rsa-sha1&Signature=EoD8bNr2jEOe%2Fumon6oU%2FZGIIF7gbJae4MLUUMrD%2BPP7P8Yf3gfdZG2qPjDnAJkzVHGf08W8DzpQ%0D%0AsDTTd5VP9MLPcvxbFQoF0CJjmvL26cPsuc54q7ourcH0jJ%2F2UkDq4DALYlZ5kPIg%2BtrykgLz0U%2BS%0D%0ANqpNHkj6W3YkGv7RBs%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](#)].

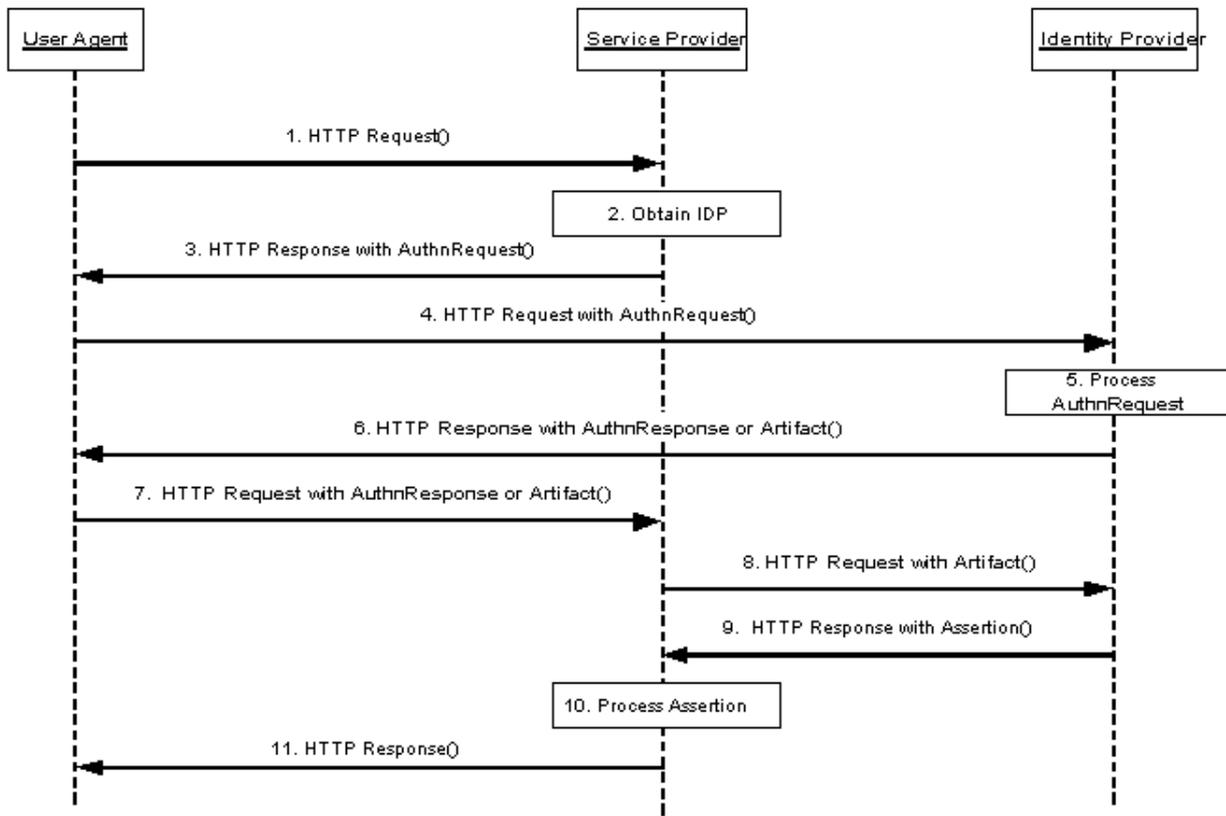
418 The single sign-on profiles make use of the following metadata elements, as defined in
419 [[LibertyProtSchema](#)].

- 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.



438

439

Figure 1: Basic single sign-on profile

440 **Step 1: HTTP Request**

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

445 It is RECOMMENDED that the HTTP Request URI contain a <query> component at its end
 446 where

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

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

451

452 It is RECOMMENDED that the HTTP request be made over either SSL 3.0 (see [SSLv3]) or TLS
 453 1.0 (see [RFC2246]) to maintain confidentiality and message integrity in step 1.

454 **Step 2: Obtain Identity Provider**

455 In step 2, the service provider obtains the address of the appropriate identity provider to redirect the
 456 user agent to in step 3. The means by which the identity provider address is obtained is

457 implementation-dependent and up to the service provider. The service provider MAY use the Liberty
458 identity provider introduction profile in this step.

459 **Step 3: HTTP Response with <AuthnRequest>**

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

462 The redirection MUST adhere to the following rules:

- 463 • The Location HTTP header MUST be set to the identity provider's single sign-on service
464 URL.
- 465 • The identity provider's single sign-on service URL MUST specify `https` as the URL
466 scheme; if another scheme is specified, the service provider MUST NOT redirect to the
467 identity provider.

468 Note: Future protocols may be adopted and enabled to work within this framework.
469 Therefore, implementers are encouraged to not hardcode a reliance on `https`.

- 470 • The Location HTTP header MUST include a `<query>` component containing the
471 `<lib:AuthnRequest>` protocol message as defined in [[LibertyProtSchema](#)] with
472 formatting as specified in 3.1.2.

473 Note: The `<lib:RelayState>` element of the `<lib:AuthnRequest>` message
474 can be used by the service provider to help maintain state information during the
475 single sign-on and federation process. For example, the originally requested resource
476 (that is, LRURL in step 1) could be stored as the value for the `<lib:RelayState>`
477 element, which would then be returned to the service provider in the
478 `<lib:AuthnResponse>` in step 7. The service provider could then use this
479 information to know how to formulate the HTTP response to the user agent in
480 step 11.

481 The HTTP response MUST take the following form:

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

486 where

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

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

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

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

492 **Step 4: HTTP Request with <AuthnRequest>**

493 In step 4, the user agent accesses the identity provider's single sign-on service URL with the
494 `<lib:AuthnRequest>` information attached to the URL fulfilling the redirect request.

495 **Step 5: Processing <AuthnRequest>**

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

498 If the Principal has not yet been authenticated with the identity provider, this point is when
499 authentication at the identity provider could occur. However, whether the identity provider should
500 authenticate the Principal is dependent upon the `<lib:AuthnRequest>` message content.

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

503 **Step 6: HTTP Response with `<AuthnResponse>` or Artifact**

504 In step 6, the identity provider MUST respond to the user agent with a `<lib:AuthnResponse>`, a
505 SAML artifact, or an error.

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

507 **Step 7: HTTP Request with `<AuthnResponse>` or Artifact**

508 In step 7, the user agent accesses the assertion consumer service URL at the service provider with a
509 `<lib:AuthnResponse>`, a SAML artifact, or an error.

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

511 **Step 8: HTTP Request with Artifact**

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

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

515 The service provider MUST send a `<samlp:Request>` SOAP message to the identity provider's
516 SOAP endpoint, requesting the assertion by supplying the SAML assertion artifact in the
517 `<samlp:AssertionArtifact>` element as specified in [\[SAMLBind\]](#).

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

519 **Step 9: HTTP Response with Assertion**

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

521 In this step if the identity provider is able to find or construct the requested assertion, it responds
522 with a `<samlp:Response>` SOAP message with the requested `<lib:Assertion>`. Otherwise, it
523 returns an appropriate error code, as defined within the "SOAP binding for SAML" (see
524 [\[SAMLBind\]](#)).

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

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

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

536 The <saml:ConfirmationMethod> element of the assertion MUST be set to the value specified
537 in [[SAMLCore](#)] for “SAML Artifact,” and the <saml:SubjectConfirmationData> element
538 MUST be present with its value being the SAML artifact supplied to obtain the assertion.

539 **Step 10: Process Assertion**

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

544 The service provider processing of the assertion MUST adhere to the rules defined in [[SAMLCore](#)]
545 for things such as assertion <saml:conditions> and <saml:advice>.

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

550 **Step 11: HTTP Response**

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

553 **3.2.2 Liberty Browser Artifact Profile**

554 The Liberty browser artifact profile relies on a reference to the needed assertion traveling in a SAML
555 artifact, which the service provider must dereference from the identity provider to determine whether
556 the Principal is authenticated. This profile is an adaptation of the “Browser/artifact profile” for
557 SAML as documented in [[SAMLBind](#)]. See Figure 2.

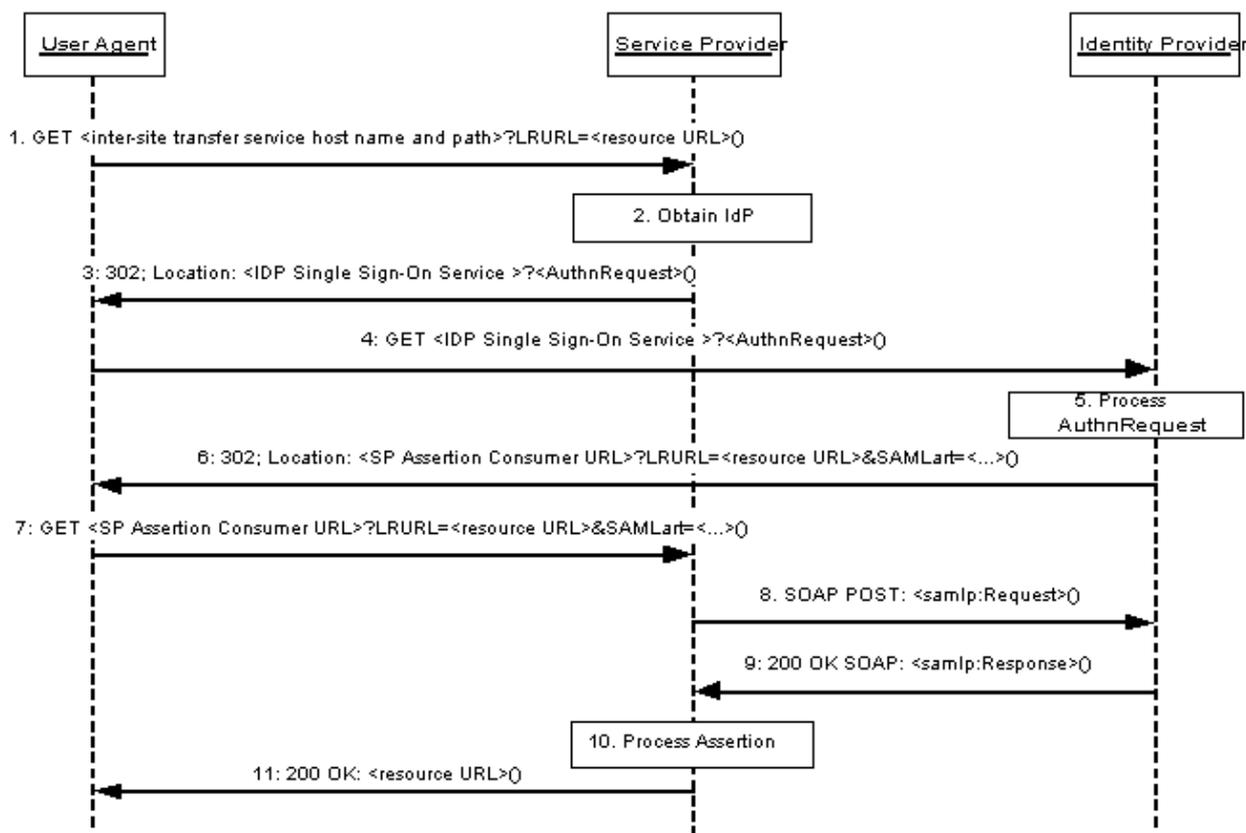
558 The following URI-based identifier MUST be used when referencing this specific profile (for
559 example, <lib:ProtocolProfile> element of the <lib:AuthnRequest> message):

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

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

564 **3.2.2.1 Interactions**

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



566

567

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

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

570 When implementing this profile, all processing rules defined in 3.2.1 for the single sign-on profiles
 571 MUST be followed. Additionally, the following rules MUST be observed as they relate to steps 6
 572 and 7:

573 **Step 6: Redirecting to the Service Provider**

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

576 The redirection MUST adhere to the following rules:

- 577 • The Location HTTP header MUST be set to the service provider’s assertion consumer
 578 service URL, the value of which was determined based upon the <lib:ProviderID>
 579 element of the <lib:AuthnRequest> message.
- 580 • The service provider’s assertion consumer service URL MUST specify https as the URL
 581 scheme; if another scheme is specified, the identity provider MUST NOT redirect to the
 582 service provider.
- 583 • The Location HTTP header MUST include a <query> parameter SAMLart, the value of
 584 which is the SAML artifact on success or empty on failure. Additionally, if the
 585 <lib:AuthnRequest> processed in step 5 included a value for the <lib:RelayState>

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

588 The HTTP response MUST take the following form:

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

593 where

594 <Service Provider assertion consumer service URL>

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

597 <query>= ...SAMLart=<SAML Artifact>...LRURL=<resource URL> ...

598 At least one SAML artifact MUST be included in the <query> component. A single LRURL
599 MUST be included if a value for <RelayState> was provided in the <lib:AuthnRequest>.
600 If more than one SAML artifact is included the <query> component, all artifacts MUST have
601 the same IdentityProviderID.

602 Step 7: Accessing the Assertion Consumer Service

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

605 3.2.2.2 Artifact Format

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

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

609

610 The notation B64 (TypeCode RemainingArtifact) stands for the application of the base64
611 transformation to the catenation of the TypeCode and RemainingArtifact. This profile defines
612 an artifact type of type code 0x0003, which is REQUIRED (mandatory to implement) for any
613 implementation of the Liberty browser artifact profile. This artifact type is defined as follows:

```
614 TypeCode := 0x0003  
615 RemainingArtifact := IdentityProviderID AssertionHandle  
616 IdentityProviderID := 20-byte_sequence  
617 AssertionHandle := 20-byte_sequence
```

618

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

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

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

- 632 • Each identity provider selects a single identification URL. The domain name used within this
- 633 URL is registered with an appropriate authority and administered by the identity provider.
- 634 • The identity provider constructs the IdentityProviderID component of the artifact by
- 635 taking the SHA-1 hash of the identification URL.
- 636 • The identity provider SHOULD employ best effort to ensure the uniqueness of the generated
- 637 IdentityProviderID.

638 The AssertionHandle value is constructed from a cryptographically strong random or pseudo-
 639 random number sequence (see [RFC1750]) generated by the identity provider. The sequence consists
 640 of values of at least eight bytes in size. These values should be padded to a total length of 20 bytes.

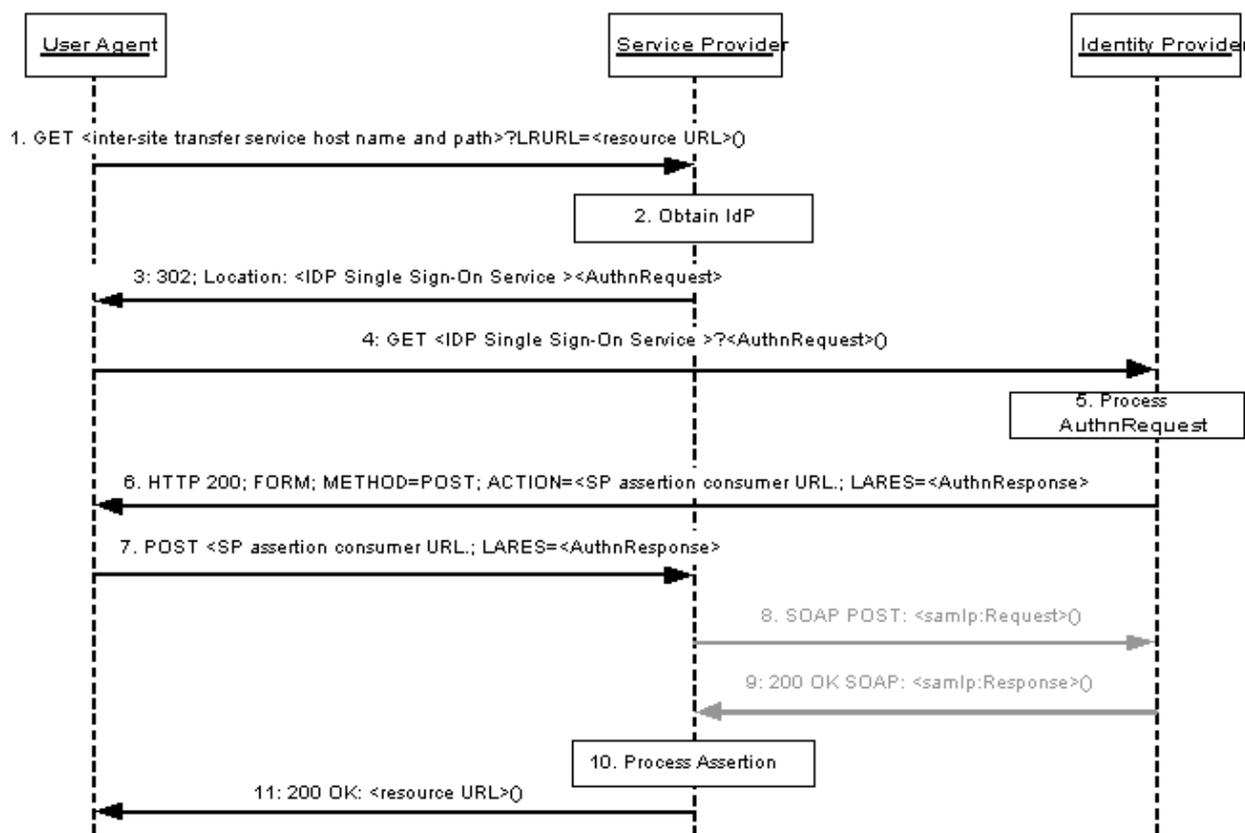
641 3.2.3 Liberty Browser POST Profile

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

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

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

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



651

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.

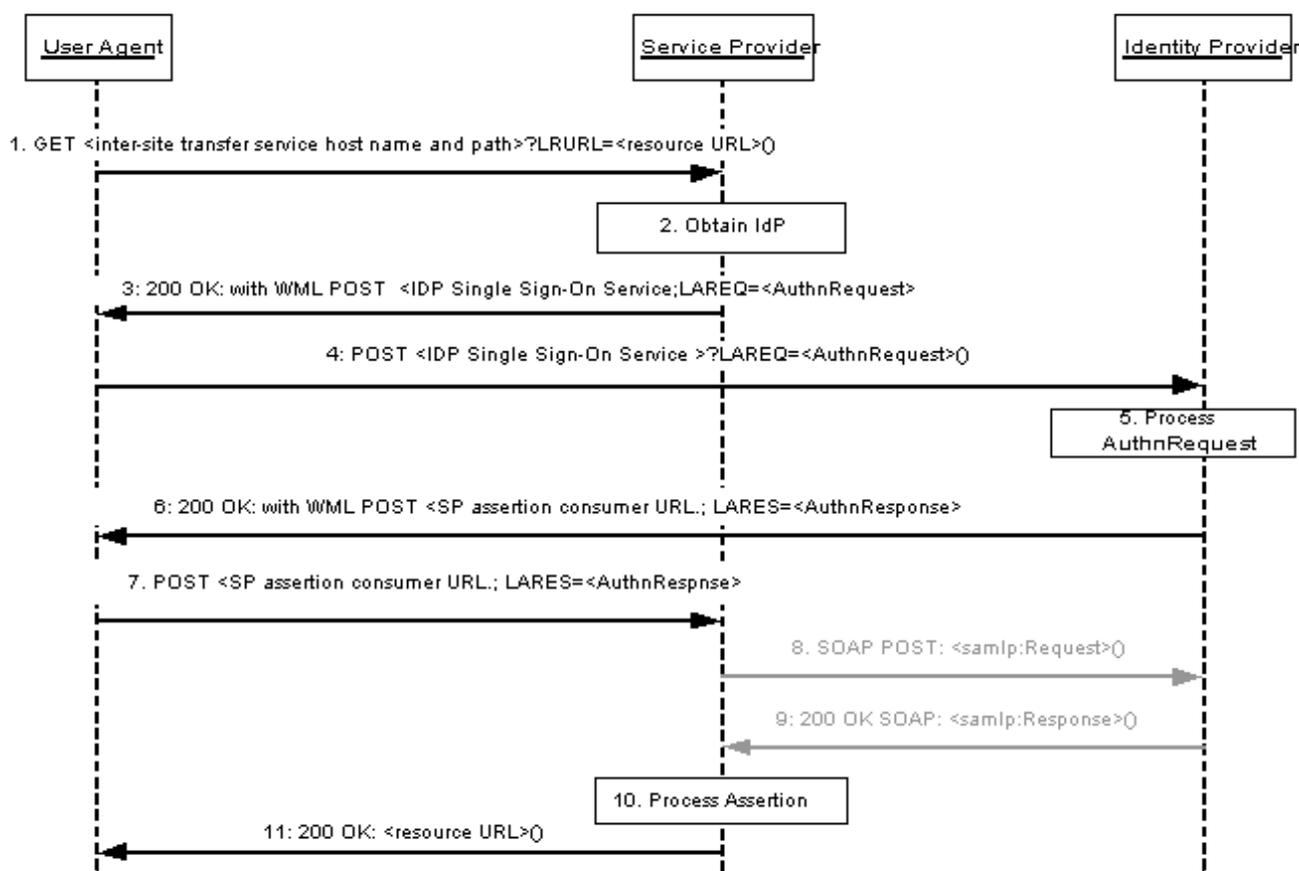
679 The following URI-based identifier MUST be used when referencing this specific profile (for
680 example, <lib:ProtocolProfile> element of the <lib:AuthnRequest> message)

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.



691

692

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

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

695 When implementing this profile, all processing rules defined in 3.2.1 for single sign-on profiles
 696 MUST be followed with the exception that steps 8 and 9 MUST be omitted. Additionally, the
 697 following rules MUST be observed as they relate to steps 3, 4, 6, and 7:

698 **Step 3: HTTP Response with <AuthnRequest>**

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

701 The form contents MUST contain the field LAREQ with the value of the <lib:AuthnRequest>
 702 protocol message as defined in [LibertyProtSchema]. The <lib:AuthnRequest> MUST be
 703 encoded by applying a base64 transformation (refer to [RFC2045]) to the <lib:AuthnRequest>
 704 and all its elements. The identity provider's single sign-on service URL used as the target of the form
 705 POST MUST specify https as the URL scheme; if another scheme is specified, the service provider
 706 MUST NOT issue the POST of the <lib:AuthnRequest> to the identity provider.

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

```
711 ...
712 <wml>
713 <card id="redirect" title="Log In">
714   <onenterforward>
715     <go method="post" href="<Identity Provider Single Sign-On service URL>" >
716     <postfield name="LAREQ" Value="(<lib:AuthnRequest>)" />
717   </go>
718   </onenterforward>
719   <onenterbackward>
720     <prev/>
721   </onenterbackward>
722   <p>
723     Contacting IdP. Please wait...
724   </p>
725 ...
726 </card>
727 ...
728 </wml>
```

729

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

735 **Step 4: HTTP Request with `<AuthnRequest>`**

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

738 **Step 6: HTTP Response with `<AuthnResponse>`**

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

741 The form MUST be constructed so that it requests a POST to the service provider's assertion
742 consumer service URL with the form contents that contain the field `LARES` with the value being the
743 `<lib:AuthnResponse>` protocol message as defined in [[LibertyProtSchema](#)]. The
744 `<lib:AuthnResponse>` MUST be encoded by applying a base64 transformation (refer to
745 [[RFC2045](#)]) to the `<lib:AuthnResponse>` and all its elements. Multiple SAML assertions MAY
746 be included in the response. The identity provider MUST digitally sign the assertions included in the
747 response. The service provider's assertion consumer service URL used as the target of the form
748 POST MUST specify `https` as the URL scheme; if another scheme is specified, it MUST be treated
749 as an error by the identity provider.

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

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

753 **Step 7: HTTP POST with `<AuthnResponse>`**

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

756 3.2.5 Liberty-Enabled Client and Proxy Profile

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

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

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

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

769 All LECPs, in addition to meeting the common requirements for profiles in 3.1, MUST indicate that
770 it is a LECP by including a Liberty-Enabled header or entry in the value of the HTTP User-Agent
771 header for each HTTP request they make. The preferred method is the Liberty-Enabled header. The
772 formats of the Liberty-Enabled header and User-Agent header entry are defined 3.2.5.1.

773 3.2.5.1 Liberty-Enabled Indications

774 A LECP SHOULD add the Liberty-Enabled header to each HTTP request. The Liberty-Enabled
775 header MUST be named Liberty-Enabled and be defined as using Augmented BNF as specified
776 in section 2 of [[RFC 2616](#)].

```
777 Liberty-LECP = "Liberty-LECP" ":" LIB_Version ["," 1#Extension]
778 LIB_Version = "LIBV" "=" 1*absoluteURI
779 ; any spaces or commas in the absoluteURI MUST be escaped as
780 defined in section 2.4 of [RFC 2396]
781 Extension = ExtName "=" ExtValue
782 ExtName = (["." host] | <any field-value but ".", ",", or "=">) <any
783 field-value but "=" or ",", ">
784 ExtValue = <any field-value but ",", ">
785
```

786 The comment, field-value, and product productions are defined in [[RFC 2616](#)]. LIB_Version
787 identifies the versions of the Liberty specifications that are supported by this LECP. Each version is
788 identified by a URI. Service providers or identity providers receiving a Liberty-Enabled header
789 MUST ignore any URIs listed in the LIB_Version production that they do not recognize. All
790 LECPs compliant with this specification MUST send out, at minimum, the URI
791 `http://projectliberty.org/specs/v1` as a value in the LIB_Version production. The
792 ordering of the URIs in the LIB_Version header is meaningful; therefore, service providers and
793 identity providers are encouraged to use the first version in the list that they support. Supported
794 Liberty versions are not negotiated between the LECP and the service provider. The LECP simply
795 advertises what version it does support, and the service provider MUST return the response for the
796 corresponding version as defined in step 3 below.

797 Optional extensions MAY be added to the Liberty-Enabled header to indicate new information. The
798 value of the ExtName production MUST use the “host” “;” prefixed form if the new extension
799 name has not been standardized and registered with Liberty or its designated registration authorities.
800 The value of the host production MUST be an IP or DNS address that is owned by the issuer of the

801 new name. By using the DNS/IP prefix, namespace collisions can be effectively prevented without
802 having to introduce yet another centralized registration agency.

803 LECPs MAY include the Liberty-Agent header in their requests. This header provides information
804 about the software implementing the LECP functionality and is similar to the User-Agent and Server
805 headers in HTTP.

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

807

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

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

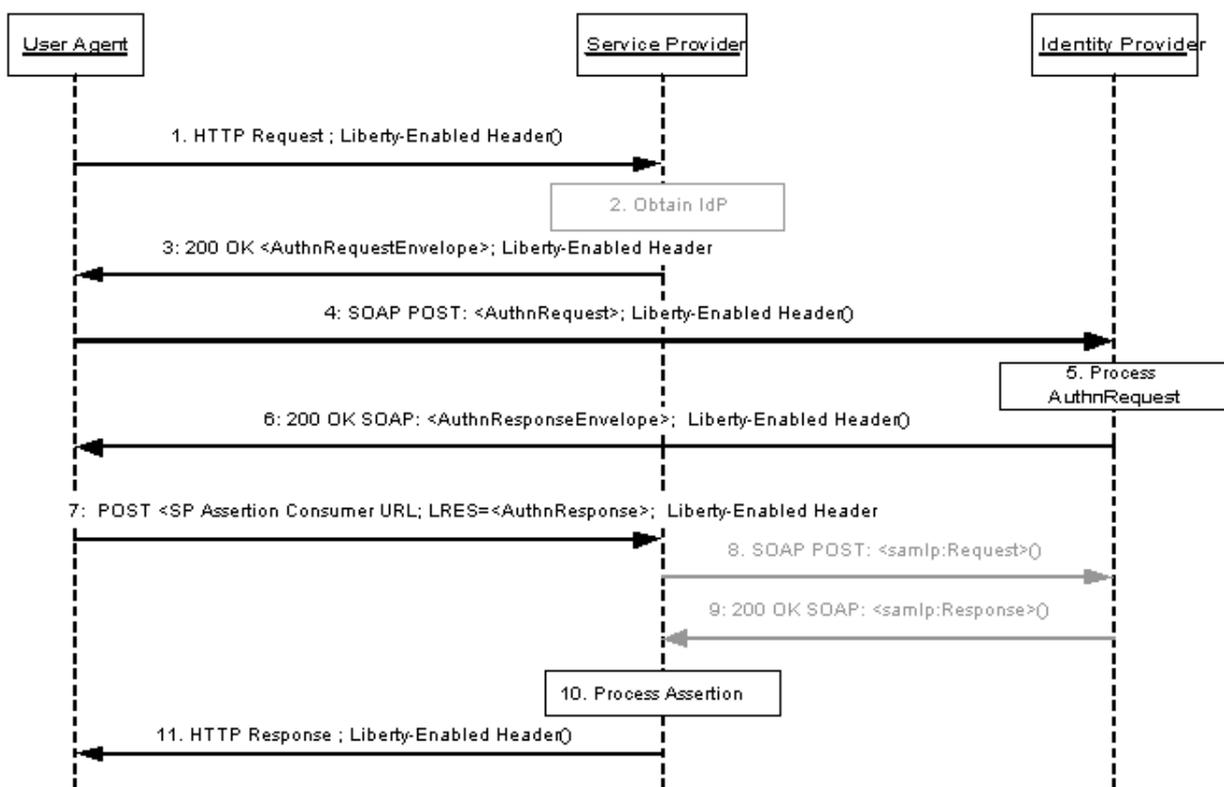
820 A LECP MAY add a Liberty-Enabled entry in the HTTP User-Agent request header. The HTTP
821 User-Agent header is specified in [[RFC2616](#)]. A LECP MAY include in the value of this header the
822 Liberty-LECP string as defined above for the Liberty-Enabled header.

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

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

829 **3.2.5.2 Interactions**

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



831

832

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

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

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

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

844 **Step 1: Accessing the Service Provider**

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

847 The HTTP request MUST contain only one Liberty-Enabled header. Hence if a proxy receives a
 848 HTTP request that contains a Liberty-Enabled header, it MUST NOT add another Liberty-Enabled
 849 header. However, a proxy MAY replace the Liberty-Enabled header. A proxy that replaces or adds a
 850 Liberty-Enabled header MUST process `<lib:AuthnRequest>` messages as defined in steps 3 and
 851 4 as well as `<lib:AuthnResponse>` messages as specified in steps 6 and 7.

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

854 **Step 3: HTTP Response with <AuthnRequest>**

855 In step 3, the service provider’s intersite transfer service issues an HTTP 200 OK response to the
856 user agent. The response MUST contain a single <lib:AuthnRequestEnvelope> with content as
857 defined in [[LibertyProtSchema](#)]. The <lib:AuthnRequestEnvelope> MUST be encoded by
858 applying a base64 transformation (refer to [[RFC2045](#)]) to the <lib:AuthnRequestEnvelope>
859 and all its elements.

860 If a service provider receives a Liberty-Enabled header, or a User-Agent header with the Liberty-
861 Enabled entry, the service provider MUST respond according to the Liberty-enabled client and proxy
862 profile and include a Liberty_Enabled header in its response. Hence service providers MUST support
863 the Liberty-enabled client and proxy profile.

864 The processing rules and default values for the Liberty-Enabled indications are as defined in 3.2.5.1.
865 The service provider MAY advertise any Liberty version supported in this header, not only the
866 version used for the specific response.

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

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

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

876 The service provider MUST provide its assertion consumer service URL by including the
877 <lib:AssertionConsumerServiceURL> element in the <lib:AuthnRequestEnvelope>.

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

```
879 <?xml version="1.0" ?>  
880 <lib:AuthnRequestEnvelope xmlns:lib="http://projectliberty.org/schemas/core/2002/08/">  
881 <lib:AssertionConsumerServiceURL>  
882 https://service-provider.com/LibertyLogin  
883 </lib:AssertionConsumerServiceURL>  
884 <lib:IDPList >  
885 . . . IdP list goes here . . .  
886 </lib:IDPList>  
887 <lib:AuthnRequest >  
888 . . . AuthnRequest goes here . . .  
889 </lib:AuthnRequest>  
890 </lib:AuthnRequestEnvelope>
```

891

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

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

899 **Step 4: HTTP Request with <AuthnRequest>**

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

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

909 If the LECP discovers a syntax error due to the service provider or cannot proceed any further for
910 other reasons (for example, cannot resolve identity provider, cannot reach the identity provider, etc),
911 the LECP MUST return to the service provider a <lib:AuthnResponse> with a
912 <samlp:Status> indicating the desired error element as defined in [[LibertyProtSchema](#)]. The
913 <lib:AuthnResponse> containing the error status MUST be sent using a POST to the service
914 provider's assertion consumer service URL obtained from the
915 <lib:AssertionConsumerServiceURL> element of the <lib:AuthnRequestEnvelope>.
916 The POST MUST be a form that contains the field named LARES with its value the
917 <lib:AuthnResponse> protocol message as defined in [[LibertyProtSchema](#)] with formatting as
918 specified in 3.1.2.

919 **Step 6: HTTP Response with <AuthnResponse>**

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

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

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

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

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

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

937 In step 7, the LECP issues an HTTP POST of the <lib:AuthnResponse> that was received in the
938 <lib:AuthnResonseEnvelope> SOAP response in step 6. The <lib:AuthnResponse> MUST
939 be sent using a POST to the service provider's assertion consumer service URL identified by the
940 <lib:AssertionConsumerServiceURL> obtained from the <lib:AuthnRequestEnvelope>
941 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:

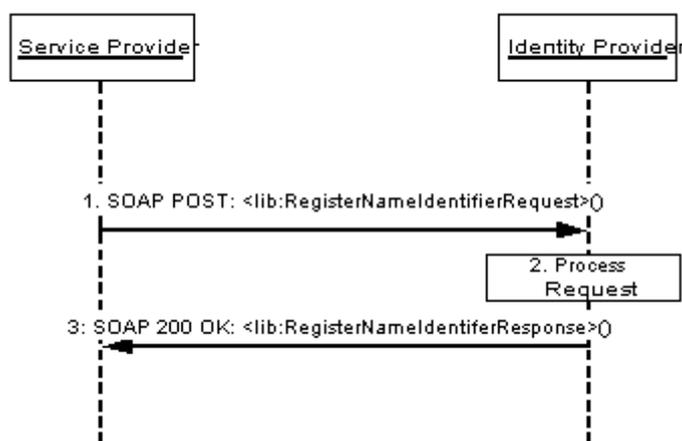
963 URI: <http://projectliberty.org/profiles/rni-soap>

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.



971

972

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

973 **Step 1: Request to Register Name Identifier**

974 In step 1, the service provider sends a `<lib:RegisterNameIdentifierRequest>` protocol
975 message to the identity provider's SOAP endpoint specifying both the
976 `<lib:IDPProviderNameIdentifier>` and the `<lib:SPPProviderNameIdentifier>` as
977 specified in [[LibertyProtSchema](#)].

978 **Step 2: Response to Register Name Identifier**

979 The identity provider, after successfully registering the `<lib:SPPProviderNameIdentifier>`
980 provided by the service provider, MUST respond with a
981 `<lib:RegisterNameIdentifierResponse>` according to the processing rules defined in
982 [[LibertyProtSchema](#)].

983 **3.4 Identity Federation Termination Notification Profiles**

984 The Liberty identity federation termination notification profiles specify how service providers and
985 identity providers are notified of federation termination (also known as defederation). Note: Other
986 means of federation termination are possible, such as federation expiration and termination of
987 business agreements between service providers and identity providers. These means of federation
988 termination are outside the scope of this specification.

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

994 • **Federation Termination Notification Initiated at Identity Provider**

- 995 • **HTTP-Redirect-Based:** Relies on a HTTP 302 redirect to communicate between the
996 identity provider and the service provider.
- 997 • **SOAP/HTTP-Based:** Relies on a SOAP call from the identity provider to the service
998 provider.

- 999 • **Federation Termination Notification Initiated at Service Provider**
- 1000 • **HTTP-Redirect-Based:** Relies on a HTTP 302 redirect to communicate between the
1001 service provider and the identity provider.
- 1002 • **SOAP/HTTP-Based:** Relies on a SOAP call from the service provider to the identity
1003 provider.
- 1004 The interactions and processing rules for the SOAP/HTTP-based and HTTP-redirect-based profiles
1005 are essentially the same regardless of whether federation termination notification was initiated at the
1006 service provider or at the identity provider.
- 1007 The identity federation termination notification profiles make use of the following metadata
1008 elements, as defined in [[LibertyProtSchema](#)]:
- 1009 • `FederationTerminationServiceURL` — The URL at the service provider or identity
1010 provider to which identity federation termination notifications are sent. It is documented in
1011 these profiles as “federation termination service URL.”
- 1012 • `FederationTerminationServiceReturnURL` — The URL used by the service provider
1013 or identity provider when redirecting the user agent at the end of the federation termination
1014 notification profile process.
- 1015 • `FederationTerminationNotificationProtocolProfile` — Used by the identity
1016 provider to determine which federation termination notification profile **MUST** be used when
1017 communicating with the service provider.
- 1018 • `SOAPEndPoint` — The SOAP endpoint location at the service provider or identity provider
1019 to which Liberty SOAP messages are sent.

1020 **3.4.1 Federation Termination Notification Initiated at Identity Provider**

1021 The profiles in 3.4.1.1 and 3.4.1.2 are specific to identity federation termination when initiated at the
1022 identity provider. Effectively, when using these profiles, the identity provider is stating to the service
1023 provider that it will no longer provide the Principal’s identity information to the service provider and
1024 that the identity provider will no longer respond to any requests by the service provider on behalf of
1025 the Principal.

1026 **3.4.1.1 HTTP-Redirect-Based Profile**

1027 The HTTP-redirect-based profile relies on using HTTP 302 redirect to communicate federation
1028 termination notification messages from the identity provider to the service provider. See Figure 7.

1029 The following URI-based identifier **MUST** be used when referencing this specific profile:

1030 URI: `http://projectliberty.org/profiles/fedterm-idp-http`

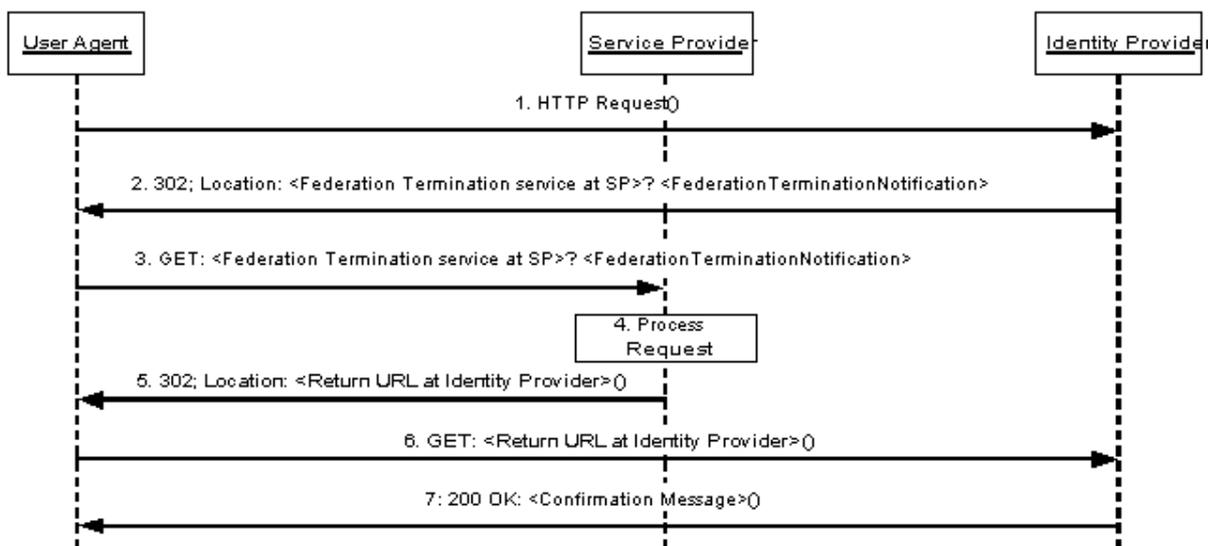


Figure 7: HTTP-redirect-based profile for federation termination

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.

1055 Note: Additionally, the URL-encoded
1056 `<lib:FederationTerminationNotification>` message MAY also include a
1057 parameter named RELAYSTATE with a value set to the URL (and/or other
1058 information) to be used by the identity provider in the HTTP response to the user
1059 agent at the completion of federation termination in step 7.

1060 The HTTP response MUST take the following form:

```
1061 <HTTP-Version> 302 <Reason Phrase>  
1062 <other headers>  
1063 Location : https://<Service Provider Federation Termination service URL>?<query>  
1064 <other HTTP 1.0 or 1.1 components>
```

1065 where

1066 `<Service Provider Federation Termination service URL>`

1067 This element provides the host name, port number, and path components of the federation
1068 termination service URL at the service provider.

1069 `<query>= ...<URL-encoded FederationTerminationNotification>...`

1070 The `<query>` component MUST contain a single terminate federation request.

1071 **Step 3: Accessing the Service Provider Federation Termination Service**

1072 In step 3, the user agent accesses the service provider's federation termination service URL with the
1073 `<lib:FederationTerminationNotification>` information attached to the URL fulfilling the
1074 redirect request.

1075 **Step 4: Processing the Notification**

1076 In step 4, the service provider MUST process the
1077 `<lib:FederationTerminationNotification>` according to the rules defined in
1078 [[LibertyProtSchema](#)].

1079 The service provider MAY remove any locally stored references to the name identifier it received
1080 from the identity provider in the `<lib:FederationTerminationNotification>`.

1081 **Step 5: Redirecting to the Identity Provider Return URL**

1082 In step 5, the service provider's federation termination service responds and redirects the user agent
1083 back to identity provider using a return URL location specified in the
1084 `FederationTerminationServiceReturnURL` metadata element. If the URL-encoded
1085 `<lib:FederationTerminationNotification>` message received in step 3 contains a
1086 parameter named RELAYSTATE, then the service provider MUST include a `<query>` component
1087 containing the same RELAYSTATE parameter and its value in its response to the identity provider.

1088 No success or failure message should be conveyed in this HTTP redirect. The sole purpose of this
1089 redirect is to return the user agent to the identity provider where the federation termination process
1090 began.

1091 The HTTP response MUST take the following form:

```
1092 <HTTP-Version> 302 <Reason Phrase>  
1093 <other headers>  
1094 Location : https://<Identity Provider Return URL >?<query>  
1095 <other HTTP 1.0 or 1.1 components>
```

1096 where

1097 `<Identity Provider Return URL>`

1098 This element provides the host name, port number, and path components of the return URL at the

1099 identity provider.

1100 <query>= ...RELAYSTATE=<...>

1101 The <query> component MUST contain the identical RELAYSTATE parameter and its value that
1102 was received in the URL-encoded federation termination message obtained in step 3. If no
1103 RELAYSTATE parameter was provided in the step 3 message, then a RELAYSTATE parameter
1104 MUST NOT be specified in the <query> component.

1105 Step 6: Accessing the Identity Provider Return URL

1106 In step 6, the user agent accesses the identity provider's return URL location fulfilling the redirect
1107 request.

1108 Step 7: Confirmation

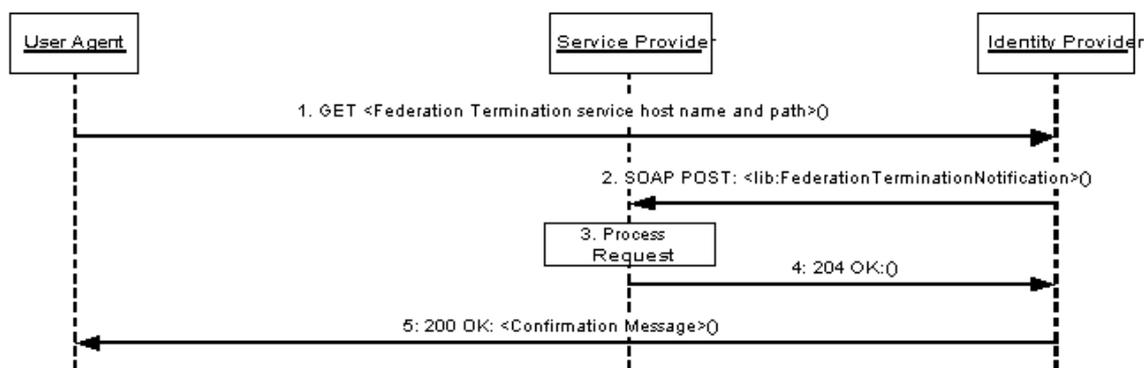
1109 In step 7, the user agent is sent an HTTP response that confirms the requested action of identity
1110 federation termination with the specific service provider.

1111 3.4.1.2 SOAP/HTTP-Based Profile

1112 The SOAP/HTTP-based profile relies on using asynchronous SOAP over HTTP to communicate
1113 federation termination notification messages from the identity provider to the service provider. See
1114 Figure 8.

1115 The following URI-based identifier MUST be used when referencing this specific profile:

1116 URI: <http://projectliberty.org/profiles/fedterm-idp-soap>



1117

1118 **Figure 8: SOAP/HTTP-based profile for federation termination**

1119 This profile description assumes the following preconditions:

- 1120 • The Principal's identity at the service provider is federated with his/her identity at the
1121 identity provider.
- 1122 • The Principal has requested to the identity provider that the federation be terminated.
- 1123 • The Principal has authenticated with the identity provider.

1124 Step 1: Accessing the Federation Termination Service

1125 In step 1, the user agent accesses the identity federation termination service URL at the identity
1126 provider specifying the service provider for with whom identity federation termination should occur.

1127 How the service provider is specified is implementation-dependent and, as such, is out of the scope
1128 of this specification.

1129 **Step 2: Notification of Federation Termination**

1130 In step 2, the identity provider sends an asynchronous SOAP over HTTP notification message to the
1131 service provider's SOAP endpoint. The SOAP message MUST contain exactly one
1132 `<lib:FederationTerminationNotification>` element in the SOAP body and adhere to the
1133 construction rules defined in [[LibertyProtSchema](#)].

1134 If a SOAP fault occurs, the identity provider SHOULD employ best effort to resolve the fault
1135 condition and resend the federation termination notification message to the service provider.

1136 **Step 3: Processing the Notification**

1137 In step 3, the service provider MUST process the
1138 `<lib:FederationTerminationNotification>` according to the rules defined in
1139 [[LibertyProtSchema](#)].

1140 The service provider MAY remove any locally stored references to the name identifier it received
1141 from the identity provider in the `<lib:FederationTerminationNotification>`.

1142 **Step 4: Responding to the Notification**

1143 In step 4, the service provider MUST respond to the
1144 `<lib:FederationTerminationNotification>` with a HTTP 204 OK response.

1145 **Step 5: Confirmation**

1146 In step 5, the user agent is sent an HTTP response that confirms the requested action of identity
1147 federation termination with the specific service provider.

1148 **3.4.2 Federation Termination Notification Initiated at Service Provider**

1149 The profiles in 3.4.2.1 and 3.4.2.2 are specific to identity federation termination notification when
1150 initiated by a Principal at the service provider. Effectively, when using this profile, the service
1151 provider is stating to the identity provider that the Principal has requested that the identity provider
1152 no longer provide the Principal's identity information to the service provider and that service
1153 provider will no longer ask the identity provider to do anything on the behalf of the Principal.

1154 **3.4.2.1 HTTP-Redirect-Based Profile**

1155 The HTTP-redirect-based profile relies on the use of a HTTP 302 redirect to communicate a
1156 federation termination notification message from the service provider to the identity provider. For a
1157 discussion of the interactions and processing steps, refer to 3.4.1.1. When reviewing that profile,
1158 interchange all references to service provider and identity provider in the interaction diagram and
1159 processing steps.

1160 The following URI-based identifier MUST be used when referencing this specific profile:

1161 URI: `http://projectliberty.org/profiles/fedterm-sp-http`

1162 3.4.2.2 SOAP/HTTP-Based Profile

1163 The SOAP/HTTP-based profile relies on using asynchronous SOAP over HTTP to communicate
1164 federation termination notification messages from the service provider to the identity provider. For a
1165 discussion of the interactions and processing steps, refer to 3.4.1.2. When reviewing that profile,
1166 interchange all references to service provider and identity provider in the interaction diagram and
1167 processing steps.

1168 The following URI-based identifier **MUST** be used when referencing this specific profile:

1169 URI: `http://projectliberty.org/profiles/fedterm-sp-soap`

1170 3.5 Single Logout Profiles

1171 The single logout profiles synchronize session logout functionality across all sessions that were
1172 authenticated by a particular identity provider. The single logout can be initiated at either the identity
1173 provider or the service provider. In either case, the identity provider will then communicate a logout
1174 notification to each service provider with which it has established a session for the Principal. The
1175 negotiation of which single logout profile the identity provider uses to communicate with each
1176 service provider is based upon the `SingleLogoutProtocolProfile` provider metadata element
1177 defined in [[LibertyProtSchema](#)].

1178 The available profiles are defined in 3.5.1 and 3.5.2, depending on whether the single logout is
1179 initiated at the identity provider or service provider:

1180 • Single Logout Initiated at Identity Provider

- 1181 • **HTTP-Redirect-Based:** Relies on using HTTP 302 redirects to communicate logout
1182 notifications from an identity provider to the service providers.
- 1183 • **HTTP-GET-Based:** Relies on using HTTP GET requests to communicate logout
1184 notifications from an identity provider to the service providers.
- 1185 • **SOAP/HTTP-Based:** Relies on asynchronous SOAP over HTTP messaging to
1186 communicate a logout notifications from an identity provider to the service providers.

1187 • Single Logout Initiated at Service Provider

- 1188 • **HTTP-Redirect-Based:** Relies on a HTTP 302 redirect to communicate a logout
1189 notification with the identity provider.
- 1190 • **SOAP/HTTP-Based:** Relies on asynchronous SOAP over HTTP messaging to
1191 communicate a logout notification from a service provider to an identity provider.

1192 The single logout profiles make use of the following metadata elements, as defined in
1193 [[LibertyProtSchema](#)].

- 1194 • `SingleLogoutServiceURL` — The URL at the service provider or identity provider to
1195 which single logout notifications are sent. It is described in these profiles as “single logout
1196 service URL.”
- 1197 • `SingleLogoutServiceReturnURL` — The URL used by the service provider when
1198 redirecting the user agent to the identity provider at the end of the single logout profile
1199 process.
- 1200 • `SingleLogoutProtocolProfile` — Used by the identity provider to determine which
1201 single logout notification profile **MUST** be used when communicating with the service
1202 provider.

- 1203 • SOAPEndPoint — The SOAP endpoint location at the service provider or identity provider
1204 to which Liberty SOAP messages are sent.

1205 3.5.1 Single Logout Initiated at Identity Provider

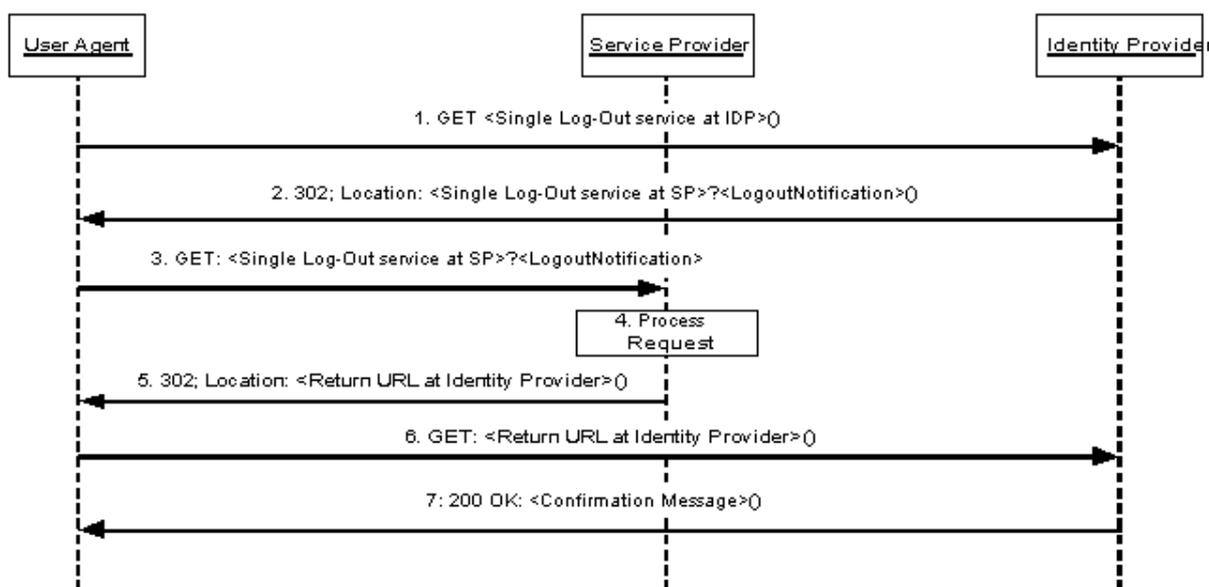
1206 The profiles in 3.5.1.1 through 3.5.1.3 are specific to single logout when initiated by a user agent at
1207 the identity provider.

1208 3.5.1.1 HTTP-Redirect-Based Profile

1209 The HTTP-redirect-based profile uses HTTP 302 redirects to communicate a logout notification to
1210 each service provider for which the identity provider has provided authentication assertions during
1211 the Principal's current session if the service provider indicated a preference to receive logout
1212 notification via the HTTP redirect profile. See Figure 9.

1213 The following URI-based identifier MUST be used when referencing this specific profile:

1214 URI: `http://projectliberty.org/profiles/slo-idp-http`



1215

1216 **Figure 9: HTTP-redirect-based profile for single logout initiated at identity provider**

1217 Note: Steps 2 through 6 may be an iterative process for each service provider that has been issued
1218 authentication assertions during the Principal's current session and has indicated a preference to
1219 receive logout notification via the HTTP redirect profile.

1220 Note: [\[RFC 2616\]](#) indicates a client should detect infinite redirection loops because
1221 such loops generate network traffic for each redirection. This requirement was
1222 introduced because previous versions of the specification recommended a maximum
1223 of five redirections. Content developers should be aware that some clients might
1224 implement such a fixed limitation.

1225 **Step 1: Accessing the Single Logout Service at the Identity Provider**

1226 In step 1, the user agent accesses the single logout service URL at the identity provider indicating
1227 that all service providers for which this identity provider has provided authentication assertions
1228 during the Principal's current session must be notified of session termination.

1229 **Step 2: Redirecting to the Single Logout Service at the Service Provider**

1230 In step 2, the identity provider's single logout service responds and redirects the user agent to the
1231 single logout service URL at each service provider for which the identity provider has provided an
1232 authentication assertion during the Principal's current session with the identity provider.

1233 The redirections MUST adhere to the following rules:

- 1234 • The Location HTTP header MUST be set to the service provider's single logout service
1235 URL.
- 1236 • The service provider's single logout service URL MUST specify https as the URL scheme;
1237 if another scheme is specified, the identity provider MUST NOT redirect to the service
1238 provider.
- 1239 • The Location HTTP header MUST include a <query> component containing the
1240 <lib:LogoutNotification> protocol message as defined in [[LibertyProtSchema](#)] with
1241 formatting as specified in 3.1.2.

1242 Note: Additionally, the URL-encoded <lib:LogoutNotification> message
1243 MAY also include a parameter named RELAYSTATE with a value set to the URL
1244 (and/or other information) to be used by the identity provider in the HTTP response
1245 to the user agent at the completion of federation termination in step 7.

1246 The HTTP response MUST take the following form:

```
1247 <HTTP-Version> 302 <Reason Phrase>  
1248 <other headers>  
1249 Location : https://<Service Provider Single Log-Out service URL>?<query>  
1250 <other HTTP 1.0 or 1.1 components>
```

1251 where

1252 <Service Provider Single Log-Out service URL>

1253 This element provides the host name, port number, and path components of the single logout
1254 service URL at the service provider.

1255 <query>= ...<URL-encoded LogoutNotification>...

1256 The <query> MUST contain a single logout notification request.

1257 **Step 3: Accessing the Service Provider Single Logout Service**

1258 In step 3, the user agent accesses the service provider's single logout service URL with the
1259 <lib:LogoutNotification> information attached to the URL fulfilling the redirect request.

1260 **Step 4: Processing the Notification**

1261 In step 4, the service provider MUST process the <lib:LogoutNotification> according to the
1262 rules defined in [[LibertyProtSchema](#)].

1263 The service provider MUST invalidate the session(s) of the Principal referred to in the name
1264 identifier it received from the identity provider in the <lib:LogoutNotification>.

1265 **Step 5: Redirecting to the Identity Provider Return URL**

1266 In step 5, the service provider's single logout service responds and redirects the user agent back to
1267 the identity provider using the return URL location obtained from the
1268 SingleLogoutServiceReturnURL metadata element. If the URL-encoded
1269 <lib:LogoutNotification> message received in step 3 contains a parameter named
1270 RELAYSTATE, then the service provider MUST include a <query> component containing the same
1271 RELAYSTATE parameter and its value in its response to the identity provider.

1272 No success or failure message should be conveyed in this HTTP redirect. The sole purpose of this
1273 redirect is to return the user agent to the identity provider so that the single logout process may
1274 continue in the same fashion with other service providers.

1275 The HTTP response MUST take the following form:

```
1276 <HTTP-Version> 302 <Reason Phrase>  
1277 <other headers>  
1278 Location : https://<Identity Provider Return URL>?<query>  
1279 <other HTTP 1.0 or 1.1 components>
```

1280 where

1281 <Identity Provider Return URL>

1282 This element provides the host name, port number, and path components of the return URL at the
1283 identity provider.

1284 <query>= ...RELAYSTATE=<...>

1285 The <query> component MUST contain the identical RELAYSTATE parameter and its value that
1286 was received in the URL-encoded logout notification message obtained in step 3. If no
1287 RELAYSTATE parameter was provided in the step 3 message, then a RELAYSTATE parameter
1288 MUST NOT be specified in the <query> component.

1289 **Step 6: Accessing the Identity Provider Return URL**

1290 In step 6, the user agent accesses the identity provider's return URL location fulfilling the redirect
1291 request.

1292 **Step 7: Confirmation**

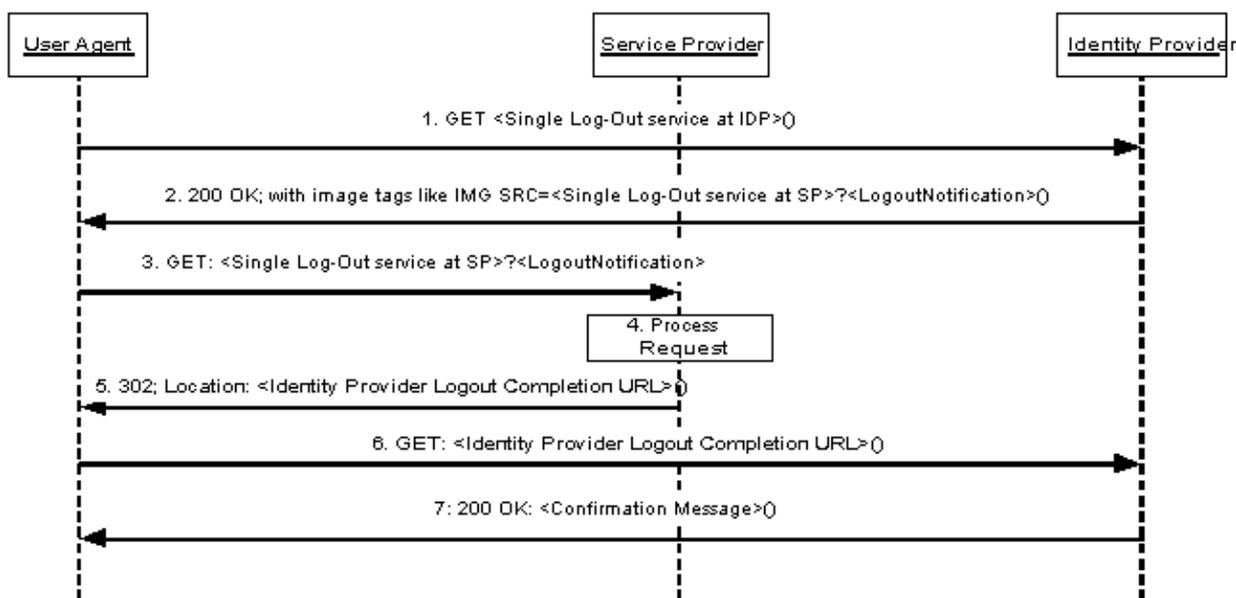
1293 In step 7, the user agent is sent an HTTP response that confirms the requested action of single logout
1294 has completed.

1295 **3.5.1.2 HTTP-GET-Based Profile**

1296 The HTTP-GET-based profile uses HTTP GET requests to communicate logout notifications to each
1297 service provider for which the identity provider has provided authentication during the Principal's
1298 current session if the service provider indicated a preference to receive logout notifications via the
1299 HTTP GET profile. See Figure 10.

1300 The following URI-based identifier MUST be used when referencing this specific profile:

1301 URI: <http://projectliberty.org/profiles/slo-idp-http-get>



1302

1303

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

1304

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.

1305

1306

1307

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

1308

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.

1309

1310

1311

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

1312

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.

1313

1314

1315

1316

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

1317

- 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.

1318

1319

1320

1321

1322

Step 3: Accessing the Service Provider Single Logout Service

1323

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

1324

1325 occur multiple times if the HTTP response includes multiple image tag statements (one for each
1326 service provider that has been issued authentication assertions during the Principal's current session).

1327 **Step 4: Processing the Notification**

1328 In step 4, the service provider MUST process the <lib:LogoutNotification> according to the
1329 rules defined in [[LibertyProtSchema](#)].

1330 The service provider MUST invalidate the session of the Principal referred to in the name identifier it
1331 received from the identity provider in the <lib:LogoutNotification>.

1332 **Step 5: Redirecting to the Identity Provider Logout Completion URL**

1333 In step 5, the service provider's single logout service responds and redirects the image load back to
1334 the identity provider's logout completion URL. This location will typically point to an image that
1335 will be loaded by the user agent to indicate that the logout is complete (for example, a checkmark).

1336 The logout completion URL is obtained from the SingleLogoutServiceReturnURL metadata
1337 element.

1338 The HTTP response MUST take the following form:

```
1339 <HTTP-Version> 302 <Reason Phrase>  
1340 <other headers>  
1341 Location : https://<Identity Provider Logout Completion URL>  
1342 <other HTTP 1.0 or 1.1 components>
```

1343 where

1344 <Identity Provider Logout Completion URL>

1345 This element provides the host name, port number, and path components of the identity provider
1346 logout completion URL at the identity provider.

1347 **Step 6: Accessing the Identity Provider Logout Completion URL**

1348 In step 6, the user agent accesses the identity provider's logout completion URL fulfilling the
1349 redirect request.

1350 **Step 7: Confirmation**

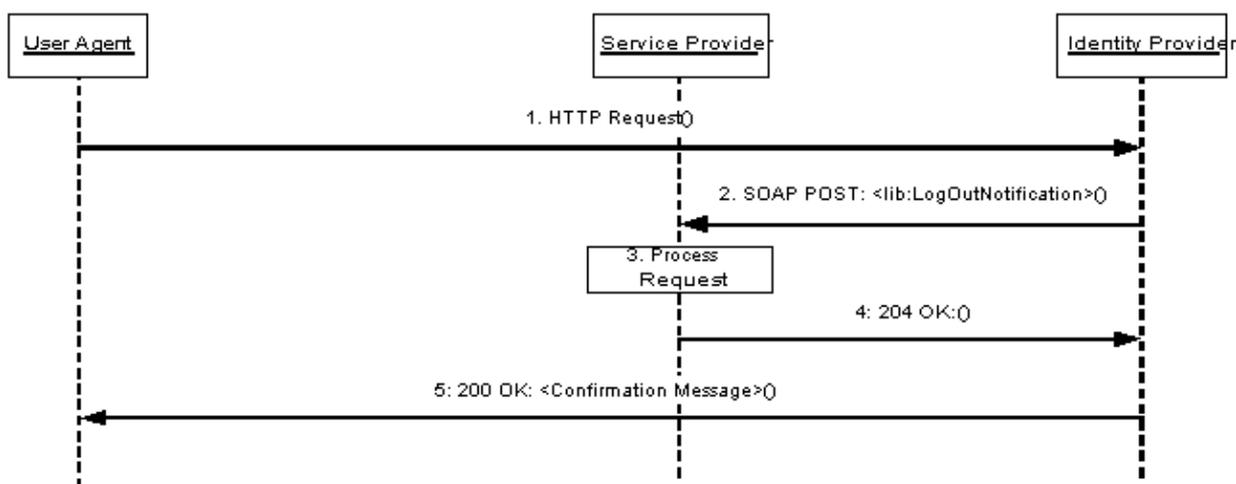
1351 In step 7, the user agent is sent an HTTP response that confirms the requested action of single logout
1352 has completed.

1353 **3.5.1.3 SOAP/HTTP-Based Profile**

1354 The SOAP/HTTP-based profile uses asynchronous SOAP over HTTP messaging to communicate a
1355 logout notification to each service provider for which the identity provider has provided
1356 authentication assertions during the Principal's current session if the service provider indicated a
1357 preference to receive logout notification via the SOAP/HTTP-based profile. See Figure 11.

1358 The following URI-based identifier MUST be used when referencing this specific profile:

1359 URI: <http://projectliberty.org/profiles/slo-idp-soap>



1360

1361

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

1362

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.

1363

1364

1365 **Step 1: Accessing the Single Logout Service**

1366

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

1367

1368 **Step 2: Notification of Logout**

1369

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](#)].

1370

1371

1372

1373

1374

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.

1375

1376 **Step 3: Processing the Notification**

1377

In step 3, the service provider MUST process the <lib:LogoutNotification> according to the rules defined in [[LibertyProtSchema](#)].

1378

1379

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

1380

1381 **Step 4: Responding to the Notification**

1382

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

1383

1384 **Step 5: Confirmation**

1385 In step 5, the user agent is sent an HTTP response that confirms the requested action of single logout
1386 has completed.

1387 **3.5.2 Single Logout Initiated at Service Provider**

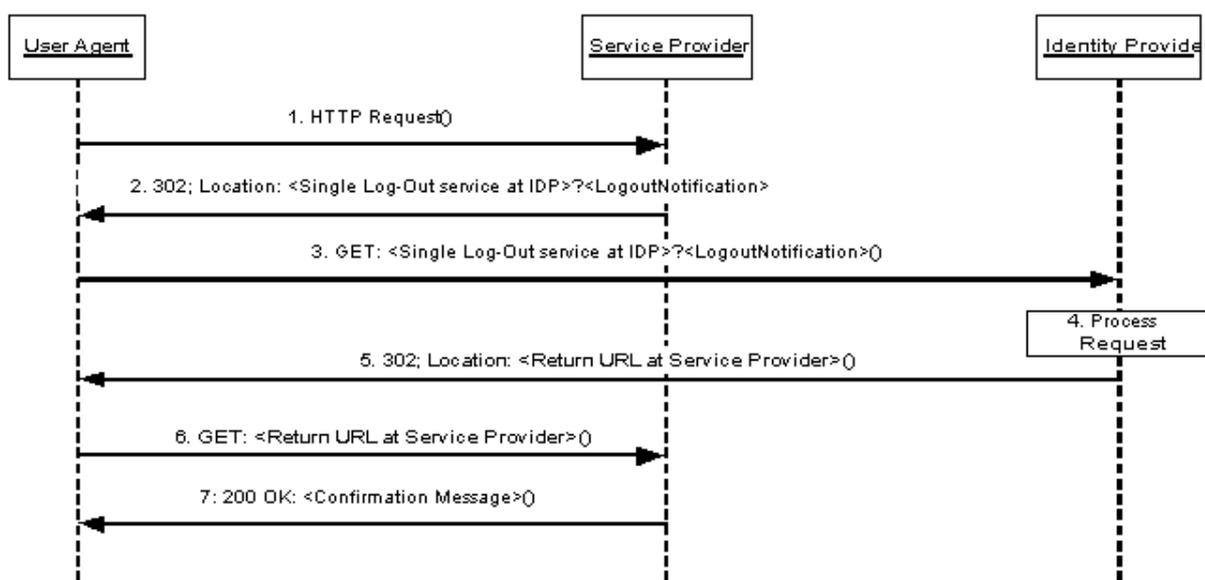
1388 The profiles in 3.5.2.1 and 3.5.2.2 are specific to when the Principal initiates the single logout
1389 notification process at the service provider.

1390 **3.5.2.1 HTTP-Redirect-Based Profile**

1391 The HTTP-redirect-based profile relies on using a HTTP 302 redirect to communicate a logout
1392 notification with the identity provider. The identity provider will then communicate a logout
1393 notification to each service provider with which it has established a session for the Principal using
1394 the service provider's preferred profile for logout notification from the identity provider (see 3.5.1).
1395 See Figure 12.

1396 The following URI-based identifier **MUST** be used when referencing this specific profile:

1397 URI: `http://projectliberty.org/profiles/slo-sp-http`



1398

1399 **Figure 12: HTTP-redirect-based profile for single logout initiated at service provider**

1400 Note: Step 4 may involve an iterative process by the identity provider to implement the preferred
1401 profile for logout notification for each service provider that has been issued authentication assertions
1402 during the Principal's current session.

1403 **Step 1: Accessing the Single Logout Service at the Service Provider**

1404 In step 1, the user agent accesses the single logout service URL at the service provider indicating that
1405 session logout is desired at the associated identity provider and all service providers for which this
1406 identity provider has provided authentication assertions during the Principal's current session. If a

1407 current session exists for the Principal at the service provider, it is RECOMMENDED that the service
1408 provider terminate that session prior to step 2.

1409 **Step 2: Redirecting to the Single Logout Service at the Identity Provider**

1410 In step 2, the service provider's single logout service responds and redirects the user agent to the
1411 single logout service URL at the identity provider.

1412 The redirection MUST adhere to the following rules:

- 1413 • The Location HTTP header MUST be set to the identity provider's single logout service
1414 URL.
- 1415 • The identity provider's single logout service URL MUST specify `https` as the URL
1416 scheme; if another scheme is specified, the service provider MUST NOT redirect to the
1417 identity provider.
- 1418 • The Location HTTP header MUST include a `<query>` component containing the
1419 `<lib:LogoutNotification>` protocol message as defined in [[LibertyProtSchema](#)] with
1420 formatting as specified in 3.1.2.

1421 Note: Additionally, the URL-encoded `<lib:LogoutNotification>` message
1422 MAY also include a parameter named `RELAYSTATE` with a value set to the URL
1423 (and/or other information) to be used by the identity provider in the HTTP response
1424 to the user agent at the completion of federation termination in step 7.

1425 The HTTP response MUST take the following form:

```
1426 <HTTP-Version> 302 <Reason Phrase>  
1427 <other headers>  
1428 Location : https://<Identity Provider single log-out service URL>?<query>  
1429 <other HTTP 1.0 or 1.1 components>
```

1430 where

1431 `<Identity Provider single log-out service URL>`

1432 This element provides the host name, port number, and path components of the single logout
1433 service URL at the identity provider.

1434 `<query>= ...<URL-encoded LogoutNotification>...`

1435 The `<query>` MUST contain a single logout notification request.

1436 **Step 3: Accessing the Identity Provider Single Logout Service**

1437 In step 3, the user agent accesses the identity provider's single logout service URL with the
1438 `<lib:LogoutNotification>` information attached to the URL fulfilling the redirect request.

1439 **Step 4: Processing the Request**

1440 In step 4, the identity provider MUST process the `<lib:LogoutNotification>` according to the
1441 rules defined in [[LibertyProtSchema](#)].

1442 Each service provider for which the identity provider has provided authentication assertions during
1443 the Principal's current session MUST be notified via the service provider's preferred profile for
1444 logout notification from the identity provider (see 3.5.1).

1445 The identity provider's current session with the Principal MUST be terminated, and no more
1446 authentication assertions for the Principal are to be given to service providers.

1447 **Step 5: Redirecting to the Service Provider Return URL**

1448 In step 5, the identity provider's single logout service responds and redirects the user agent back to
1449 service provider using the return URL location obtained from the
1450 SingleLogoutServiceReturnURL metadata element. If the URL-encoded
1451 <lib:LogoutNotification> message received in step 3 contains a parameter named
1452 RELAYSTATE, then the identity provider MUST include a <query> component containing the same
1453 RELAYSTATE parameter and its value in its response to the service provider.

1454 No success or failure message should be conveyed in this HTTP redirect. The sole purpose of this
1455 redirect is to return the user agent to the service provider.

1456 The HTTP response MUST take the following form:

```
1457 <HTTP-Version> 302 <Reason Phrase>  
1458 <other headers>  
1459 Location : https://<Service Provider Return URL>?<query>  
1460 <other HTTP 1.0 or 1.1 components>
```

1461 where

1462 <Service Provider Return URL>

1463 This element provides the host name, port number, and path components of the return URL
1464 location at the service provider.

1465 <query>= ..RELAYSTATE=<...>

1466 The <query> component MUST contain the identical RELAYSTATE parameter and its value that
1467 was received in the URL-encoded logout notification message obtained in step 3. If no
1468 RELAYSTATE parameter was provided in the step 3 message, then a RELAYSTATE parameter
1469 MUST NOT be specified in the <query> component.

1470 **Step 6: Accessing the Service Provider Return URL**

1471 In step 6, the user agent accesses the service provider's return URL location fulfilling the redirect
1472 request.

1473 **Step 7: Confirmation**

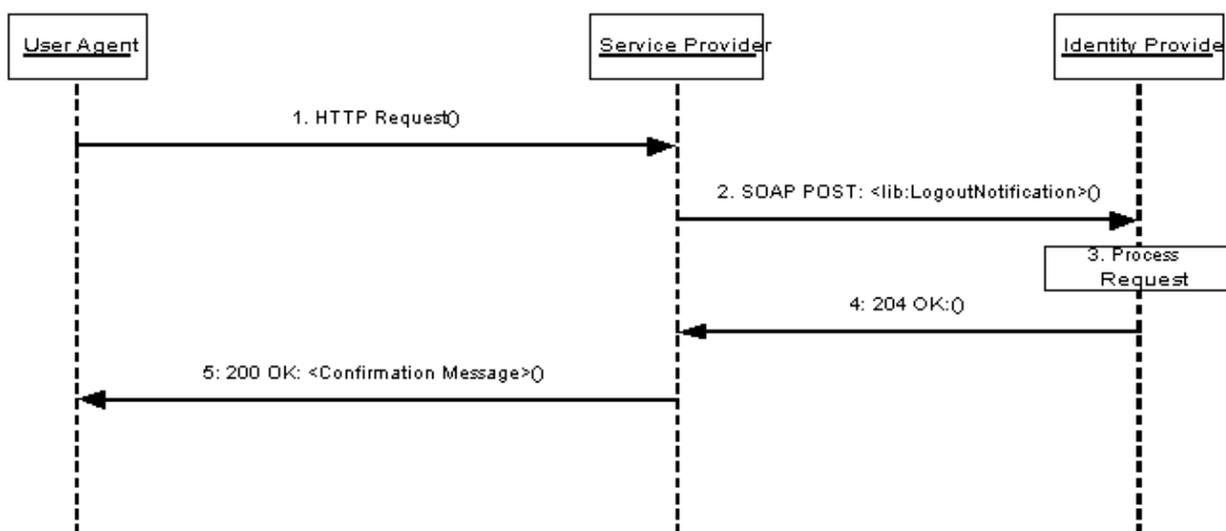
1474 In step 7, the user agent is sent an HTTP response that confirms the requested action of single logout
1475 has completed.

1476 **3.5.2.2 SOAP/HTTP-Based Profile**

1477 The SOAP/HTTP-based profile relies on using asynchronous SOAP over HTTP messages to
1478 communicate a logout notification with the identity provider. The identity provider will then
1479 communicate a logout notification to each service provider it has established a session with for the
1480 Principal via the service provider's preferred profile for logout notification from the identity provider
1481 (see 3.5.1). See Figure 13.

1482 The following URI-based identifier MUST be used when referencing this specific profile:

1483 URI: <http://projectliberty.org/profiles/slo-sp-soap>



1484

1485

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

1486 Note: Step 3 may involve an iterative process by the identity provider to implement the preferred
 1487 profile for logout notification for each service provider that has been issued authentication assertions
 1488 during the Principal’s current session.

1489 **Step 1: Accessing Single Logout Service**

1490 In step 1, the user agent accesses the single logout service URL at the service provider via an HTTP
 1491 request.

1492 **Step 2: Notification of Logout**

1493 In step 2, the service provider sends an asynchronous SOAP over HTTP notification message to the
 1494 identity provider’s SOAP endpoint. The SOAP message MUST contain exactly one
 1495 <lib:logoutNotification> element in the SOAP body and adhere to the construction rules as
 1496 defined in [[LibertyProtSchema](#)].

1497 If a SOAP fault occurs, the service provider SHOULD employ best effort to resolve the fault
 1498 condition and resend the single logout notification message to the identity provider.

1499 **Step 3: Processing the Notification**

1500 In step 3, the identity provider MUST process the <lib:LogoutNotification> according to the
 1501 rules defined in [[LibertyProtSchema](#)].

1502 Each service provider for which the identity provider has provided authentication assertions during
 1503 the Principal’s current session MUST be notified via the service provider’s preferred profile for
 1504 logout notification from the identity provider. If any of the active service provider’s preferred single
 1505 logout profile is HTTP-redirect-based or HTTP-GET-based, then the identity provider MUST
 1506 respond to the sender of the <lib:LogoutNotification> message in step 2 with a SOAP fault.
 1507 The SOAP fault MUST adhere to the following rules;

- 1508 • The FaultCode MUST specify “Server.”

- 1509 • The `FaultString` must specify “Cannot execute Single Log Out using web service.”

1510 This SOAP fault condition will indicate to the service provider that it **MUST** redirect the user agent
1511 to the identity provider to achieve single logout.

1512 The identity provider’s current session with the Principal **MUST** be terminated, and no more
1513 authentication assertions for the Principal are to be given to service providers.

1514 **Step 4: Responding to the Notification**

1515 In step 4, the identity provider **MUST** respond to the `<lib:LogoutNotification>` with a HTTP
1516 204 OK response, unless a SOAP error was sent as specified in step 3.

1517 **Step 5: Confirmation**

1518 In step 5, the user agent is sent an HTTP response that confirms the requested action of single logout
1519 was completed.

1520 **3.6 Identity Provider Introduction**

1521 This section defines the profiles by which a service provider discovers which identity providers a
1522 Principal is using. In identity federation networks having more than one identity provider, service
1523 providers need a means to discover which identity providers a Principal uses. The introduction
1524 profile relies on a cookie that is written in a domain that is common between identity providers and
1525 service providers in an identity federation network. The domain that the identity federation network
1526 predetermines for a deployment is known as the *common domain* in this specification, and the cookie
1527 containing the list of identity providers is known as the *common domain cookie*.

1528 Implementation of this profile is **OPTIONAL**. Whether identity providers and service providers
1529 implement this profile is a policy and deployment issue outside the scope of this specification. Also,
1530 which entities host web servers in the common domain is a deployment issue and is outside the scope
1531 of this specification.

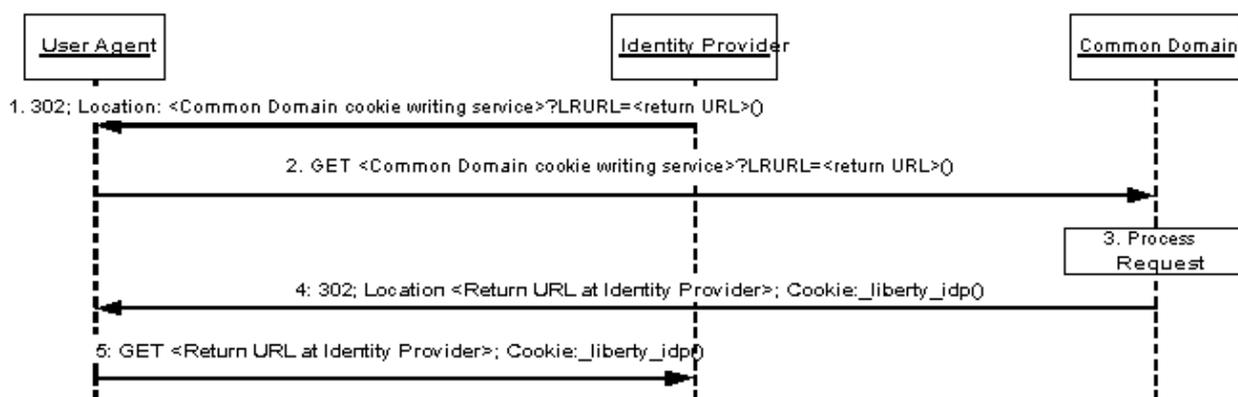
1532 **3.6.1 Common Domain Cookie**

1533 The name of the cookie **MUST** be `_liberty_idp`. The format of the cookie content **MUST** be a list
1534 of base64-encoded (see [\[RFC2045\]](#)) identity provider IDs separated by a single white space
1535 character. The identity provider IDs **MUST** adhere to the creation rules as defined in 3.2.2.2. The
1536 identity provider ID is a metadata element, as described in 3.1.3 and defined in [\[LibertyProtSchema\]](#).

1537 The cookies **MAY** be either session or persistent (see [\[RFC2109\]](#)), the implementation of which is a
1538 policy and deployment issue of the identity federation network.

1539 **3.6.2 Setting the Common Domain Cookie**

1540 After an identity provider authenticates a Principal, it **MAY** set the common domain cookie. See
1541 Figure 14. The profile for setting the cookie follows the figure.



1542

1543

Figure 14: Setting the common domain cookie

1544 **Step 1: Redirecting to Cookie Writing Service**

1545 In step 1, the identity provider redirects the user agent to a cookie writing service on a HTTP server
1546 in the common domain. The redirection MUST adhere to the following rules:

- 1547 • The Location HTTP header MUST be set to a URL hosting the cookie writing service in the
1548 common domain.
- 1549 • The common domain’s cookie writing service URL MUST specify https as the URL
1550 scheme; if another scheme is specified, the identity provider MUST NOT redirect to the
1551 common domain.
- 1552 • The Location HTTP header MUST include a <query> parameter LRURL, the value of which
1553 MUST be a URL-encoded URL that the cookie writing service will use as the location of a
1554 redirect back to the identity provider.

1555 Note: For a more seamless experience, the Return-URL parameter can be the
1556 location of a redirect back to the service provider.

1557 The HTTP response MUST take the following form:

```

1558 <HTTP-Version> 302 <Reason Phrase>
1559 <other headers>
1560 Location : https://<Common Domain cookie writing service URL>?<query>
1561 <other HTTP 1.0 or 1.1 components>
    
```

1562 where

1563 <Common Domain cookie writing service URL>

1564 This element provides the host name, port number, and path components of the cookie writing
1565 service URL on the HTTP server in the common domain.

1566 <query>= ..LRURL=<return URL>

1567 The return URL description MUST be included in the <query> component to indicate the URL
1568 to use in the redirect back to the identity provider.

1569 **Step 2: Accessing Cookie Writing Service**

1570 In step 2, the user agent accesses the common domain cookie writing service with the return URL
1571 (LRURL) information attached to the URL fulfilling the redirect request.

1572 The common domain cookie `_liberty_idp` is presented if it exists.

1573 **Step 3: Process the Request**

1574 In step 3, the common domain cookie writing service SHOULD append the identity provider ID to
1575 the list. If the identity provider ID is already present in the list, it MAY remove and append it when
1576 authentication of the Principal occurs. The intent is that the most recently established identity
1577 provider session is the last one in the list.

1578 **Step 4: Redirecting to Identity Provider Return URL**

1579 In step 4, the common domain cookie writing service responds with a redirection of the user agent
1580 back to the Identity provider using the LRURL specified in step 2.

1581 The redirection MUST adhere to the following rules:

- 1582 • The Location HTTP header MUST be set to the return URL for the identity provider.
- 1583 • The Set-Cookie header MUST be included in the response, specifying `_liberty_idp` as
1584 the name.
- 1585 • The identity provider return URL MUST specify `https` as the URL scheme; if another
1586 scheme is specified, the common domain cookie writing service MUST NOT redirect to the
1587 identity provider.

1588 The HTTP response MUST take the following form:

```
1589 <HTTP-Version> 302 <Reason Phrase>  
1590 Set-Cookie: _liberty_idp=<value>;Version="1";Domain"...";Max-Age=...  
1591 <other headers>  
1592 Location : https://<Identity Provider Return URL>  
1593 <other HTTP 1.0 or 1.1 components>
```

1594 where

1595 <Identity Provider Return URL>

1596 This element provides the host name, port number, and path components of the return URL at the
1597 identity provider.

1598 Set-Cookie:

1599 The Set-Cookie header MUST include the cookie `_liberty_idp` with its value as previously
1600 defined in 3.6.1. The cookie MUST be marked as `secure`.

1601

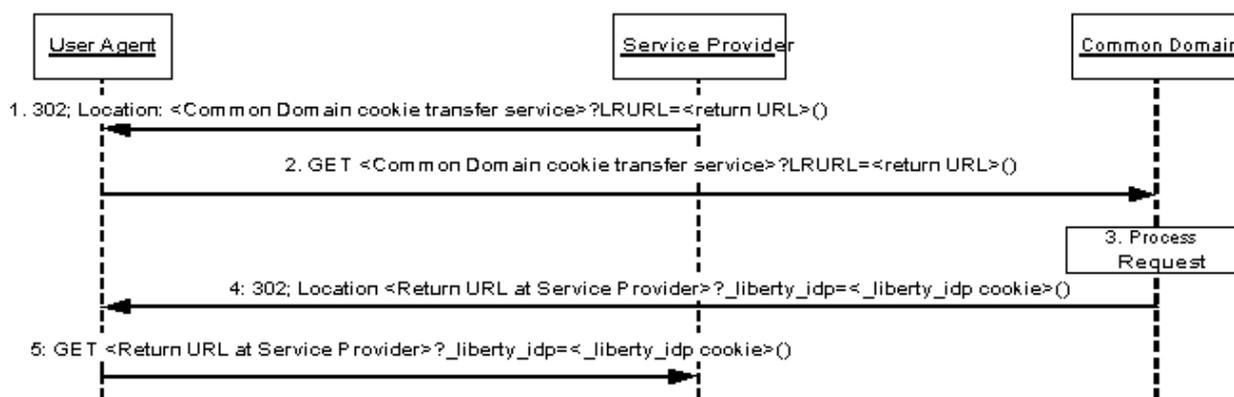
1602 **Step 5: Accessing Identity Provider Return URL**

1603 In step 5, the user agent accesses the identity provider's return URL and fulfills the redirect request.

1604 **3.6.3 Obtaining the Common Domain Cookie**

1605 When a service provider needs to discover which identity providers the Principal uses, it invokes a
1606 protocol exchange designed to present the common domain cookie to the service provider after it is
1607 read by an HTTP server in the common domain. See Figure 15.

1608 If the HTTP server in the common domain is operated by the service provider, the service provider
1609 MAY redirect the user agent to an identity provider's intersite transfer service for an optimized
1610 single sign-on process.



1611

1612

Figure 15: Obtaining the common domain cookie

1613 Step 1: Redirecting to Cookie Transfer Service

1614 In step 1, the service provider redirects the user agent to a cookie transfer service on a HTTP server
1615 in the common domain.

1616 The redirection MUST adhere to the following rules:

- 1617 • The Location HTTP header MUST be set to a URL hosting the cookie transfer service in the
1618 common domain.
- 1619 • The common domain's cookie transfer service URL MUST specify https as the URL
1620 scheme; if another scheme is specified, the service provider MUST NOT redirect to the
1621 common domain.
- 1622 • The Location HTTP header MUST include a <query> parameter LRURL, the value of which
1623 MUST be a URL-encoded URL that the cookie transfer service will use as the location of a
1624 redirect back to the service provider.

1625 The HTTP response MUST take the following form:

```

1626 <HTTP-Version> 302 <Reason Phrase>
1627 <other headers>
1628 Location : https://<Common Domain cookie transfer service URL>?<query>
1629 <other HTTP 1.0 or 1.1 components>
    
```

1630 where

1631 <Common Domain cookie writing service URL>

1632 This element provides the host name, port number, and path components of the cookie transfer
1633 service URL on the HTTP server in the common domain.

1634 <query>= ...LRURL=<return URL>

1635 The return URL description MUST be included in the <query> component to indicate the URL
1636 to use in the redirect back to the service provider.

1637 Step 2: Accessing Cookie Transfer Service

1638 In step 2, the user agent accesses the common domain cookie transfer service with the return URL
1639 (LRURL) information attached to the URL fulfilling the redirect request.

1640 The common domain cookie `_liberty_idp` is presented if it exists.

1641 **Step 3: Process the Request**

1642 In step 3, the common domain cookie transfer service MUST read the `_liberty_idp` cookie and
1643 redirect the user agent back to the service provider with the value of the cookie included as a
1644 `<query>` parameter.

1645 **Step 4: Redirecting to Service Provider Return URL**

1646 In step 4, the common domain cookie transfer service responds with a redirection of the user agent
1647 back to the service provider using the `LRURL` specified in step 2. The redirection MUST adhere to the
1648 following rules:

- 1649 • The `Location` HTTP header MUST be set to return URL for the service provider.
- 1650 • The common domain's cookie writing service URL MUST specify `https` as the URL
1651 scheme; if another scheme is specified, the common domain cookie transfer service MUST
1652 NOT redirect to the service provider.
- 1653 • The `Location` HTTP header MUST include a `<query>` parameter `_liberty_idp`, whose
1654 value MUST be the value of the `_liberty_idp` cookie.

1655 The HTTP response MUST take the following form:

```
1656 <HTTP-Version> 302 <Reason Phrase>  
1657 <other headers>  
1658 Location : https://<Service Provider Return URL>?<query>  
1659 <other HTTP 1.0 or 1.1 components>
```

1660 where

1661 `<Service Provider Return URL>`

1662 This element provides the host name, port number, and path components of the return URL at the
1663 service provider.

1664 `<query>= ..._liberty_idp=<_liberty_idp cookie>`

1665 The `_liberty_idp` cookie value MUST be included in the `<query>` component to indicate the
1666 list of identity providers to the service provider.

1667 **Step 5: Accessing Service Provider Return URL**

1668 In step 5, the user agent accesses the service provider's return URL with the `_liberty_idp` cookie
1669 information attached to the URL and fulfills the redirect request.

1670 **4 Security Considerations**

1671 **4.1 Introduction**

1672 This section describes security considerations associated with Liberty protocols for identity
1673 federation, single sign-on, federation termination, and single logout.

1674 Liberty protocols, schemas, bindings, and profiles inherit and heavily utilize the SAML protocols.
1675 Therefore, the security considerations published along with the SAML specification have direct
1676 relevance (see [[SAMLCore](#)], [[SAMLBind](#)], and [[SAMLSec](#)]). Throughout this section if, for any
1677 reason, a specific consideration or countermeasure does not apply or differs, notice of this fact is
1678 made; and a description of alternatives is supplied, where possible.

1679 4.2 General Requirements

1680 4.2.1 Security of SSL and TLS

1681 SSL and TLS utilize a suite of possible cipher suites. The security of the SSL or TLS session
1682 depends on the chosen cipher suite. An entity (that is, a user agent, service provider, or identity
1683 provider) that terminates an SSL or TLS connection needs to offer (or accept) suitable cipher suites
1684 during the handshake. The following list of TLS 1.0 cipher suites (or their SSL 3.0 equivalent) is
1685 recommended.

- 1686 • TLS_RSA_WITH_RC4_128_SHA
- 1687 • TLS_RSA_WITH_3DES_EDE_CBC_SHA
- 1688 • TLS_DHE_DSS_WITH_3DES_EDE_CBC_SHA

1689 The above list is by no means exhaustive. The recommended cipher suites are among the most
1690 commonly used. Note: New cipher suites are added as they are standardized and should be
1691 considered for inclusion if they have sufficiently strong security properties. For example, it is
1692 anticipated that the AES-based cipher suites being standardized in the IETF will be widely adopted
1693 and deployed.

1694 4.2.2 Security Implementation

1695 The suitable implementation of security protocols is necessary to maintain the security of a system,
1696 including

- 1697 • Secure random or pseudo-random number generator
- 1698 • Secure storage

1699 4.3 Classification of Threats

1700 4.3.1 Threat Model

1701 For an analysis of threat classifications, an Internet threat model has been used. In other words, the
1702 threat model assumes that intermediary and end-systems participating in Liberty protocol exchanges
1703 have not been compromised. However, where possible, the consequences and containment properties
1704 of a compromised system entity are described and countermeasures are suggested to bolster the
1705 security posture so that the exposure from a security breach is minimized.

1706 Given the nature of the Internet, the assumption is made that deployment is across the global Internet
1707 and, therefore, crosses multiple administrative boundaries. Thus, an assumption is also made that the
1708 adversary has the capacity to engage in both passive and active attacks (see 4.3.3).

1709 4.3.2 Rogue and Spurious Entities

1710 Attackers may be classified based on their capabilities and the roles that they play in launching
1711 attacks on a Liberty system as follows:

- 1712 • **Rogue Entities:** Entities that misuse their privileges. The rogue actors may be Principals,
1713 user agents, service providers, or identity providers. A rogue Principal is a legitimate
1714 participant who attempts to escalate its privileges or masquerade as another system Principal.
1715 A rogue user agent may, for instance, misuse the relationships between its associated
1716 Principals and an identity provider to launch certain attacks. Similarly, a rogue service

1717 provider may be able to exploit the relationship that it has either with a Principal or with an
1718 identity provider to launch certain attacks.

- 1719 • **Spurious Entities:** Entities that masquerade as a legitimate entity or are completely
1720 unknown to the system. The spurious actors include Principals, user agents (i.e., user agents
1721 without associated legitimate Liberty Principals), service providers, or identity providers. A
1722 spurious service provider may, for instance, pretend to be a service provider that has a
1723 legitimate relationship with an identity provider. Similarly, a spurious Principal may be one
1724 that pretends to be a legitimate Principal that has a relationship with either a service provider
1725 or an identity provider.

1726 4.3.3 Active and Passive Attackers

1727 Both rogue and spurious entities may be able to launch active or passive attacks on the system. A
1728 passive attack is one where the attacker does not inject any traffic or modify any traffic in any way.
1729 Such an attacker usually passively monitors the traffic flow, and the information that is obtained in
1730 that flow may be used at a later time. An active attacker, on the other hand, is capable of modifying
1731 existing traffic as well as injecting new traffic into the system.

1732 4.3.4 Scenarios

1733 The following scenarios describe possible attacks:

- 1734 • **Collusion: The secret cooperation between two or more Liberty entities to launch an**
1735 **attack, for example,**
 - 1736 • Collusion between Principal and service provider
 - 1737 • Collusion between Principal and identity provider
 - 1738 • Collusion between identity provider and service provider
 - 1739 • Collusion among two or more Principals
 - 1740 • Collusion between two or more service providers
 - 1741 • Collusion between two or more identity providers
- 1742 • **Denial-of-Service Attacks:** The prevention of authorized access to a system resource or the
1743 delaying of system operations and functions.
- 1744 • **Man-in-the-Middle Attacks:** A form of active wiretapping attack in which the attacker
1745 intercepts and selectively modifies communicated data to masquerade as one or more of the
1746 entities involved in a communication association.
- 1747 • **Replay Attacks:** An attack in which a valid data transmission is maliciously or fraudulently
1748 repeated, either by the originator or by an adversary who intercepts the data and retransmits
1749 it, possibly as part of a masquerade attack.
- 1750 • **Session Hijacking:** A form of active wiretapping in which the attacker seizes control of a
1751 previously established communication association.

1752 4.4 Threat Scenarios and Countermeasures

1753 In this section, threats that may apply to all the Liberty profiles are considered first. Then threats that
1754 are specific to individual profiles are considered. In each discussion the threat is described as well as

1755 what countermeasures exist in the profile or what additional countermeasures may be implemented
1756 to mitigate the threat.

1757 **4.4.1 Common Threats for All Profiles**

1758 **Threat:** Request messages sent in cleartext

1759 **Description:** Most profile protocol exchanges do not mandate that all exchanges commence
1760 over a secure communication channel. This lack of transport security potentially exposes
1761 requests and responses to both passive and active attacks.

1762 One obvious manifestation is when the initial contact is not over a secure transport and the
1763 Liberty profile begins to exchange messages by carrying the request message back to the user
1764 agent in the location header of a redirect.

1765 Another such manifestation could be a request or response message which carries a URI that
1766 may be resolved on a subsequent exchange, for instance `lib:AuthnContextClassRef`. If
1767 this URI were to specify a less or insecure transport, then the exchange may be vulnerable to
1768 the types of attacks described above.

1769 **Countermeasure:** Ensure that points of entry to Liberty protocol exchanges utilize the
1770 `https` URL `<scheme>` and that all interactions for that profile consistently exchange
1771 messages over `https`.

1772 **Threat:** Malicious redirects into identity or service provider targets

1773 **Description:** A spurious entity could issue a redirect to a user agent so that the user agent
1774 would access a resource that disrupts single sign-on. For example, an attacker could redirect
1775 the user agent to a logout resource of a service provider causing the Principal to be logged out
1776 of all existing authentication sessions.

1777 **Countermeasure:** Access to resources that produce side effects could be specified with a
1778 transient qualifier that must correspond to the current authentication session. Alternatively, a
1779 confirmation dialog could be interposed that relies on a transient qualifier with similar
1780 semantics.

1781 **Threat:** Relay state tampering or fabrication

1782 **Description:** Some of the messages may carry a `<lib:RelayState>` element, which is
1783 recommended to be integrity-protected by the producer and optionally confidentiality-
1784 protected. If these practices are not followed, an adversary could trigger unwanted side
1785 effects. In addition, by not confidentiality-protecting the value of this element, a legitimate
1786 system entity could inadvertently expose information to the identity provider or a passive
1787 attacker.

1788 **Countermeasure:** Follow the recommended practice of confidentiality- and integrity-
1789 protecting the `<lib:RelayState>` data. Note: Because the value of this element is both
1790 produced and consumed by the same system entity, symmetric cryptographic primitives could
1791 be utilized.

1792 **4.4.2 Single Sign-On and Federation**

1793 **4.4.2.1 Common Interactions for All Single Sign-On and Federation Profiles**

1794 **Threat:** `<lib:AuthnRequest>` sent over insecure channel

1795 **Description:** It is recommended that the initial exchange to access the intersite transfer
1796 service be conducted over a TLS-secured transport. Not following this recommendation can
1797 expose the exchange to both passive and active attacks.

1798 **Countermeasure:** Deploy the intersite transfer service under an https scheme.

1799 **Threat:** Unsigned <lib:AuthnRequest> message

1800 **Description:** The signature element of an <lib:AuthnRequest> is optional and thus the
1801 absence of the signature could pose a threat to the identity provider or even the targeted
1802 service provider. For example, a spurious system entity could generate an unsigned
1803 <lib:AuthnRequest> and redirect the user agent to the identity provider. The identity
1804 provider must then consume resources.

1805 **Countermeasure:** Sign the <lib:AuthnRequest>.

1806 **Threat:** Replay of an authentication assertion

1807 **Description:** After obtaining a valid assertion from an identity provider, either legitimately or
1808 surreptitiously, the entity replays the assertion to the Service at a later time. A digital
1809 signature must cover the entire assertion, thus elements within the assertion cannot be
1810 corrupted without detection during the mandatory verification step. However, it is possible to
1811 fabricate an <lib:AuthnResponse> with the valid assertion.

1812 **Countermeasure:** The issuer should sign <lib:AuthnResponse> messages. Signing binds
1813 the <samlp:IssueInstant> of the response message to the assertion it contains. This
1814 binding accords the relying party the opportunity to temporally judge the response.
1815 Additionally, a valid signature over the response binds the <samlp:InResponseTo>
1816 element to the corresponding <lib:AuthnRequest>. (Specifying a short period that the
1817 authentication assertion can be relied upon will minimize, but not mitigate this threat.
1818 Binding the <lib:AssertionId> to the request/<samlp:InResponseTo> element may
1819 also be handy.)

1820 **Threat:** Fabricated <lib:AuthnResponse> denial of service

1821 **Description:** An attacker captures the <samlp:RequestID> sent in an
1822 <lib:AuthnRequest> message by a service provider to an identity provider, and sends
1823 several spurious <lib:AuthnResponse> messages to the service provider with the same
1824 <samlp:InResponseTo>. Because the <samlp:InResponseTo> matches a
1825 <samlp:RequestID> that the service provider had used, the service provider goes through
1826 the process of validating the signature in the message. Thus, it is subject to a denial of service
1827 attack.

1828 **Countermeasure:** A secure communication channel should be established before
1829 transferring requests and responses.

1830 **Threat:** Collusion between two Principals

1831 **Description:** After getting an artifact or <lib:AuthnResponse> in step 6 (see 3.2), a
1832 legitimate Principal A could pass this artifact or <lib:AuthnResponse> on to another
1833 Principal, B. Principal B is now able to use the artifact or <lib:AuthnResponse>, while
1834 the actual authentication happened via Principal A.

1835 **Countermeasure:** Implementations where this threat is a concern MUST use the
1836 <saml:AuthenticationLocality> in the authentication statement. The IP address that
1837 Principal B uses would be different from the IP address within the
1838 <saml:AuthenticationLocality>. This countermeasure may not suffice when the user

1839 agent is behind a firewall or proxy server. IP spoofing may also circumvent this
1840 countermeasure.

1841 **Threat:** Stolen artifact and subsequent Principal impersonation

1842 **Description:** See Section 4.1.1.9.1 in [[SAMLBind](#)]

1843 **Countermeasure:** Identity providers MUST enforce a policy of one-time retrieval of the
1844 assertion corresponding to an artifact so that a stolen artifact can be used only once.
1845 Implementations where this threat is a concern MUST use the
1846 `<saml:AuthenticationLocality>` in the authentication statement. The IP address of a
1847 spurious user agent that attempts to use the stolen artifact would be different from IP address
1848 within the `<saml:AuthenticationLocality>`. The service provider may then be able to
1849 detect that the IP addresses differ. This countermeasure may not suffice when the user agent
1850 is behind a firewall or proxy server. IP address spoofing may also circumvent this
1851 countermeasure.

1852 **Threat:** Stolen assertion and subsequent Principal impersonation

1853 **Description:** See Section 4.1.1.9.1 in [[SAMLBind](#)]

1854 **Countermeasure:** Refer to the previous threat for requirements.

1855 **Threat:** Rogue service provider uses artifact or assertion to impersonate Principal at a different
1856 service provider

1857 **Description:** Because the `<lib:AuthnResponse>` contains the `<lib:ProviderID>`, this
1858 threat is not possible.

1859 **Countermeasure:** None

1860 **Threat:** Rogue identity provider impersonates Principal at a service provider

1861 **Description:** Because the Principal trusts the identity provider, it is assumed that the identity
1862 provider does not misuse the Principal's trust.

1863 **Countermeasure:** None

1864 4.4.2.2 Federation

1865 **Threat:** Collusion among service providers can violate privacy of the Principal

1866 **Description:** When a group of service providers collude to share the
1867 `<lib:IDPProvidedNameIdentifier>` of a Principal among themselves, they can track
1868 the Principal's behavior among them.

1869 **Countermeasure:** The `<lib:IDPProvidedNameIdentifier>` is required to be unique
1870 for each identity provider to service provider relationship.

1871 **Threat:** Poorly generated name identifiers may compromise privacy

1872 **Description:** The federation protocol mandates that the `<lib:NameIdentifier>` elements
1873 be unique within a Principal's federated identities. The name identifiers exchanged are
1874 pseudonyms and, to maintain the privacy of the Principal, should be resistant to guessing or
1875 derivation attacks.

1876 **Countermeasure:** Name identifiers should be constructed using pseudo-random values that
1877 have no discernable correspondence with the Principal's identifier (or name) used by the
1878 entity that generates the name identifier.

1879 **4.4.3 Name Registration**

1880 No known threats.

1881 **4.4.4 Federation Termination: HTTP-Redirect-Based Profile**

1882 **Threat:** Attacker can monitor and disrupt termination

1883 **Description:** During the initial steps, a passive attacker can collect the
1884 `<lib:FederationTerminationNotification>` information when it is issued in the
1885 redirect. This threat is possible because the first and second steps are not required to use
1886 `https` as the URL scheme. An active attacker may be able to intercept and modify the
1887 message conveyed in step 2 because the digital signature only covers a portion of the
1888 message. This initial exchange also exposes the name identifier. Exposing these data poses a
1889 privacy threat.

1890 **Countermeasure:** All exchanges should be conducted over a secure transport such as SSL or
1891 TLS.

1892 **4.4.5 Single Logout: HTTP-Redirect-Based Profile**

1893 **Threat:** Passive attacker can collect a Principal's name identifier

1894 **Description:** During the initial steps, a passive attacker can collect the
1895 `<lib:LogoutNotification>` information when it is issued in the redirect. Exposing these
1896 data poses a privacy threat.

1897 **Countermeasure:** All exchanges should be conducted over a secure transport such as SSL or
1898 TLS.

1899 **Threat:** Unsigned `<lib:LogoutNotification>` message

1900 **Description:** An Unsigned `<lib:LogoutNotification>` could be injected by a spurious
1901 system entity thus denying service to the Principal. Assuming that the `NameIdentifier` can
1902 be deduced or derived then it is conceivable that the user agent could be directed to deliver a
1903 fabricated `<lib:LogoutNotification>` message.

1904 **Countermeasure:** Sign the `<lib:LogoutNotification>` message.

1905 **4.4.6 Identity Provider Introduction**

1906 No known threats.

1907 **5 References**

1908 [Anders] Rundgren, A., "A suggestion on how to implement SAML browser bindings
1909 without using Artifacts," [http://www.x-obi.com/OBI400/andersr-browser-](http://www.x-obi.com/OBI400/andersr-browser-artifact.ppt)
1910 [artifact.ppt](http://www.x-obi.com/OBI400/andersr-browser-artifact.ppt), August 2001.

1911 [HTML4] Raggett, D., Le Hors, A., Jacobs, I., "HTML 4.01 Specification,"
1912 <http://www.w3.org/TR/html401>, W3C, December 1999.

1913 [LibertyGloss] Mauldin, H., "Liberty Glossary," [https://66.34.4.93/members/technology](https://66.34.4.93/members/technology-expert-group/draft-liberty-tech-glossary-07.pdf)
1914 [expert-group/draft-liberty-tech-glossary-07.pdf](https://66.34.4.93/members/technology-expert-group/draft-liberty-tech-glossary-07.pdf), April 2002.

- 1915 [LibertyProtSchema] Beatty, J., "Liberty Protocols and Schemas Specification,"
1916 [https://66.34.4.93/members/technology_20expert_group/architecture/draft-](https://66.34.4.93/members/technology_20expert_group/architecture/draft-liberty-architecture-protocols-schemas-08.pdf)
1917 [liberty-architecture-protocols-schemas-08.pdf](https://66.34.4.93/members/technology_20expert_group/architecture/draft-liberty-architecture-protocols-schemas-08.pdf), April 2002.
- 1918 [Rescorla-Sec] Rescorla, E., et al., "Guidelines for Writing RFC Text on Security
1919 Considerations," [http://www.ietf.org/internet-drafts/draft-rescorla-sec-](http://www.ietf.org/internet-drafts/draft-rescorla-sec-cons-04.txt)
1920 [cons-04.txt](http://www.ietf.org/internet-drafts/draft-rescorla-sec-cons-04.txt), February 2002.
- 1921 [RFC1750] Eastlake, D., Croker, S., Schiller, J., "Randomness Recommendations for
1922 Security," <ftp://ftp.isi.edu/in-notes/rfc1750.txt>, RFC 1750, December 1994
- 1923 [RFC1945] Berners-Lee, T., Fielding, R., Frystyk, H., "Hypertext Transfer Protocol --
1924 HTTP/1.0," <ftp://ftp.isi.edu/in-notes/rfc1945.txt>, RFC 1945, May 1996.
- 1925 [RFC2045] Freed, N., Borenstein, N., "Multipurpose Internet Mail Extensions (MIME)
1926 Part One: Format of Internet Message Bodies," [ftp://ftp.isi.edu/in-](ftp://ftp.isi.edu/in-notes/rfc2045.txt)
1927 [notes/rfc2045.txt](ftp://ftp.isi.edu/in-notes/rfc2045.txt), RFC 2045, November 1996.
- 1928 [RFC2109] Kristol, D., Montulli, L., "HTTP State Management Mechanism,"
1929 <ftp://ftp.isi.edu/in-notes/rfc2109.txt>, RFC 2109, February 1997.
- 1930 [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels,"
1931 <ftp://ftp.isi.edu/in-notes/rfc2119.txt>, RFC 2119, March 1997.
- 1932 [RFC2246] Dierks, T., Allen, C., "The TLS Protocol Version 1.0," [ftp://ftp.isi.edu/in-](ftp://ftp.isi.edu/in-notes/rfc2246.txt)
1933 [notes/rfc2246.txt](ftp://ftp.isi.edu/in-notes/rfc2246.txt), RFC 2246, January 1999.
- 1934 [RFC2396] Berners-Lee, T., Fielding, R., Masinter, L., "Uniform Resource Identifiers
1935 (URI): Generic Syntax," <ftp://ftp.isi.edu/in-notes/rfc2396.txt>, RFC 2396,
1936 August 1998.
- 1937 [RFC2616] Fielding, R., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P.,
1938 Berners-Lee, T., "Hypertext Transfer Protocol -- HTTP/1.1,"
1939 <ftp://ftp.isi.edu/in-notes/rfc2616.txt>, RFC 2616, June 1999.
- 1940 [RFC2617] Franks, J., Hallam-Baker, P., Hostetler, J., Lawrence, S., Leach, P., Luotonen,
1941 A., Stewart, L., "HTTP Authentication: Basic and Digest Access
1942 Authentication," <ftp://ftp.isi.edu/in-notes/rfc2617.txt>, RFC 2617, June 1999.
- 1943 [RFC2774] Nielsen, H., Leach P., Lawrence, S., "An HTTP Extension Framework,"
1944 <ftp://ftp.isi.edu/in-notes/rfc2774.txt>, RFC 2774, February 2000.
- 1945 [RFC3106] Eastlake, D., Goldstein, T., "ECML v1.1: Field Specifications for E-
1946 Commerce," <ftp://ftp.isi.edu/in-notes/rfc3106.txt>, RFC 3106, April 2001
- 1947 [SAMLBind] Mishra, Prateek et al., "Bindings and Profiles for the OASIS Security
1948 Assertion Markup Language (SAML)," [http://www.oasis-](http://www.oasis-open.org/committees/security/docs/draft-sstc-bindings-model-14.doc)
1949 [open.org/committees/security/docs/draft-sstc-bindings-model-14.doc](http://www.oasis-open.org/committees/security/docs/draft-sstc-bindings-model-14.doc),
1950 OASIS, April 2002.
- 1951 [SAMLCore] Hallam-Baker, P., et al., "Assertions and Protocol for the OASIS Security
1952 Assertion Markup Language (SAML)," [http://www.oasis-](http://www.oasis-open.org/committees/security/docs/draft-sstc-core-31.pdf)
1953 [open.org/committees/security/docs/draft-sstc-core-31.pdf](http://www.oasis-open.org/committees/security/docs/draft-sstc-core-31.pdf), OASIS, April
1954 2002.
- 1955 [SAMLGloss] Hodges, J., et al., "Glossary for the OASIS Security Assertion Markup
1956 Language (SAML)," [http://www.oasis-](http://www.oasis-open.org/committees/security/docs/draft-sstc-glossary-02.pdf)
1957 [open.org/committees/security/docs/draft-sstc-glossary-02.pdf](http://www.oasis-open.org/committees/security/docs/draft-sstc-glossary-02.pdf), OASIS,
1958 December 2001.

- 1959 [SAMLReqs] Platt, D., et al., “SAML Requirements and Use Cases,” [http://www.oasis-](http://www.oasis-open.org/committees/security/docs/draft-sstc-saml-reqs-01.pdf)
1960 [open.org/committees/security/docs/draft-sstc-saml-reqs-01.pdf](http://www.oasis-open.org/committees/security/docs/draft-sstc-saml-reqs-01.pdf), OASIS,
1961 December 2001.
- 1962 [SAMLSec] Hodges, J., et al., “Security Considerations for the OASIS Security Assertion
1963 Markup Language (SAML),” [http://www.oasis-](http://www.oasis-open.org/committees/security/docs/draft-sec-consider-04.pdf)
1964 [open.org/committees/security/docs/draft-sec-consider-04.pdf](http://www.oasis-open.org/committees/security/docs/draft-sec-consider-04.pdf), OASIS,
1965 December 2001.
- 1966 [Schema1] Thompson, H. S., et al., “XML Schema Part 1: Structures,”
1967 <http://www.w3.org/TR/xmlschema-1/>, W3C Recommendation, May 2001.
- 1968 [Schema2] Biron, P. V., et al., “XML Schema Part 2: Datatypes,”
1969 <http://www.w3.org/TR/xmlschema-2/>, W3C Recommendation, May 2001.
- 1970 [ShibMarlena] Erdos, M., “Shibboleth Architecture DRAFT v1.1,”
1971 [http://middleware.internet2.edu/shibboleth/docs/draft-erdos-shibboleth-](http://middleware.internet2.edu/shibboleth/docs/draft-erdos-shibboleth-architecture1-01.pdf)
1972 [architecture1-01.pdf](http://middleware.internet2.edu/shibboleth/docs/draft-erdos-shibboleth-architecture1-01.pdf), June 2001.
- 1973 [SOAP1.1] Box, D., et al., “Simple Object Access Protocol (SOAP) 1.1,”
1974 <http://www.w3.org/TR/SOAP>, W3C Note, May 2000.
- 1975 [SSLv3] Freier, A. O., Karlton, P., Kocher, P., “The SSL Protocol Version 3.0,”
1976 <http://www.mozilla.org/projects/security/pki/nss/ssl/draft302.txt>, Internet
1977 Draft, November 1996.
- 1978 [WML1.3] “Wireless Application Protocol Wireless Markup Language Specification
1979 Version 1.3,” Wireless Application Protocol Forum, Ltd.,
1980 <http://www.wapforum.org/>, February 2000
- 1981 [XMLSig] Eastlake, D., Reagle, J., Solo, D., “XML-Signature Syntax and Processing,”
1982 <http://www.w3.org/TR/xmlsig-core/>, W3C Recommendation, February
1983 2002.
- 1984